Referencial de competencias profesionales

Formación Desarrollo Web Full Stack

País: Spain

Profesiones cubiertas: Desarrollo Web Full Stack

el 12/06/2026

Índice

ANEXO I Presentación sintética del marco del diploma

PRESENTACIÓN GENERAL

Nombre de la formación: Certificado de Profesionalidad en Desarrollo Web Full Stack

Tipo de certificación: Certificado de Profesionalidad

Modalidades de formación:

Formación presencial, semipresencial y a distancia con prácticas en entorno real

Finalidad y objetivos:

Capacitar al profesional para analizar requisitos funcionales y técnicos, diseñar arquitecturas web escalables, desarrollar interfaces de usuario responsivas y accesibles, implementar lógica de negocio robusta en servidor, gestionar bases de datos, realizar testing y aseguramiento de calidad, aplicar medidas de seguridad, optimizar rendimiento, desplegar en infraestructura cloud, y mantener aplicaciones web complejas. El desarrollador web full stack integra competencias de análisis, diseño, desarrollo front-end, desarrollo back-end, gestión de datos, testing, seguridad, operaciones y comunicación técnica para crear soluciones web completas y de calidad.

Contexto profesional:

El Desarrollo Web Full Stack es una disciplina profesional que requiere dominio integral de todas las capas de una aplicación web moderna. Los desarrolladores web full stack trabajan en equipos multidisciplinarios en empresas de tecnología, agencias digitales, startups y departamentos de TI de organizaciones de todos los sectores. El mercado laboral demanda profesionales capaces de gestionar el ciclo completo de desarrollo, desde la comprensión de requisitos hasta el despliegue en producción y mantenimiento continuo. Las competencias requeridas incluyen análisis y diseño de soluciones, desarrollo de interfaces responsivas, implementación de APIs REST, gestión de bases de datos, testing automatizado, seguridad web, optimización de rendimiento y operaciones en cloud. La evolución profesional permite especialización en front-end, back-end, arquitectura o liderazgo técnico.

Público objetivo y condiciones de acceso:

Profesionales con conocimientos básicos de programación, HTML, CSS y JavaScript que desean desarrollar competencias integrales en desarrollo web full stack. Personas en transición de carrera desde roles relacionados (diseño web, administración de sistemas, testing) que buscan ampliar sus habilidades técnicas. Desarrolladores junior que desean consolidar conocimientos en todas las capas de una aplicación web.

Prerrequisitos:

- Conocimientos básicos de programación (variables, funciones, control de flujo)
- Familiaridad con HTML y CSS
- Experiencia básica con JavaScript
- Comprensión de conceptos de bases de datos relacionales
- Capacidad de trabajo en equipo y comunicación técnica

Vías de acceso:

- Formación continua
- Vía escolar pública o privada concertada
- Aprendizaje
- Enseñanza a distancia

ORGANIZACIÓN DE LA FORMACIÓN

Enfoque pedagógico:

La formación utiliza una metodología activa y práctica que combina teoría con proyectos reales. Los participantes aprenden mediante análisis de casos, desarrollo de proyectos incrementales, code review colaborativo y simulación de entornos profesionales. Se enfatiza la resolución de problemas, la toma de decisiones técnicas y la comunicación en equipos multidisciplinarios.

- Aprendizaje basado en proyectos con casos de uso reales del mercado laboral
- Desarrollo incremental de una aplicación web completa a lo largo de la formación
- Sesiones de code review y feedback técnico constructivo
- Simulación de entornos de desarrollo profesionales con Git, CI/CD y cloud
- Integración de herramientas y tecnologías actuales utilizadas en la industria
- Énfasis en buenas prácticas, estándares de calidad y seguridad desde el inicio
- Colaboración en equipos pequeños reflejando dinámicas de trabajo real
- Evaluación continua mediante pruebas prácticas y proyectos incrementales

Métodos de aprendizaje:

- Clases teóricas con ejemplos prácticos
- Laboratorios prácticos con ejercicios guiados
- Proyectos individuales y en equipo
- Code review y sesiones de feedback
- Estudio de casos y análisis de arquitecturas reales
- Prácticas en entorno profesional simulado
- Documentación técnica y especificaciones
- Herramientas de debugging y análisis de rendimiento

Puntos fuertes de la formación:

- Formación integral que cubre todas las capas de una aplicación web moderna
- Énfasis en arquitectura escalable y patrones de diseño profesionales
- Integración de seguridad y testing desde las primeras etapas del desarrollo
- Experiencia práctica con tecnologías y herramientas actuales del mercado
- Desarrollo de competencias transversales de comunicación y colaboración
- Preparación para roles de desarrollador full stack, especialista front-end o back-end
- Alineación con estándares de calidad y buenas prácticas de la industria
- Capacidad de participar en decisiones arquitectónicas y de infraestructura

TABLA DE SÍNTESIS DE ACTIVIDADES - BLOQUES DE COMPETENCIAS - UNIDADES

Actividades	Bloques de competencias	Unidades
Función 1: Análisis y Diseño de Soluciones Web • Recopilar y analizar requisitos funcionales con stakeholders • Analizar requisitos técnicos y restricciones de infraestructura • Diseñar la arquitectura global de la aplicación web • Definir la estructura modular del front-end • Definir la estructura modular del back-end • Seleccionar tecnologías y frameworks adecuados	**Bloque 1 - Análisis y Diseño de Soluciones Web** • Analizar requisitos funcionales y técnicos de una solución web para identificar restricciones y viabilidad técnica • Diseñar la arquitectura de una solución web completa considerando patrones de diseño y separación de responsabilidades • Seleccionar tecnologías y frameworks adecuados para cada capa de la aplicación web basándose en requisitos y restricciones	**Unidad U1 Análisis y Diseño de Soluciones Web**
Función 2: Desarrollo de Interfaces de Usuario y Componentes Front-End • Desarrollar interfaces de usuario con HTML, CSS y frameworks de UI • Diseñar e implementar componentes reutilizables • Implementar lógica de negocio en el cliente • Gestionar estado global y local de la aplicación • Configurar enrutamiento y navegación entre vistas • Optimizar rendimiento de carga y accesibilidad web	**Bloque 2 - Desarrollo de Interfaces de Usuario y Componentes Front-End** • Desarrollar interfaces de usuario responsivas y accesibles que se adapten a múltiples dispositivos • Implementar componentes reutilizables y modularizados con documentación clara para facilitar mantenimiento • Programar lógica de negocio en cliente validando datos y optimizando la experiencia de usuario • Gestionar estado de aplicación compleja utilizando patrones de gestión de estado y optimizando re-renderizados • Implementar navegación entre vistas con rutas protegidas y dinámicas gestionando historial del navegador • Optimizar rendimiento de carga implementando lazy loading, code splitting y medición de métricas	**Unidad U2 Desarrollo de Interfaces de Usuario y Componentes Front-End**
Función 3: Desarrollo de Lógica de Negocio y APIs Back-End • Implementar autenticación y autorización • Programar lógica de negocio en el servidor • Diseñar e implementar APIs REST • Integrar servicios externos mediante APIs • Implementar validación de datos de entrada • Proteger datos sensibles y credenciales	**Bloque 3 - Desarrollo de Lógica de Negocio y APIs Back-End** • Programar lógica de negocio en servidor aplicando principios SOLID y garantizando integridad de datos • Construir APIs REST bien diseñadas con documentación clara y buenas prácticas de versionado • Integrar servicios externos mediante APIs gestionando autenticación y errores de forma robusta • Gestionar versionado de APIs comunicando cambios y manteniendo compatibilidad hacia atrás	**Unidad U3 Desarrollo de Lógica de Negocio y APIs Back-End**

Actividades	Bloques de competencias	Unidades
Función 4: Gestión de Datos y Bases de Datos • Conectar la aplicación con bases de datos • Diseñar y optimizar consultas a bases de datos • Desarrollar operaciones CRUD • Sincronizar datos entre cliente y servidor, y gestionar migraciones	**Bloque 4 - Gestión de Datos y Bases de Datos** • Validar datos de entrada en múltiples capas previniendo vulnerabilidades de inyección y XSS • Conectar aplicaciones con bases de datos utilizando ORMs y gestionar pools de conexiones eficientemente • Diseñar y optimizar consultas a bases de datos utilizando índices y análisis de planes de ejecución • Desarrollar operaciones CRUD manteniendo integridad referencial y transaccionalidad en bases de datos • Sincronizar datos entre front-end y back-end mediante WebSockets y mecanismos de caché consistente	**Unidad U4 Gestión de Datos y Bases de Datos**
Función 5: Testing, Depuración y Aseguramiento de Calidad • Escribir y ejecutar pruebas unitarias e integración • Ejecutar pruebas funcionales y end-to-end • Depurar errores en cliente y servidor • Revisar código y corregir incidencias de pruebas • Aplicar estándares de calidad de código	**Bloque 5 - Testing, Depuración y Aseguramiento de Calidad** • Escribir pruebas unitarias e integración con cobertura de código adecuada y frameworks de testing • Ejecutar pruebas funcionales end-to-end y reportar defectos con información detallada para resolución • Depurar errores en front-end y back-end utilizando herramientas de debugging y análisis de logs • Corregir incidencias detectadas en pruebas analizando causa raíz y actualizando pruebas automatizadas • Revisar código de otros desarrolladores proporcionando feedback técnico constructivo y verificando estándares	**Unidad U5 Testing, Depuración y Aseguramiento de Calidad**
Función 6: Seguridad, Rendimiento y Optimización • Aplicar medidas de seguridad web • Optimizar rendimiento del servidor • Optimizar código para escalabilidad • Optimizar rendimiento de carga en cliente	**Bloque 6 - Seguridad, Rendimiento y Optimización** • Gestionar autenticación y autorización implementando sistemas seguros de control de acceso • Aplicar medidas de seguridad web implementando cabeceras HTTP y auditorías de vulnerabilidades • Optimizar rendimiento del servidor implementando caché y monitorización de métricas de rendimiento • Actualizar dependencias y librerías resolviendo conflictos de versiones y validando compatibilidad	**Unidad U6 Seguridad, Rendimiento y Optimización**

Actividades	Bloques de competencias	Unidades
Función 7: Despliegue, Infraestructura y Operaciones • Configurar entornos de desarrollo y CI/CD • Automatizar compilaciones y despliegues • Desplegar aplicaciones en servidores y plataformas cloud • Containerizar aplicaciones con Docker • Monitorizar aplicaciones y analizar logs	**Bloque 7 - Despliegue, Infraestructura y Operaciones** • Configurar entornos de desarrollo con herramientas y documentación para facilitar onboarding • Automatizar compilaciones y despliegues utilizando herramientas de bundling y estrategias de despliegue • Desplegar aplicaciones en plataformas cloud configurando infraestructura y escalabilidad	**Unidad U7 Despliegue, Infraestructura y Operaciones**
Función 8: Mantenimiento, Evolución y Gestión de Incidencias • Mantener aplicaciones web en producción • Gestionar incidencias críticas de producción • Actualizar dependencias y gestionar versionado de APIs • Gestionar evolución y versionado de APIs	**Bloque 8 - Mantenimiento, Evolución y Gestión de Incidencias** • Implementar internacionalización y localización adaptando contenido a múltiples idiomas y regiones	**Unidad U8 Mantenimiento, Evolución y Gestión de Incidencias**
Función 9: Documentación, Comunicación y Colaboración • Documentar componentes, APIs y arquitectura • Colaborar con diseñadores y equipos multidisciplinarios • Comunicar con stakeholders y coordinar tareas de desarrollo	**Bloque 9 - Documentación, Comunicación y Colaboración**	**Unidad U9 Documentación, Comunicación y Colaboración**
Función 10: Planificación, Estimación y Gestión de Proyectos • Estimar esfuerzo y participar en planificación de sprints • Priorizar tareas y gestionar deuda técnica	**Bloque 10 - Planificación, Estimación y Gestión de Proyectos**	**Unidad U10 Planificación, Estimación y Gestión de Proyectos**
Función 11: Control de Versiones y Gestión de Código • Gestionar control de versiones y estrategias de branching	**Bloque 11 - Control de Versiones y Gestión de Código**	**Unidad U11 Control de Versiones y Gestión de Código**
Función 12: Funcionalidades Avanzadas y Experiencia de Usuario • Implementar comunicación en tiempo real • Implementar internacionalización y localización • Implementar Progressive Web Apps (PWA)	**Bloque 12 - Funcionalidades Avanzadas y Experiencia de Usuario** • Configurar integración continua automatizando pruebas y validaciones en cada commit	**Unidad U12 Funcionalidades Avanzadas y Experiencia de Usuario**

Tabla de síntesis de funciones y actividades

Función 1: Análisis y Diseño de Soluciones Web
Actividad 1.1 Recopilar y analizar requisitos funcionales con stakeholders
Actividad 1.2 Analizar requisitos técnicos y restricciones de infraestructura
Actividad 1.3 Diseñar la arquitectura global de la aplicación web
Actividad 1.4 Definir la estructura modular del front-end
Actividad 1.5 Definir la estructura modular del back-end
Actividad 1.6 Seleccionar tecnologías y frameworks adecuados
Función 2: Desarrollo de Interfaces de Usuario y Componentes Front-End
Actividad 2.1 Desarrollar interfaces de usuario con HTML, CSS y frameworks de UI
Actividad 2.2 Diseñar e implementar componentes reutilizables
Actividad 2.3 Implementar lógica de negocio en el cliente
Actividad 2.4 Gestionar estado global y local de la aplicación
Actividad 2.5 Configurar enrutamiento y navegación entre vistas
Actividad 2.6 Optimizar rendimiento de carga y accesibilidad web
Función 3: Desarrollo de Lógica de Negocio y APIs Back-End
Actividad 3.1 Implementar autenticación y autorización
Actividad 3.2 Programar lógica de negocio en el servidor
Actividad 3.3 Diseñar e implementar APIs REST
Actividad 3.4 Integrar servicios externos mediante APIs
Actividad 3.5 Implementar validación de datos de entrada
Actividad 3.6 Proteger datos sensibles y credenciales
Función 4: Gestión de Datos y Bases de Datos
Actividad 4.1 Conectar la aplicación con bases de datos
Actividad 4.2 Diseñar y optimizar consultas a bases de datos
Actividad 4.3 Desarrollar operaciones CRUD
Actividad 4.4 Sincronizar datos entre cliente y servidor, y gestionar migraciones
Función 5: Testing, Depuración y Aseguramiento de Calidad
Actividad 5.1 Escribir y ejecutar pruebas unitarias e integración
Actividad 5.2 Ejecutar pruebas funcionales y end-to-end
Actividad 5.3 Depurar errores en cliente y servidor
Actividad 5.4 Revisar código y corregir incidencias de pruebas
Actividad 5.5 Aplicar estándares de calidad de código
Función 6: Seguridad, Rendimiento y Optimización

Actividad 6.1 Aplicar medidas de seguridad web
Actividad 6.2 Optimizar rendimiento del servidor
Actividad 6.3 Optimizar código para escalabilidad
Actividad 6.4 Optimizar rendimiento de carga en cliente
Función 7: Despliegue, Infraestructura y Operaciones
Actividad 7.1 Configurar entornos de desarrollo y CI/CD
Actividad 7.2 Automatizar compilaciones y despliegues
Actividad 7.3 Desplegar aplicaciones en servidores y plataformas cloud
Actividad 7.4 Containerizar aplicaciones con Docker
Actividad 7.5 Monitorizar aplicaciones y analizar logs
Función 8: Mantenimiento, Evolución y Gestión de Incidencias
Actividad 8.1 Mantener aplicaciones web en producción
Actividad 8.2 Gestionar incidencias críticas de producción
Actividad 8.3 Actualizar dependencias y gestionar versionado de APIs
Actividad 8.4 Gestionar evolución y versionado de APIs
Función 9: Documentación, Comunicación y Colaboración
Actividad 9.1 Documentar componentes, APIs y arquitectura
Actividad 9.2 Colaborar con diseñadores y equipos multidisciplinarios
Actividad 9.3 Comunicar con stakeholders y coordinar tareas de desarrollo
Función 10: Planificación, Estimación y Gestión de Proyectos
Actividad 10.1 Estimar esfuerzo y participar en planificación de sprints
Actividad 10.2 Priorizar tareas y gestionar deuda técnica
Función 11: Control de Versiones y Gestión de Código
Actividad 11.1 Gestionar control de versiones y estrategias de branching
Función 12: Funcionalidades Avanzadas y Experiencia de Usuario
Actividad 12.1 Implementar comunicación en tiempo real
Actividad 12.2 Implementar internacionalización y localización
Actividad 12.3 Implementar Progressive Web Apps (PWA)

Estas áreas, desglosadas en actividades que el profesional realiza con autonomía o bajo supervisión, pueden no ejercerse todas en la entidad empleadora, especialmente en un primer empleo.

ANEXO II Marco de actividades profesionales

Contexto profesional y empleos

El Desarrollo Web Full Stack es una disciplina profesional que integra competencias de análisis, diseño, desarrollo, testing, seguridad y operaciones para crear aplicaciones web complejas y escalables. Los desarrolladores web full stack dominan todas las capas de una aplicación: interfaz de usuario (front-end), lógica de negocio (back-end), persistencia de datos (bases de datos) e infraestructura (cloud y operaciones). En el contexto profesional español, el desarrollo web full stack es una competencia altamente demandada en empresas de tecnología, agencias digitales, startups, consultorías y departamentos de TI de grandes organizaciones. Los profesionales en este rol participan en el ciclo completo de desarrollo: desde el análisis de requisitos con stakeholders, diseño de arquitecturas escalables, implementación de componentes front-end responsivos, desarrollo de APIs REST robustas, gestión de bases de datos optimizadas, ejecución de testing exhaustivo, aplicación de medidas de seguridad, optimización de rendimiento, despliegue en plataformas cloud y mantenimiento continuo. El mercado laboral valora especialmente la capacidad de tomar decisiones técnicas fundamentadas, comunicar con claridad a equipos multidisciplinarios, y adaptarse a tecnologías emergentes. La evolución profesional permite especialización profunda en front-end o back-end, transición hacia arquitectura de software, liderazgo técnico de equipos, o especialización en áreas como DevOps, seguridad o machine learning.

La persona titulada desempeña diferentes funciones:
– análisis y diseño de soluciones web ;
– desarrollo de interfaces de usuario y componentes front-end ;
– desarrollo de lógica de negocio y apis back-end ;
– gestión de datos y bases de datos ;
– testing, depuración y aseguramiento de calidad ;
– seguridad, rendimiento y optimización ;
– despliegue, infraestructura y operaciones ;
– mantenimiento, evolución y gestión de incidencias ;
– documentación, comunicación y colaboración ;
– planificación, estimación y gestión de proyectos ;
– control de versiones y gestión de código ;
– funcionalidades avanzadas y experiencia de usuario ;

Empleos relacionados

Los empleos de estos profesionales se encuentran en distintas estructuras públicas y privadas, entre ellas:
– Empresas de tecnología y software ;
– Agencias digitales y consultorías ;
– Startups y empresas emergentes ;
– Departamentos de TI de grandes organizaciones ;
– Empresas de servicios profesionales ;
– Organismos públicos y administración ;
– Trabajo autónomo y freelance ;

Los puestos reciben denominaciones diferentes según el sector. Algunos ejemplos son:
– Desarrollador Web Full Stack ;
– Ingeniero de Software Web ;
– Desarrollador Full Stack Senior ;

– Arquitecto de Soluciones Web ;
– Líder Técnico de Desarrollo Web ;
– Especialista en Desarrollo Front-End ;
– Especialista en Desarrollo Back-End ;
– Ingeniero DevOps ;
– Consultor Técnico Web ;

Evolución profesional (3 a 5 años)

Los desarrolladores web full stack pueden evolucionar profesionalmente en múltiples direcciones según sus intereses y fortalezas. La progresión típica incluye especialización profunda, liderazgo técnico o transición hacia roles estratégicos.

- Especialista Front-End: profundización en interfaces de usuario, frameworks modernos, accesibilidad y experiencia de usuario
- Especialista Back-End: profundización en arquitectura de servidores, bases de datos, APIs y sistemas distribuidos
- Arquitecto de Software: diseño de soluciones complejas, evaluación de tecnologías y toma de decisiones estratégicas
- Líder Técnico: liderazgo de equipos de desarrollo, mentoría, definición de estándares técnicos
- Ingeniero DevOps: especialización en infraestructura, automatización, despliegue y operaciones
- Especialista en Seguridad: profundización en seguridad web, auditorías, cumplimiento normativo
- Consultor Técnico: asesoramiento a clientes sobre soluciones web, evaluación de tecnologías
- Emprendedor: creación de startups y productos digitales propios

Continuación de estudios

Los profesionales que completen esta formación pueden continuar su desarrollo mediante formaciones especializadas en tecnologías específicas (frameworks avanzados, cloud computing, machine learning), certificaciones profesionales (AWS, Azure, Google Cloud), formaciones en liderazgo y gestión de proyectos, o estudios superiores en ingeniería informática, ciberseguridad o especialidades relacionadas. La formación continua es esencial en el desarrollo web debido a la rápida evolución de tecnologías y estándares.

ÉTICA Y DEONTOLOGÍA PROFESIONAL

El desarrollo web full stack requiere compromiso con principios éticos que garanticen la calidad, seguridad y responsabilidad social de las aplicaciones desarrolladas.

- Seguridad y protección de datos: implementar medidas robustas para proteger información sensible de usuarios y cumplir con regulaciones como RGPD
- Accesibilidad e inclusión: crear aplicaciones usables por personas con diferentes capacidades, cumpliendo estándares WCAG
- Transparencia y honestidad: comunicar limitaciones técnicas, riesgos y restricciones de forma clara a stakeholders
- Calidad y responsabilidad: desarrollar código de calidad, realizar testing exhaustivo y asumir responsabilidad por defectos
- Sostenibilidad: optimizar rendimiento y eficiencia energética de aplicaciones
- Respeto a la privacidad: implementar prácticas que respeten la privacidad de usuarios y cumplan con legislación aplicable
- Mejora continua: mantener actualización en tecnologías, estándares y buenas prácticas
- Colaboración y comunicación: trabajar efectivamente en equipos, compartir conocimiento y proporcionar feedback constructivo

Función 1: Análisis y Diseño de Soluciones Web

Función 1: Análisis y Diseño de Soluciones Web

Actividad 1.1 Recopilar y analizar requisitos funcionales con stakeholders

- Realizar sesiones de elicitación de requisitos con clientes y stakeholders para comprender las necesidades del negocio
  - ■ Conducir entrevistas estructuradas y no estructuradas con usuarios finales y responsables de negocio
  - ■ Documentar casos de uso y escenarios de usuario que reflejen las necesidades identificadas
  - ■ Validar la completitud y coherencia de los requisitos mediante sesiones de revisión con stakeholders
- Traducir las necesidades del negocio en especificaciones técnicas claras y medibles
  - ■ Crear historias de usuario con criterios de aceptación explícitos
  - ■ Definir funcionalidades, flujos de datos y comportamientos esperados de la aplicación
  - ■ Establecer prioridades y dependencias entre requisitos
- Validar que los requisitos sean completos, coherentes y realizables dentro de las restricciones del proyecto
  - ■ Identificar conflictos o ambigüedades en los requisitos y resolverlos con stakeholders
  - ■ Evaluar la viabilidad técnica de cada requisito en función de los recursos disponibles
  - ■ Documentar supuestos y restricciones que afecten a la implementación

■ Medios y recursos

- Herramientas de colaboración (Jira, Confluence, Miro)
- Plantillas de historias de usuario y casos de uso
- Documentación de requisitos funcionales
- Sesiones de trabajo con stakeholders y usuarios finales

■ Resultados esperados

- Especificación de requisitos funcionales documentada y validada
- Historias de usuario con criterios de aceptación claros
- Matriz de trazabilidad entre requisitos y funcionalidades

Función 1: Análisis y Diseño de Soluciones Web

Actividad 1.2 Analizar requisitos técnicos y restricciones de infraestructura

- Identificar y documentar los requisitos técnicos de la solución (rendimiento, seguridad, escalabilidad, integraciones)
  - ■ Definir objetivos de rendimiento (tiempos de respuesta, throughput, latencia)
  - ■ Especificar requisitos de seguridad (autenticación, autorización, cifrado, cumplimiento normativo)
  - ■ Establecer requisitos de escalabilidad y disponibilidad (uptime, capacidad de carga)
- Evaluar restricciones tecnológicas y de infraestructura que afecten al diseño de la solución
  - ■ Analizar la infraestructura existente y las limitaciones de recursos disponibles
  - ■ Identificar dependencias con sistemas legados o integraciones externas
  - ■ Evaluar restricciones de presupuesto, tiempo y equipo disponible
- Asegurar la viabilidad técnica de las funcionalidades solicitadas y documentar riesgos técnicos
  - ■ Realizar análisis de viabilidad para funcionalidades complejas o innovadoras
  - ■ Identificar riesgos técnicos y proponer estrategias de mitigación

- Documentar decisiones técnicas y justificaciones en un registro de decisiones arquitectónicas

■ Medios y recursos

- Documentación de infraestructura existente
- Herramientas de análisis de rendimiento y seguridad
- Registros de decisiones arquitectónicas (ADR)
- Evaluaciones de viabilidad técnica

■ Resultados esperados

- Especificación de requisitos técnicos documentada
- Análisis de restricciones y limitaciones técnicas
- Registro de riesgos técnicos y estrategias de mitigación

Función 1: Análisis y Diseño de Soluciones Web

Actividad 1.3 Diseñar la arquitectura global de la aplicación web

- Definir la arquitectura global de la aplicación (monolítica, microservicios, serverless, etc.)
  - Evaluar patrones arquitectónicos disponibles y seleccionar el más adecuado según requisitos
  - Definir los componentes principales de la arquitectura y sus responsabilidades
  - Establecer los límites entre servicios o módulos principales
- Establecer patrones de diseño y principios arquitectónicos a seguir por el equipo
  - Definir patrones de comunicación entre componentes (síncrona, asíncrona, eventos)
  - Establecer principios de separación de responsabilidades y desacoplamiento
  - Documentar convenciones de nomenclatura y organización de código a nivel arquitectónico
- Documentar la arquitectura de forma clara para guiar el desarrollo del equipo
  - Crear diagramas de arquitectura (C4, UML, etc.) que muestren componentes y relaciones
  - Redactar documentación de arquitectura explicando decisiones y trade-offs
  - Establecer un registro de decisiones arquitectónicas (ADR) para futuras referencias

■ Medios y recursos

- Herramientas de modelado arquitectónico (Lucidchart, Draw.io, ArchiMate)
- Patrones de arquitectura de software (TOGAF, C4 Model)
- Documentación de arquitectura empresarial
- Registros de decisiones arquitectónicas

■ Resultados esperados

- Diagrama de arquitectura global documentado
- Especificación de componentes principales y sus responsabilidades
- Documentación de patrones arquitectónicos y principios de diseño

Función 1: Análisis y Diseño de Soluciones Web

Actividad 1.4 Definir la estructura modular del front-end

- Planificar y estructurar los módulos, componentes y capas de la parte cliente de la aplicación
  - Identificar componentes reutilizables y módulos funcionales del front-end
  - Definir la jerarquía de componentes y sus dependencias
  - Establecer la separación entre componentes de presentación, contenedores y servicios
- Establecer convenciones de nomenclatura, organización de carpetas y flujo de datos en el front-end
  - Definir estructura de carpetas estándar para componentes, estilos, servicios y utilidades

- Establecer convenciones de nomenclatura para archivos, componentes y variables
- Documentar el flujo de datos (props, estado, eventos) entre componentes

• Garantizar la coherencia y mantenibilidad del código cliente mediante estándares claros
- Definir patrones de gestión de estado (Redux, Zustand, Context API, etc.)
- Establecer guías de estilo y convenciones de código para el front-end
- Crear plantillas de componentes para asegurar consistencia

■ Medios y recursos

• Herramientas de diseño de componentes (Figma, Storybook)
• Guías de estilo y patrones de componentes
• Documentación de arquitectura front-end
• Ejemplos de componentes reutilizables

■ Resultados esperados

• Estructura modular del front-end documentada
• Convenciones de nomenclatura y organización de carpetas definidas
• Guía de patrones de componentes y flujo de datos

Función 1: Análisis y Diseño de Soluciones Web

Actividad 1.5 Definir la estructura modular del back-end

• Planificar y estructurar los servicios, capas y módulos del servidor de la aplicación
- Identificar servicios de negocio y módulos funcionales del back-end
- Definir las capas de la aplicación (controladores, servicios, repositorios, modelos)
- Establecer las responsabilidades de cada capa y sus interfaces

• Establecer la separación de responsabilidades (controladores, servicios, repositorios, etc.)
- Definir patrones de capas (MVC, MVVM, Clean Architecture, etc.)
- Establecer convenciones para la organización de código en el back-end
- Documentar las responsabilidades de cada componente y sus dependencias

• Garantizar la coherencia y escalabilidad del código servidor mediante estándares claros
- Definir patrones de acceso a datos (DAO, Repository, ORM)
- Establecer guías de estilo y convenciones de código para el back-end
- Crear plantillas de servicios y controladores para asegurar consistencia

■ Medios y recursos

• Patrones de arquitectura de back-end (MVC, MVVM, Clean Architecture)
• Documentación de arquitectura de capas
• Guías de estilo y convenciones de código
• Ejemplos de servicios y controladores

■ Resultados esperados

• Estructura modular del back-end documentada
• Definición de capas y responsabilidades de cada componente
• Guía de patrones de servicios y acceso a datos

Función 1: Análisis y Diseño de Soluciones Web

Actividad 1.6 Seleccionar tecnologías y frameworks adecuados

• Evaluar y comparar tecnologías, frameworks y librerías disponibles para el proyecto

- Investigar opciones de tecnologías para front-end, back-end y bases de datos
- Comparar características, rendimiento, comunidad y madurez de cada opción
- Realizar pruebas de concepto (POC) para tecnologías críticas

• Justificar la elección tecnológica en función de los requisitos, el equipo y el contexto
- Documentar criterios de evaluación (rendimiento, seguridad, escalabilidad, costo, curva de aprendizaje)
- Crear matriz de comparación entre opciones tecnológicas
- Presentar recomendaciones con argumentos técnicos y de negocio

• Documentar las decisiones tecnológicas adoptadas para futuras referencias
- Registrar las decisiones en un registro de decisiones arquitectónicas (ADR)
- Documentar las versiones específicas de frameworks y librerías seleccionadas
- Crear guías de instalación y configuración de las tecnologías elegidas

■ Medios y recursos

• Matriz de comparación de tecnologías
• Documentación de frameworks y librerías
• Registros de decisiones arquitectónicas (ADR)
• Pruebas de concepto (POC) y benchmarks

■ Resultados esperados

• Stack tecnológico seleccionado y documentado
• Justificación de decisiones tecnológicas
• Guías de instalación y configuración de tecnologías

Función 2: Desarrollo de Interfaces de Usuario y Componentes Front-End

Función 2: Desarrollo de Interfaces de Usuario y Componentes Front-End

Actividad 2.1 Desarrollar interfaces de usuario con HTML, CSS y frameworks de UI

- Implementar las vistas y pantallas de la aplicación web a partir de diseños o wireframes
  - ■ Traducir diseños de UI/UX en código HTML semántico y accesible
  - ■ Implementar estilos CSS para lograr la apariencia visual deseada
  - ■ Utilizar frameworks de UI (Bootstrap, Tailwind, Material Design) para acelerar el desarrollo
- Asegurar la coherencia visual y la usabilidad de la interfaz en todos los contextos
  - ■ Aplicar guías de estilo y sistemas de diseño de la organización
  - ■ Validar la coherencia visual entre diferentes pantallas y estados de la interfaz
  - ■ Realizar pruebas de usabilidad para identificar problemas de navegación o claridad
- Optimizar el código HTML y CSS para rendimiento y mantenibilidad
  - ■ Minimizar y optimizar archivos CSS para reducir tamaño de descarga
  - ■ Utilizar técnicas de CSS modular (BEM, SMACSS) para facilitar el mantenimiento
  - ■ Implementar media queries y técnicas responsive para adaptación a dispositivos

■ Medios y recursos

- Herramientas de diseño (Figma, Adobe XD, Sketch)
- Frameworks de UI (Bootstrap, Tailwind CSS, Material Design)
- Editores de código (VS Code, WebStorm)
- Herramientas de validación HTML/CSS (W3C Validator, Lighthouse)

■ Resultados esperados

- Vistas HTML semánticas y accesibles implementadas
- Estilos CSS coherentes y optimizados
- Interfaz responsive que funciona en múltiples dispositivos

Función 2: Desarrollo de Interfaces de Usuario y Componentes Front-End

Actividad 2.2 Diseñar e implementar componentes reutilizables

- Diseñar componentes de interfaz modulares y reutilizables en el front-end
  - ■ Identificar patrones comunes en la interfaz que puedan ser extraídos como componentes
  - ■ Definir la API de cada componente (props, eventos, slots)
  - ■ Crear componentes independientes y sin dependencias externas innecesarias
- Documentar la API de cada componente para facilitar su uso por el equipo
  - ■ Crear documentación clara de props, eventos y comportamientos de cada componente
  - ■ Utilizar herramientas como Storybook para documentar componentes interactivamente
  - ■ Proporcionar ejemplos de uso y casos de uso comunes
- Garantizar que los componentes sean independientes, testeables y mantenibles
  - ■ Implementar componentes sin efectos secundarios y con lógica predecible
  - ■ Escribir pruebas unitarias para cada componente
  - ■ Mantener componentes pequeños y enfocados en una única responsabilidad

■ Medios y recursos

- Herramientas de documentación de componentes (Storybook, Bit)
- Frameworks de componentes (React, Vue, Angular)
- Herramientas de testing (Jest, Vitest, React Testing Library)
- Guías de diseño de componentes

■ Resultados esperados

- Biblioteca de componentes reutilizables implementada
- Documentación interactiva de componentes en Storybook
- Componentes con pruebas unitarias y cobertura de código

Función 2: Desarrollo de Interfaces de Usuario y Componentes Front-End

Actividad 2.3 Implementar lógica de negocio en el cliente

- Implementar la lógica de negocio que se ejecuta en el navegador del usuario
  - ■ Traducir reglas de negocio en código JavaScript/TypeScript ejecutable
  - ■ Implementar cálculos, transformaciones de datos y flujos de trabajo en el cliente
  - ■ Gestionar el estado de la lógica de negocio de forma predecible
- Gestionar validaciones, cálculos y flujos de trabajo en el lado cliente
  - ■ Implementar validaciones de datos antes de enviar al servidor
  - ■ Realizar cálculos y transformaciones de datos en el cliente cuando sea apropiado
  - ■ Implementar flujos de trabajo complejos (wizards, multi-step forms, etc.)
- Optimizar la experiencia de usuario reduciendo llamadas innecesarias al servidor
  - ■ Implementar caché local de datos para evitar llamadas redundantes
  - ■ Utilizar validación en cliente para proporcionar feedback inmediato
  - ■ Implementar operaciones optimistas para mejorar la percepción de rendimiento

■ Medios y recursos

- Frameworks de front-end (React, Vue, Angular)
- Librerías de validación (Yup, Zod, Joi)
- Herramientas de gestión de estado (Redux, Zustand, Pinia)
- Herramientas de debugging (DevTools, React DevTools)

■ Resultados esperados

- Lógica de negocio implementada en el cliente
- Validaciones de datos funcionando correctamente
- Flujos de trabajo complejos implementados y testeados

Función 2: Desarrollo de Interfaces de Usuario y Componentes Front-End

Actividad 2.4 Gestionar estado global y local de la aplicación

- Implementar soluciones de gestión de estado global y local en el front-end
  - ■ Seleccionar y configurar herramientas de gestión de estado (Redux, Zustand, Pinia, Context API)
  - ■ Definir la estructura del estado global de la aplicación
  - ■ Implementar acciones, reducers y selectores para manipular el estado
- Definir la estructura del estado y los flujos de actualización de forma clara
  - ■ Documentar la estructura del estado y sus cambios esperados
  - ■ Implementar flujos de actualización predecibles y rastreables
  - ■ Utilizar patrones como Redux Thunk o Sagas para manejar efectos secundarios
- Optimizar el rendimiento evitando re-renderizados innecesarios

- Utilizar selectores memoizados para evitar re-renderizados
- Implementar code splitting y lazy loading de reducers
- Monitorizar el rendimiento con herramientas de profiling

■ Medios y recursos

- Herramientas de gestión de estado (Redux, Zustand, Pinia, Context API)
- Herramientas de debugging (Redux DevTools, React DevTools)
- Librerías de memoización (Reselect, Immer)
- Herramientas de profiling (React Profiler, Chrome DevTools)

■ Resultados esperados

- Sistema de gestión de estado implementado y documentado
- Flujos de actualización de estado predecibles y rastreables
- Rendimiento optimizado sin re-renderizados innecesarios

Función 2: Desarrollo de Interfaces de Usuario y Componentes Front-End

Actividad 2.5 Configurar enrutamiento y navegación entre vistas

- Configurar y gestionar el enrutamiento de la aplicación web
  - Configurar rutas estáticas y dinámicas en el router (React Router, Vue Router, Angular Router)
  - Implementar navegación programática y basada en enlaces
  - Gestionar parámetros de URL y query strings
- Implementar rutas protegidas, rutas dinámicas y navegación programática
  - Crear guards de ruta para proteger vistas que requieren autenticación
  - Implementar rutas dinámicas basadas en datos (ej: /user/:id)
  - Gestionar redirecciones y navegación condicional
- Gestionar el historial del navegador y los parámetros de URL
  - Implementar navegación hacia atrás/adelante compatible con el historial del navegador
  - Preservar el estado de la aplicación al navegar
  - Implementar deep linking para permitir compartir URLs específicas

■ Medios y recursos

- Librerías de enrutamiento (React Router, Vue Router, Angular Router)
- Herramientas de debugging de rutas (React Router DevTools)
- Documentación de patrones de enrutamiento
- Herramientas de testing de navegación (Cypress, Playwright)

■ Resultados esperados

- Sistema de enrutamiento configurado y funcional
- Rutas protegidas implementadas con guards
- Navegación fluida entre vistas con gestión de historial

Función 2: Desarrollo de Interfaces de Usuario y Componentes Front-End

Actividad 2.6 Optimizar rendimiento de carga y accesibilidad web

- Analizar y mejorar los tiempos de carga de la aplicación web mediante técnicas de optimización
  - Implementar lazy loading de componentes y recursos
  - Realizar code splitting para reducir el tamaño del bundle inicial
  - Optimizar imágenes, fuentes y recursos estáticos

- Utilizar herramientas como Lighthouse o WebPageTest para medir el rendimiento
    - ■ Ejecutar auditorías de rendimiento regularmente
    - ■ Analizar métricas de Core Web Vitals (LCP, FID, CLS)
    - ■ Identificar cuellos de botella y oportunidades de mejora
- Aplicar las pautas de accesibilidad web (WCAG) para garantizar usabilidad por personas con discapacidades
    - ■ Implementar atributos ARIA para mejorar la accesibilidad
    - ■ Asegurar navegación por teclado en toda la interfaz
    - ■ Validar contraste de colores y legibilidad de texto

■ Medios y recursos

- Herramientas de medición de rendimiento (Lighthouse, WebPageTest, GTmetrix)
- Herramientas de optimización (Webpack, Vite, ImageOptim)
- Herramientas de validación de accesibilidad (axe DevTools, WAVE)
- Guías de accesibilidad web (WCAG 2.1)

■ Resultados esperados

- Tiempos de carga reducidos y optimizados
- Puntuación de Lighthouse mejorada (>90)
- Interfaz accesible según estándares WCAG 2.1 AA

Función 3: Desarrollo de Lógica de Negocio y APIs Back-End

Función 3: Desarrollo de Lógica de Negocio y APIs Back-End

Actividad 3.1 Implementar autenticación y autorización

- Implementar mecanismos de autenticación de usuarios (JWT, sesiones, OAuth2, etc.)
  - ■ Configurar estrategias de autenticación (local, OAuth2, SAML, etc.)
  - ■ Implementar generación y validación de tokens (JWT, sesiones)
  - ■ Gestionar el ciclo de vida de sesiones y tokens (creación, renovación, revocación)
- Definir y aplicar políticas de autorización basadas en roles y permisos
  - ■ Implementar control de acceso basado en roles (RBAC)
  - ■ Definir permisos granulares para diferentes recursos y operaciones
  - ■ Implementar middleware de autorización en el servidor
- Asegurar la protección de rutas y recursos sensibles de la aplicación
  - ■ Implementar guards de ruta que verifiquen autenticación y autorización
  - ■ Proteger endpoints de API con validación de tokens
  - ■ Implementar manejo de errores de autenticación (401, 403)

■ Medios y recursos

- Librerías de autenticación (Passport.js, Auth0, Firebase Auth)
- Herramientas de generación de tokens (jsonwebtoken, bcrypt)
- Middleware de autenticación y autorización
- Documentación de estándares de seguridad (OAuth2, OpenID Connect)

■ Resultados esperados

- Sistema de autenticación implementado y funcional
- Políticas de autorización basadas en roles definidas
- Rutas y recursos protegidos correctamente

Función 3: Desarrollo de Lógica de Negocio y APIs Back-End

Actividad 3.2 Programar lógica de negocio en el servidor

- Implementar la lógica de negocio en el servidor, incluyendo reglas de negocio, cálculos y procesos
  - ■ Traducir requisitos de negocio en código servidor ejecutable
  - ■ Implementar reglas de negocio complejas y flujos de trabajo
  - ■ Gestionar transacciones y consistencia de datos
- Estructurar el código servidor siguiendo principios SOLID y patrones de diseño
  - ■ Aplicar principios SOLID (Single Responsibility, Open/Closed, Liskov, Interface Segregation, Dependency Inversion)
  - ■ Utilizar patrones de diseño (Factory, Strategy, Observer, etc.)
  - ■ Implementar inyección de dependencias para facilitar testing
- Garantizar la seguridad y la integridad de los datos procesados en el servidor
  - ■ Validar y sanitizar datos de entrada
  - ■ Implementar controles de acceso a nivel de lógica de negocio
  - ■ Registrar y auditar operaciones críticas

■ Medios y recursos

- Frameworks de back-end (Express, Django, Spring, Laravel)

- Patrones de arquitectura (MVC, MVVM, Clean Architecture)
- Herramientas de testing (Jest, Mocha, Pytest)
- Herramientas de debugging (Node Inspector, Python Debugger)

■ **Resultados esperados**

- Lógica de negocio implementada según requisitos
- Código servidor estructurado siguiendo principios SOLID
- Integridad de datos garantizada en operaciones críticas

Función 3: Desarrollo de Lógica de Negocio y APIs Back-End

Actividad 3.3 Diseñar e implementar APIs REST

- Diseñar e implementar APIs RESTful siguiendo las convenciones y buenas prácticas
  - ■ Utilizar verbos HTTP correctamente (GET, POST, PUT, DELETE, PATCH)
  - ■ Implementar códigos de estado HTTP apropiados (200, 201, 400, 401, 403, 404, 500)
  - ■ Diseñar URLs semánticas y coherentes para recursos
- Definir los contratos de la API y documentarlos con herramientas como Swagger/OpenAPI
  - ■ Crear especificaciones OpenAPI/Swagger para cada endpoint
  - ■ Documentar parámetros, respuestas y códigos de error
  - ■ Proporcionar ejemplos de solicitudes y respuestas
- Garantizar la seguridad, el rendimiento y la mantenibilidad de los endpoints
  - ■ Implementar autenticación y autorización en endpoints
  - ■ Implementar rate limiting y throttling para proteger contra abuso
  - ■ Optimizar consultas y respuestas para rendimiento

■ **Medios y recursos**

- Frameworks de API REST (Express, Django REST, Spring Boot)
- Herramientas de documentación (Swagger UI, ReDoc, Postman)
- Herramientas de testing de API (Postman, Insomnia, REST Client)
- Librerías de validación (Joi, Yup, Pydantic)

■ **Resultados esperados**

- APIs REST implementadas siguiendo estándares RESTful
- Documentación OpenAPI/Swagger completa y actualizada
- Endpoints seguros, performantes y bien documentados

Función 3: Desarrollo de Lógica de Negocio y APIs Back-End

Actividad 3.4 Integrar servicios externos mediante APIs

- Conectar la aplicación con servicios de terceros a través de sus APIs
  - ■ Investigar y evaluar APIs de servicios externos (pasarelas de pago, email, mapas, redes sociales)
  - ■ Implementar clientes HTTP para consumir APIs externas
  - ■ Gestionar endpoints, parámetros y respuestas de servicios externos
- Gestionar la autenticación con servicios externos (OAuth, API keys, etc.)
  - ■ Implementar flujos de autenticación OAuth2 con servicios externos
  - ■ Gestionar API keys y credenciales de forma segura
  - ■ Renovar tokens y manejar expiración de credenciales
- Manejar errores y reintentos en las integraciones externas
  - ■ Implementar manejo robusto de errores HTTP y timeouts

- Implementar reintentos con backoff exponencial
- Registrar y monitorizar fallos de integración

■ Medios y recursos

- Librerías HTTP (axios, requests, fetch)
- Herramientas de testing de integraciones (Postman, Insomnia)
- Documentación de APIs externas
- Herramientas de monitorización (Sentry, New Relic)

■ Resultados esperados

- Integraciones con servicios externos implementadas
- Autenticación con servicios externos funcionando correctamente
- Manejo robusto de errores y reintentos en integraciones

Función 3: Desarrollo de Lógica de Negocio y APIs Back-End

Actividad 3.5 Implementar validación de datos de entrada

- Validar los datos recibidos tanto en el cliente como en el servidor para garantizar su integridad y seguridad
  - Implementar validación de tipos de datos (string, number, boolean, etc.)
  - Validar formato de datos (email, URL, fecha, etc.)
  - Validar rangos y límites de datos (longitud, valor mínimo/máximo)
- Implementar mensajes de error claros y útiles para el usuario
  - Proporcionar mensajes de error específicos y accionables
  - Localizar mensajes de error según idioma del usuario
  - Mostrar errores en contexto (junto al campo que causó el error)
- Prevenir vulnerabilidades como inyección SQL, XSS y otros ataques basados en datos maliciosos
  - Sanitizar datos de entrada para prevenir inyección SQL
  - Escapar datos en salida para prevenir XSS
  - Validar y rechazar datos sospechosos o malformados

■ Medios y recursos

- Librerías de validación (Yup, Zod, Joi, Pydantic)
- Herramientas de sanitización (DOMPurify, sanitize-html)
- Herramientas de testing de seguridad (OWASP ZAP, Burp Suite)
- Documentación de vulnerabilidades web (OWASP Top 10)

■ Resultados esperados

- Validación de datos implementada en cliente y servidor
- Mensajes de error claros y útiles para usuarios
- Protección contra vulnerabilidades comunes de inyección

Función 3: Desarrollo de Lógica de Negocio y APIs Back-End

Actividad 3.6 Proteger datos sensibles y credenciales

- Implementar mecanismos de cifrado y almacenamiento seguro de datos sensibles
  - Cifrar datos sensibles en reposo (contraseñas, tokens, información personal)
  - Utilizar algoritmos de hash seguros (bcrypt, Argon2) para contraseñas
  - Implementar cifrado en tránsito (HTTPS, TLS)

- Gestionar variables de entorno y secretos de forma segura
  - ■ Utilizar herramientas de gestión de secretos (Vault, AWS Secrets Manager, HashiCorp Vault)
  - ■ Nunca almacenar secretos en código fuente o archivos de configuración
  - ■ Rotar secretos regularmente y auditar acceso
- Asegurar el cumplimiento de normativas de protección de datos (RGPD, etc.)
  - ■ Implementar derecho al olvido (eliminación de datos personales)
  - ■ Obtener consentimiento explícito para recopilación de datos
  - ■ Documentar políticas de privacidad y tratamiento de datos

■ Medios y recursos

- Herramientas de cifrado (OpenSSL, libsodium)
- Herramientas de gestión de secretos (Vault, AWS Secrets Manager)
- Librerías de hash (bcrypt, Argon2)
- Documentación de normativas (RGPD, CCPA)

■ Resultados esperados

- Datos sensibles cifrados y protegidos
- Secretos gestionados de forma segura
- Cumplimiento de normativas de protección de datos

Función 4: Gestión de Datos y Bases de Datos

Función 4: Gestión de Datos y Bases de Datos

Actividad 4.1 Conectar la aplicación con bases de datos

- Configurar y gestionar la conexión entre la aplicación y los sistemas de bases de datos
  - ■ Configurar drivers o ORMs para conectar con bases de datos (MySQL, PostgreSQL, MongoDB, etc.)
  - ■ Gestionar credenciales de conexión de forma segura
  - ■ Implementar manejo de errores de conexión y reconexión automática
- Utilizar ORMs o drivers nativos para interactuar con la base de datos
  - ■ Seleccionar y configurar ORM apropiado (Sequelize, TypeORM, SQLAlchemy, Mongoose)
  - ■ Implementar modelos de datos que representen entidades del negocio
  - ■ Utilizar métodos del ORM para operaciones CRUD
- Gestionar pools de conexiones y asegurar la eficiencia de las operaciones de acceso a datos
  - ■ Configurar tamaño de pool de conexiones según carga esperada
  - ■ Monitorizar uso de conexiones y detectar fugas
  - ■ Implementar timeouts y límites de conexión

■ Medios y recursos

- ORMs y drivers de base de datos (Sequelize, TypeORM, SQLAlchemy, Mongoose)
- Herramientas de gestión de bases de datos (pgAdmin, MongoDB Compass)
- Herramientas de monitorización (New Relic, DataDog)
- Documentación de bases de datos

■ Resultados esperados

- Conexión a base de datos configurada y funcional
- Modelos de datos implementados con ORM
- Pool de conexiones optimizado y monitorizado

Función 4: Gestión de Datos y Bases de Datos

Actividad 4.2 Diseñar y optimizar consultas a bases de datos

- Diseñar consultas SQL o NoSQL eficientes para recuperar y manipular datos
  - ■ Escribir consultas SQL optimizadas con JOINs y subconsultas apropiadas
  - ■ Utilizar agregaciones y funciones de base de datos cuando sea apropiado
  - ■ Diseñar consultas NoSQL con proyecciones y filtros eficientes
- Analizar y optimizar el rendimiento de las consultas mediante índices y otras técnicas
  - ■ Utilizar EXPLAIN PLAN para analizar ejecución de consultas
  - ■ Crear índices en columnas frecuentemente consultadas
  - ■ Identificar y eliminar consultas lentas mediante profiling
- Evitar problemas de rendimiento como N+1 queries o full table scans
  - ■ Implementar eager loading para evitar N+1 queries
  - ■ Utilizar batch queries cuando sea apropiado
  - ■ Evitar full table scans mediante índices adecuados

■ Medios y recursos

- Herramientas de análisis de consultas (EXPLAIN PLAN, Query Profiler)

- Herramientas de monitorización de base de datos (pgAdmin, MongoDB Compass)
- Documentación de optimización de bases de datos
- Herramientas de testing de rendimiento (Apache JMeter, Locust)

■ Resultados esperados

- Consultas SQL/NoSQL optimizadas y eficientes
- Índices creados en columnas críticas
- Problemas de rendimiento identificados y resueltos

Función 4: Gestión de Datos y Bases de Datos

Actividad 4.3 Desarrollar operaciones CRUD

- Implementar las operaciones básicas de creación, lectura, actualización y eliminación de datos
  - ■ Implementar operación CREATE para insertar nuevos registros
  - ■ Implementar operación READ para recuperar registros existentes
  - ■ Implementar operaciones UPDATE y DELETE para modificar y eliminar registros
- Asegurar la integridad referencial y las transacciones en las operaciones de escritura
  - ■ Validar restricciones de integridad referencial (foreign keys)
  - ■ Implementar transacciones para operaciones multi-paso
  - ■ Manejar conflictos de concurrencia (optimistic/pessimistic locking)
- Exponer estas operaciones a través de la API o la interfaz de usuario según corresponda
  - ■ Crear endpoints de API para operaciones CRUD
  - ■ Implementar formularios en interfaz de usuario para operaciones CRUD
  - ■ Validar permisos de usuario antes de permitir operaciones

■ Medios y recursos

- ORMs y drivers de base de datos
- Frameworks de API REST
- Herramientas de testing (Jest, Mocha, Pytest)
- Herramientas de validación de datos

■ Resultados esperados

- Operaciones CRUD implementadas y funcionales
- Integridad referencial garantizada
- Endpoints de API y formularios de UI para CRUD

Función 4: Gestión de Datos y Bases de Datos

Actividad 4.4 Sincronizar datos entre cliente y servidor, y gestionar migraciones

- Gestionar el flujo de datos entre el cliente y el servidor, asegurando la consistencia
  - ■ Implementar mecanismos de sincronización de datos (polling, WebSockets, Server-Sent Events)
  - ■ Gestionar caché en cliente para reducir llamadas al servidor
  - ■ Manejar conflictos de datos cuando cliente y servidor divergen
- Implementar mecanismos de caché, polling o WebSockets según los requisitos
  - ■ Implementar caché local en cliente (localStorage, IndexedDB)
  - ■ Utilizar polling para actualizaciones periódicas
  - ■ Utilizar WebSockets para comunicación en tiempo real
- Planificar y ejecutar migraciones de esquema de base de datos de forma controlada
  - ■ Utilizar herramientas de migración (Flyway, Liquibase, Alembic)

■ Crear migraciones reversibles para facilitar rollback
■ Validar integridad de datos antes y después de migraciones

■ Medios y recursos

- Librerías de sincronización (Socket.io, SockJS)
- Herramientas de caché (Redis, Memcached)
- Herramientas de migración (Flyway, Liquibase, Alembic)
- Herramientas de testing de sincronización

■ Resultados esperados

- Sincronización de datos cliente-servidor funcionando
- Caché implementado y optimizado
- Migraciones de base de datos ejecutadas exitosamente

Función 5: Testing, Depuración y Aseguramiento de Calidad

Función 5: Testing, Depuración y Aseguramiento de Calidad

Actividad 5.1 Escribir y ejecutar pruebas unitarias e integración

- Desarrollar pruebas unitarias para funciones, métodos y componentes individuales
  - ■ Escribir pruebas para funciones de lógica de negocio
  - ■ Escribir pruebas para componentes de UI
  - ■ Utilizar frameworks de testing (Jest, Vitest, Mocha, Pytest)
- Desarrollar pruebas que verifiquen la correcta interacción entre módulos y servicios
  - ■ Escribir pruebas de integración para endpoints de API
  - ■ Escribir pruebas de integración para flujos de datos cliente-servidor
  - ■ Utilizar herramientas como Supertest, Cypress o Playwright
- Mantener una cobertura de código adecuada y asegurar que las pruebas sean rápidas y deterministas
  - ■ Medir cobertura de código con herramientas (Istanbul, Coverage.py)
  - ■ Mantener cobertura mínima del 80% en código crítico
  - ■ Optimizar pruebas para ejecución rápida

■ Medios y recursos

- Frameworks de testing (Jest, Vitest, Mocha, Pytest)
- Herramientas de cobertura de código (Istanbul, Coverage.py)
- Herramientas de mocking (Sinon, Jest Mocks, unittest.mock)
- Herramientas de testing de API (Supertest, REST Assured)

■ Resultados esperados

- Suite de pruebas unitarias implementada
- Pruebas de integración para flujos críticos
- Cobertura de código >80% en módulos críticos

Función 5: Testing, Depuración y Aseguramiento de Calidad

Actividad 5.2 Ejecutar pruebas funcionales y end-to-end

- Planificar y ejecutar pruebas funcionales que verifiquen que la aplicación cumple con los requisitos
  - ■ Crear casos de prueba basados en requisitos funcionales
  - ■ Ejecutar pruebas manuales en diferentes navegadores y dispositivos
  - ■ Documentar resultados de pruebas y defectos encontrados
- Realizar pruebas end-to-end simulando el comportamiento real del usuario
  - ■ Utilizar herramientas de automatización (Cypress, Playwright, Selenium)
  - ■ Crear scripts de prueba que simulen flujos de usuario completos
  - ■ Ejecutar pruebas en diferentes navegadores y resoluciones
- Reportar y gestionar los defectos encontrados durante las pruebas
  - ■ Documentar defectos con pasos de reproducción claros
  - ■ Clasificar defectos por severidad y prioridad
  - ■ Hacer seguimiento de defectos hasta su resolución

■ Medios y recursos

- Herramientas de automatización de pruebas (Cypress, Playwright, Selenium)
- Herramientas de gestión de defectos (Jira, Azure DevOps)

- Herramientas de reporte de pruebas (Allure, TestRail)
- Documentación de casos de prueba

■ Resultados esperados

- Casos de prueba funcionales documentados y ejecutados
- Pruebas end-to-end automatizadas para flujos críticos
- Defectos reportados y gestionados correctamente

Función 5: Testing, Depuración y Aseguramiento de Calidad

Actividad 5.3 Depurar errores en cliente y servidor

- Identificar, reproducir y corregir errores en el código cliente utilizando herramientas de desarrollo
  - ■ Utilizar DevTools del navegador para depuración de JavaScript
  - ■ Analizar errores de renderizado y problemas de DOM
  - ■ Depurar llamadas a API desde el cliente
- Identificar, reproducir y corregir errores en el código servidor
  - ■ Utilizar herramientas de depuración de servidor (Node Inspector, Python Debugger)
  - ■ Analizar trazas de error y excepciones
  - ■ Depurar problemas de lógica de negocio
- Documentar los errores encontrados y las soluciones aplicadas
  - ■ Crear reportes de error con contexto y pasos de reproducción
  - ■ Documentar soluciones para evitar problemas similares en el futuro
  - ■ Mantener un registro de errores comunes y sus soluciones

■ Medios y recursos

- Herramientas de debugging (Chrome DevTools, Firefox DevTools, Node Inspector)
- Herramientas de análisis de logs (ELK Stack, Splunk)
- Herramientas de monitorización (Sentry, New Relic)
- Documentación de debugging

■ Resultados esperados

- Errores en cliente identificados y corregidos
- Errores en servidor identificados y corregidos
- Documentación de errores y soluciones

Función 5: Testing, Depuración y Aseguramiento de Calidad

Actividad 5.4 Revisar código y corregir incidencias de pruebas

- Realizar revisiones de código (code reviews) para garantizar la calidad y cumplimiento de estándares
  - ■ Revisar código de otros desarrolladores antes de merge
  - ■ Verificar cumplimiento de estándares de calidad y convenciones
  - ■ Proporcionar feedback constructivo y sugerencias de mejora
- Analizar los defectos reportados durante las fases de prueba e identificar su causa raíz
  - ■ Reproducir defectos en entorno de desarrollo
  - ■ Analizar logs y trazas para identificar causa raíz
  - ■ Documentar análisis de causa raíz
- Implementar las correcciones necesarias y verificar que no introducen regresiones
  - ■ Implementar fix para el defecto identificado
  - ■ Ejecutar pruebas de regresión para verificar que no se rompió nada

■ Actualizar pruebas automatizadas para cubrir el caso que falló

■ Medios y recursos

- Herramientas de control de versiones (Git, GitHub, GitLab)
- Herramientas de code review (GitHub Pull Requests, GitLab Merge Requests)
- Herramientas de análisis de código (SonarQube, ESLint)
- Herramientas de testing (Jest, Mocha, Pytest)

■ Resultados esperados

- Code reviews realizados y feedback proporcionado
- Defectos analizados y causa raíz identificada
- Correcciones implementadas sin regresiones

Función 5: Testing, Depuración y Aseguramiento de Calidad

Actividad 5.5 Aplicar estándares de calidad de código

- Definir y aplicar estándares de calidad de código en el equipo
  - ■ Configurar linters (ESLint, Pylint, Checkstyle) con reglas estándar
  - ■ Configurar formateadores de código (Prettier, Black)
  - ■ Establecer convenciones de nomenclatura y estructura de código
- Implementar procesos de revisión y validación para garantizar calidad
  - ■ Configurar pre-commit hooks para validación automática
  - ■ Implementar CI/CD checks para validar calidad antes de merge
  - ■ Realizar auditorías periódicas de calidad de código
- Promover una cultura de calidad continua en el equipo
  - ■ Compartir mejores prácticas y patrones de código
  - ■ Realizar sesiones de refactorización periódicas
  - ■ Documentar lecciones aprendidas y anti-patrones a evitar

■ Medios y recursos

- Herramientas de linting (ESLint, Pylint, Checkstyle)
- Herramientas de formateo (Prettier, Black)
- Herramientas de análisis estático (SonarQube, CodeClimate)
- Documentación de estándares de código

■ Resultados esperados

- Estándares de calidad de código definidos y documentados
- Validación automática de calidad en CI/CD
- Cobertura de linting y formateo en todo el código

Función 6: Seguridad, Rendimiento y Optimización

Función 6: Seguridad, Rendimiento y Optimización

Actividad 6.1 Aplicar medidas de seguridad web

- Implementar las mejores prácticas de seguridad web (OWASP Top 10)
  - ■ Proteger contra inyección SQL mediante prepared statements
  - ■ Proteger contra XSS mediante escapado de datos
  - ■ Proteger contra CSRF mediante tokens CSRF
  - ■ Proteger contra autenticación débil mediante contraseñas fuertes
- Configurar cabeceras de seguridad HTTP, políticas CORS y protección CSRF
  - ■ Configurar Content-Security-Policy (CSP)
  - ■ Configurar X-Frame-Options para prevenir clickjacking
  - ■ Configurar CORS restrictivo para controlar acceso cross-origin
  - ■ Implementar protección CSRF mediante tokens
- Realizar auditorías de seguridad periódicas y aplicar las correcciones necesarias
  - ■ Utilizar herramientas de escaneo de seguridad (OWASP ZAP, Burp Suite)
  - ■ Realizar pruebas de penetración regularmente
  - ■ Mantener un registro de vulnerabilidades encontradas y corregidas

■ Medios y recursos

- Documentación de OWASP Top 10
- Herramientas de escaneo de seguridad (OWASP ZAP, Burp Suite)
- Librerías de seguridad (helmet.js, django-cors-headers)
- Herramientas de análisis de dependencias (Snyk, Dependabot)

■ Resultados esperados

- Medidas de seguridad OWASP implementadas
- Cabeceras de seguridad HTTP configuradas
- Auditorías de seguridad realizadas y vulnerabilidades corregidas

Función 6: Seguridad, Rendimiento y Optimización

Actividad 6.2 Optimizar rendimiento del servidor

- Identificar y eliminar cuellos de botella en el rendimiento del servidor
  - ■ Analizar consultas lentas a base de datos
  - ■ Identificar procesos bloqueantes y operaciones síncronas innecesarias
  - ■ Utilizar herramientas de profiling para identificar hotspots
- Implementar técnicas de caché en servidor (Redis, Memcached, etc.)
  - ■ Implementar caché de consultas frecuentes
  - ■ Implementar caché de sesiones de usuario
  - ■ Implementar caché de respuestas HTTP
- Monitorizar los tiempos de respuesta y establecer alertas ante degradaciones
  - ■ Configurar monitorización de tiempos de respuesta
  - ■ Establecer alertas para degradaciones de rendimiento
  - ■ Analizar métricas de rendimiento regularmente

■ Medios y recursos

- Herramientas de caché (Redis, Memcached)
- Herramientas de profiling (Node Profiler, Python Profiler)
- Herramientas de monitorización (New Relic, DataDog, Prometheus)
- Herramientas de testing de carga (Apache JMeter, Locust)

■ **Resultados esperados**

- Cuellos de botella identificados y eliminados
- Caché implementado y optimizado
- Monitorización de rendimiento configurada con alertas

Función 6: Seguridad, Rendimiento y Optimización

Actividad 6.3 Optimizar código para escalabilidad

- Refactorizar y mejorar el código para que la aplicación pueda escalar horizontalmente y verticalmente
  - ■ Eliminar estado compartido para permitir escalado horizontal
  - ■ Implementar desacoplamiento entre componentes
  - ■ Utilizar colas de mensajes para procesamiento asíncrono
- Aplicar principios de diseño escalable (stateless, desacoplamiento, colas de mensajes, etc.)
  - ■ Diseñar servicios stateless para facilitar escalado
  - ■ Implementar patrones de comunicación asíncrona
  - ■ Utilizar bases de datos distribuidas cuando sea apropiado
- Realizar pruebas de carga para identificar límites y cuellos de botella
  - ■ Crear escenarios de prueba de carga realistas
  - ■ Ejecutar pruebas de carga con herramientas (JMeter, Locust)
  - ■ Analizar resultados y identificar límites de escalabilidad

■ **Medios y recursos**

- Herramientas de testing de carga (Apache JMeter, Locust, Gatling)
- Herramientas de monitorización (New Relic, DataDog)
- Patrones de arquitectura escalable (microservicios, serverless)
- Colas de mensajes (RabbitMQ, Kafka, AWS SQS)

■ **Resultados esperados**

- Código refactorizado para escalabilidad
- Principios de diseño escalable aplicados
- Pruebas de carga ejecutadas y límites identificados

Función 6: Seguridad, Rendimiento y Optimización

Actividad 6.4 Optimizar rendimiento de carga en cliente

- Analizar y mejorar los tiempos de carga mediante técnicas de optimización
  - ■ Implementar lazy loading de componentes y imágenes
  - ■ Realizar code splitting para reducir tamaño del bundle inicial
  - ■ Optimizar imágenes, fuentes y recursos estáticos
- Utilizar herramientas como Lighthouse o WebPageTest para medir el rendimiento
  - ■ Ejecutar auditorías de rendimiento regularmente
  - ■ Analizar métricas de Core Web Vitals
  - ■ Identificar oportunidades de mejora
- Implementar caché del navegador y optimizaciones de red

- ■ Configurar headers de caché HTTP
- ■ Implementar service workers para caché offline
- ■ Utilizar CDN para distribución de contenido estático

■ **Medios y recursos**

- • Herramientas de medición (Lighthouse, WebPageTest, GTmetrix)
- • Herramientas de optimización (Webpack, Vite, ImageOptim)
- • Herramientas de análisis (Chrome DevTools, Lighthouse CI)
- • CDN y servicios de distribución

■ **Resultados esperados**

- • Tiempos de carga reducidos
- • Puntuación de Lighthouse mejorada
- • Caché del navegador configurado correctamente

Función 7: Despliegue, Infraestructura y Operaciones

Función 7: Despliegue, Infraestructura y Operaciones

Actividad 7.1 Configurar entornos de desarrollo y CI/CD

- Preparar y configurar los entornos locales de desarrollo para el equipo
  - ■ Documentar requisitos de software y versiones
  - ■ Crear scripts de instalación automatizados
  - ■ Configurar variables de entorno y archivos de configuración
- Configurar pipelines de CI/CD utilizando herramientas como GitHub Actions, GitLab CI, Jenkins
  - ■ Crear workflows de CI/CD para automatizar pruebas y validaciones
  - ■ Configurar triggers para ejecutar pipeline en cada commit/PR
  - ■ Implementar notificaciones de fallos de pipeline
- Automatizar la ejecución de pruebas, análisis de código y validaciones
  - ■ Ejecutar pruebas unitarias e integración en pipeline
  - ■ Ejecutar análisis estático de código (linting, SAST)
  - ■ Validar cobertura de código y seguridad

■ Medios y recursos

- Herramientas de CI/CD (GitHub Actions, GitLab CI, Jenkins)
- Herramientas de análisis de código (SonarQube, ESLint)
- Herramientas de testing (Jest, Mocha, Pytest)
- Documentación de configuración de entornos

■ Resultados esperados

- Entorno de desarrollo configurado y documentado
- Pipeline de CI/CD automatizado
- Validaciones automáticas ejecutándose en cada commit

Función 7: Despliegue, Infraestructura y Operaciones

Actividad 7.2 Automatizar compilaciones y despliegues

- Crear y mantener scripts y pipelines para automatizar el proceso de build y despliegue
  - ■ Crear scripts de build que compilen y empaqueten la aplicación
  - ■ Automatizar generación de artefactos (JAR, WAR, Docker images)
  - ■ Implementar versionado automático de artefactos
- Configurar herramientas de bundling (Webpack, Vite, etc.) y optimización para producción
  - ■ Configurar bundlers para optimizar código y assets
  - ■ Implementar minificación y compresión de código
  - ■ Configurar source maps para debugging en producción
- Implementar estrategias de despliegue como blue-green, canary o rolling updates
  - ■ Implementar despliegue blue-green para cero downtime
  - ■ Implementar despliegue canary para validar cambios
  - ■ Implementar rolling updates para actualización gradual

■ Medios y recursos

- Herramientas de bundling (Webpack, Vite, Parcel)
- Herramientas de build (Maven, Gradle, npm)

- Herramientas de CI/CD (GitHub Actions, GitLab CI, Jenkins)
- Herramientas de despliegue (Kubernetes, Docker, Terraform)

■ Resultados esperados

- Scripts de build automatizados
- Bundling y optimización configurados
- Estrategias de despliegue implementadas

Función 7: Despliegue, Infraestructura y Operaciones

Actividad 7.3 Desplegar aplicaciones en servidores y plataformas cloud

- Publicar y gestionar el despliegue de aplicaciones web en servidores propios o plataformas cloud
  - ■ Desplegar aplicaciones en servidores propios (VPS, servidores dedicados)
  - ■ Desplegar en plataformas cloud (AWS, Azure, GCP, Vercel, Heroku)
  - ■ Gestionar múltiples entornos (desarrollo, staging, producción)
- Configurar los recursos de infraestructura necesarios
  - ■ Provisionar servidores, bases de datos, almacenamiento
  - ■ Configurar redes, firewalls y grupos de seguridad
  - ■ Configurar balanceadores de carga y auto-scaling
- Gestionar certificados SSL, dominios y configuraciones de infraestructura
  - ■ Obtener y renovar certificados SSL/TLS
  - ■ Configurar DNS y dominios
  - ■ Configurar HTTPS y redirecciones

■ Medios y recursos

- Plataformas cloud (AWS, Azure, GCP, Vercel, Heroku)
- Herramientas de infraestructura como código (Terraform, CloudFormation)
- Herramientas de orquestación (Kubernetes, Docker Swarm)
- Herramientas de gestión de certificados (Let's Encrypt, AWS Certificate Manager)

■ Resultados esperados

- Aplicación desplegada en plataforma cloud
- Infraestructura configurada y escalable
- Certificados SSL configurados y renovación automatizada

Función 7: Despliegue, Infraestructura y Operaciones

Actividad 7.4 Containerizar aplicaciones con Docker

- Crear y gestionar contenedores Docker para empaquetar y distribuir la aplicación
  - ■ Escribir Dockerfiles optimizados para aplicación
  - ■ Crear imágenes Docker eficientes y seguras
  - ■ Gestionar versiones de imágenes Docker
- Escribir Dockerfiles optimizados y gestionar imágenes de contenedor
  - ■ Utilizar multi-stage builds para reducir tamaño de imagen
  - ■ Implementar best practices de seguridad en Dockerfiles
  - ■ Optimizar capas de imagen para caché
- Utilizar Docker Compose para orquestar entornos multi-contenedor
  - ■ Crear archivos docker-compose.yml para desarrollo
  - ■ Orquestar múltiples servicios (app, base de datos, caché)

■ Gestionar volúmenes y redes de contenedores

■ Medios y recursos

- Docker y Docker Compose
- Registros de imágenes (Docker Hub, ECR, GCR)
- Herramientas de escaneo de seguridad (Trivy, Snyk)
- Documentación de Docker best practices

■ Resultados esperados

- Dockerfiles optimizados y seguros
- Imágenes Docker creadas y versionadas
- Docker Compose configurado para desarrollo

Función 7: Despliegue, Infraestructura y Operaciones

Actividad 7.5 Monitorizar aplicaciones y analizar logs

- Implementar y gestionar herramientas de monitorización para supervisar el estado de la aplicación
  - ■ Configurar herramientas de monitorización (New Relic, Datadog, Sentry)
  - ■ Implementar instrumentación de código para recopilar métricas
  - ■ Crear dashboards de monitorización
- Configurar alertas ante anomalías, caídas o degradaciones del servicio
  - ■ Definir umbrales de alerta para métricas críticas
  - ■ Configurar notificaciones (email, Slack, PagerDuty)
  - ■ Crear runbooks para respuesta a incidentes
- Revisar y analizar los registros de la aplicación para identificar errores y patrones anómalos
  - ■ Centralizar logs con herramientas (ELK Stack, Splunk, CloudWatch)
  - ■ Buscar y analizar logs para debugging
  - ■ Identificar patrones de error y anomalías

■ Medios y recursos

- Herramientas de monitorización (New Relic, Datadog, Prometheus)
- Herramientas de gestión de logs (ELK Stack, Splunk, CloudWatch)
- Herramientas de error tracking (Sentry, Rollbar)
- Herramientas de alertas (PagerDuty, Opsgenie)

■ Resultados esperados

- Monitorización implementada con dashboards
- Alertas configuradas para anomalías
- Logs centralizados y analizables

Función 8: Mantenimiento, Evolución y Gestión de Incidencias

Función 8: Mantenimiento, Evolución y Gestión de Incidencias

Actividad 8.1 Mantener aplicaciones web en producción

- Realizar tareas de mantenimiento correctivo, preventivo y evolutivo sobre aplicaciones web
  - Corregir bugs reportados en producción
  - Realizar actualizaciones de seguridad y parches
  - Implementar mejoras y nuevas funcionalidades
- Analizar el código legado para comprender su funcionamiento y proponer mejoras
  - Documentar código legado para facilitar comprensión
  - Identificar deuda técnica y oportunidades de refactorización
  - Proponer mejoras de arquitectura y rendimiento
- Garantizar la continuidad del servicio y la estabilidad de las aplicaciones existentes
  - Implementar monitorización y alertas
  - Mantener backups y planes de recuperación
  - Realizar pruebas de regresión antes de cambios

Medios y recursos

- Herramientas de gestión de incidencias (Jira, Azure DevOps)
- Herramientas de monitorización (New Relic, Datadog)
- Herramientas de testing (Jest, Mocha, Pytest)
- Documentación de aplicaciones

Resultados esperados

- Bugs corregidos y estabilidad mantenida
- Código legado documentado y mejorado
- Continuidad del servicio garantizada

Función 8: Mantenimiento, Evolución y Gestión de Incidencias

Actividad 8.2 Gestionar incidencias críticas de producción

- Responder y resolver de forma urgente los incidentes que afectan al funcionamiento de la aplicación
  - Activar protocolo de respuesta a incidentes
  - Comunicar el estado del incidente a stakeholders
  - Coordinar esfuerzos de resolución del equipo
- Diagnosticar la causa raíz del incidente, aplicar una solución temporal (hotfix) y planificar la solución definitiva
  - Analizar logs y métricas para identificar causa raíz
  - Implementar hotfix temporal para restaurar servicio
  - Planificar solución definitiva para prevenir recurrencia
- Documentar el incidente y las acciones tomadas en un post-mortem
  - Crear reporte de incidente con timeline y acciones
  - Realizar sesión de post-mortem con el equipo
  - Documentar lecciones aprendidas y mejoras

Medios y recursos

- Herramientas de gestión de incidencias (PagerDuty, Opsgenie)

- Herramientas de monitorización (New Relic, Datadog, Sentry)
- Herramientas de comunicación (Slack, Teams)
- Documentación de runbooks de incidentes

■ **Resultados esperados**

- Incidente resuelto en tiempo mínimo
- Hotfix aplicado y servicio restaurado
- Post-mortem documentado con mejoras

Función 8: Mantenimiento, Evolución y Gestión de Incidencias

Actividad 8.3 Actualizar dependencias y gestionar versionado de APIs

- Revisar y actualizar periódicamente las dependencias del proyecto
  - ■ Identificar dependencias desactualizadas
  - ■ Evaluar impacto de actualizaciones
  - ■ Actualizar dependencias de forma gradual
- Gestionar conflictos de versiones y asegurar la compatibilidad tras las actualizaciones
  - ■ Resolver conflictos de versiones entre dependencias
  - ■ Ejecutar pruebas de regresión después de actualizaciones
  - ■ Documentar cambios de versión
- Utilizar herramientas como Dependabot o Renovate para automatizar actualizaciones
  - ■ Configurar Dependabot para actualizaciones automáticas
  - ■ Revisar y aprobar pull requests de dependencias
  - ■ Monitorizar seguridad de dependencias

■ **Medios y recursos**

- Herramientas de gestión de dependencias (npm, pip, Maven)
- Herramientas de automatización (Dependabot, Renovate)
- Herramientas de análisis de seguridad (Snyk, npm audit)
- Documentación de versionado semántico

■ **Resultados esperados**

- Dependencias actualizadas regularmente
- Compatibilidad mantenida tras actualizaciones
- Automatización de actualizaciones configurada

Función 8: Mantenimiento, Evolución y Gestión de Incidencias

Actividad 8.4 Gestionar evolución y versionado de APIs

- Planificar y gestionar la evolución de las APIs para mantener la compatibilidad con clientes existentes
  - ■ Planificar cambios de API de forma compatible
  - ■ Mantener versiones antiguas de API durante período de transición
  - ■ Documentar cambios de API
- Implementar estrategias de versionado de API (URL versioning, header versioning, etc.)
  - ■ Implementar versionado en URL (/v1/, /v2/)
  - ■ Implementar versionado en headers (Accept-Version)
  - ■ Soportar múltiples versiones de API simultáneamente
- Comunicar los cambios y deprecaciones a los consumidores de la API
  - ■ Publicar changelog de API

■ Notificar deprecaciones con período de transición
■ Proporcionar guías de migración para clientes

■ Medios y recursos

- Herramientas de documentación de API (Swagger, OpenAPI)
- Herramientas de versionado (Git, semantic versioning)
- Herramientas de comunicación (email, blog, documentación)
- Herramientas de testing de compatibilidad

■ Resultados esperados

- Estrategia de versionado de API definida
- Múltiples versiones de API soportadas
- Cambios y deprecaciones comunicados a clientes

Función 9: Documentación, Comunicación y Colaboración

Función 9: Documentación, Comunicación y Colaboración

Actividad 9.1 Documentar componentes, APIs y arquitectura

- Redactar documentación técnica clara y actualizada para los componentes de la aplicación
  - ■ Documentar API de componentes (props, eventos, métodos)
  - ■ Crear ejemplos de uso de componentes
  - ■ Mantener documentación sincronizada con código
- Documentar endpoints de la API con herramientas como Swagger/OpenAPI
  - ■ Crear especificaciones OpenAPI para endpoints
  - ■ Documentar parámetros, respuestas y códigos de error
  - ■ Proporcionar ejemplos de solicitudes y respuestas
- Utilizar herramientas como Storybook para documentar componentes de UI interactivamente
  - ■ Crear historias de Storybook para componentes
  - ■ Documentar variantes y estados de componentes
  - ■ Proporcionar ejemplos de uso interactivos

■ Medios y recursos

- Herramientas de documentación (Storybook, Swagger UI, ReDoc)
- Herramientas de generación de documentación (JSDoc, Sphinx)
- Herramientas de versionado (Git, GitHub Pages)
- Plantillas de documentación

■ Resultados esperados

- Documentación de componentes en Storybook
- Documentación de APIs en Swagger/OpenAPI
- Documentación técnica clara y actualizada

Función 9: Documentación, Comunicación y Colaboración

Actividad 9.2 Colaborar con diseñadores y equipos multidisciplinarios

- Trabajar estrechamente con diseñadores UX/UI para implementar fielmente los diseños
  - ■ Revisar diseños y prototipos con diseñadores
  - ■ Proporcionar feedback sobre viabilidad técnica
  - ■ Implementar diseños con fidelidad visual
- Participar en sesiones de diseño y revisión de prototipos
  - ■ Asistir a sesiones de diseño y brainstorming
  - ■ Revisar prototipos y proporcionar feedback técnico
  - ■ Colaborar en decisiones de diseño
- Asegurar la coherencia entre el diseño visual y la implementación técnica
  - ■ Validar que la implementación coincide con diseño
  - ■ Documentar desviaciones y justificaciones
  - ■ Mantener consistencia visual en toda la aplicación

■ Medios y recursos

- Herramientas de diseño (Figma, Adobe XD, Sketch)
- Herramientas de prototipado (InVision, Framer)

- Herramientas de colaboración (Miro, Mural)
- Sistemas de diseño y guías de estilo

■ Resultados esperados

- Diseños implementados con fidelidad visual
- Colaboración efectiva con diseñadores
- Coherencia entre diseño e implementación

Función 9: Documentación, Comunicación y Colaboración

Actividad 9.3 Comunicar con stakeholders y coordinar tareas de desarrollo

- Participar en reuniones con product managers y stakeholders de negocio
  - ■ Asistir a reuniones de planificación y revisión
  - ■ Comunicar avance técnico de forma clara
  - ■ Proporcionar estimaciones realistas de esfuerzo
- Comunicar el avance técnico y las limitaciones de forma comprensible para perfiles no técnicos
  - ■ Traducir conceptos técnicos a lenguaje de negocio
  - ■ Explicar trade-offs técnicos y sus implicaciones
  - ■ Proporcionar recomendaciones basadas en análisis técnico
- Organizar y distribuir las tareas de desarrollo dentro del equipo
  - ■ Utilizar herramientas de gestión de proyectos (Jira, Trello, Linear)
  - ■ Distribuir tareas según capacidades del equipo
  - ■ Resolver dependencias técnicas entre desarrolladores

■ Medios y recursos

- Herramientas de gestión de proyectos (Jira, Trello, Linear, Azure DevOps)
- Herramientas de comunicación (Slack, Teams, email)
- Documentación de requisitos y especificaciones
- Métricas de progreso y reportes

■ Resultados esperados

- Comunicación efectiva con stakeholders
- Tareas distribuidas y coordinadas
- Alineación técnica con objetivos de negocio

Función 10: Planificación, Estimación y Gestión de Proyectos

Función 10: Planificación, Estimación y Gestión de Proyectos

Actividad 10.1 Estimar esfuerzo y participar en planificación de sprints

- Contribuir a la estimación del esfuerzo y complejidad de las tareas de desarrollo
  - Analizar tareas para identificar complejidad
  - Estimar esfuerzo basado en experiencia previa
  - Justificar estimaciones con argumentos técnicos
- Utilizar técnicas de estimación ágil como planning poker o story points
  - Participar en sesiones de planning poker
  - Asignar story points a tareas
  - Calibrar estimaciones con el equipo
- Participar en la planificación de sprints para definir y priorizar las tareas
  - Asistir a reuniones de planificación de sprint
  - Seleccionar tareas según capacidad del equipo
  - Definir objetivos del sprint

■ Medios y recursos

- Herramientas de gestión de proyectos (Jira, Azure DevOps)
- Técnicas de estimación (planning poker, story points)
- Histórico de velocidad del equipo
- Documentación de requisitos

■ Resultados esperados

- Estimaciones realistas de esfuerzo
- Sprints planificados con tareas priorizadas
- Velocidad del equipo mejorada

Función 10: Planificación, Estimación y Gestión de Proyectos

Actividad 10.2 Priorizar tareas y gestionar deuda técnica

- Participar en la priorización de tareas según el valor de negocio y la capacidad del equipo
  - Evaluar valor de negocio de cada tarea
  - Considerar dependencias técnicas
  - Priorizar según impacto y urgencia
- Gestionar el backlog técnico y asegurar que las deudas técnicas se abordan de forma planificada
  - Identificar deuda técnica en el código
  - Documentar deuda técnica en backlog
  - Asignar tiempo para reducir deuda técnica
- Adaptar las prioridades ante cambios de requisitos o incidentes
  - Responder a cambios de requisitos de negocio
  - Priorizar incidentes críticos
  - Replanificar sprints cuando sea necesario

■ Medios y recursos

- Herramientas de gestión de proyectos (Jira, Azure DevOps)
- Métricas de valor de negocio

- Análisis de deuda técnica
- Documentación de requisitos

■ **Resultados esperados**

- Tareas priorizadas según valor de negocio
- Deuda técnica gestionada y reducida
- Adaptación a cambios de requisitos

Función 11: Control de Versiones y Gestión de Código

Función 11: Control de Versiones y Gestión de Código

Actividad 11.1 Gestionar control de versiones y estrategias de branching

- Utilizar sistemas de control de versiones (Git) para gestionar el historial del código fuente
  - ■ Crear commits con mensajes descriptivos
  - ■ Mantener historial limpio y coherente
  - ■ Utilizar tags para marcar versiones
- Aplicar estrategias de branching (Git Flow, trunk-based development, etc.)
  - ■ Crear ramas para features, bugfixes y releases
  - ■ Mantener rama main/master estable
  - ■ Implementar convenciones de nomenclatura de ramas
- Gestionar merges y conflictos de forma efectiva
  - ■ Resolver conflictos de merge correctamente
  - ■ Realizar code reviews antes de merge
  - ■ Mantener historial de commits limpio

■ Medios y recursos

- Sistemas de control de versiones (Git, GitHub, GitLab)
- Herramientas de gestión de ramas (GitHub, GitLab, Bitbucket)
- Herramientas de análisis de código (SonarQube, CodeClimate)
- Documentación de estrategias de branching

■ Resultados esperados

- Historial de Git limpio y coherente
- Estrategia de branching implementada
- Merges realizados sin conflictos

Función 12: Funcionalidades Avanzadas y Experiencia de Usuario

Función 12: Funcionalidades Avanzadas y Experiencia de Usuario

Actividad 12.1 Implementar comunicación en tiempo real

- Desarrollar funcionalidades de comunicación en tiempo real utilizando WebSockets o tecnologías similares
  - ■ Implementar servidor WebSocket
  - ■ Crear cliente WebSocket en navegador
  - ■ Gestionar conexiones persistentes
- Gestionar la conexión persistente entre cliente y servidor y manejar reconexiones
  - ■ Implementar reconexión automática ante desconexión
  - ■ Gestionar heartbeats para mantener conexión viva
  - ■ Manejar timeouts y errores de conexión
- Aplicar casos de uso como notificaciones push, chats o actualizaciones en vivo
  - ■ Implementar sistema de notificaciones en tiempo real
  - ■ Crear funcionalidad de chat en vivo
  - ■ Implementar actualizaciones en vivo de datos

■ Medios y recursos

- Librerías de WebSocket (Socket.io, SockJS, ws)
- Herramientas de testing (Cypress, Playwright)
- Herramientas de monitorización (New Relic, Datadog)
- Documentación de protocolos WebSocket

■ Resultados esperados

- Comunicación en tiempo real implementada
- Reconexión automática funcionando
- Casos de uso de tiempo real implementados

Función 12: Funcionalidades Avanzadas y Experiencia de Usuario

Actividad 12.2 Implementar internacionalización y localización

- Adaptar la aplicación web para soportar múltiples idiomas y configuraciones regionales
  - ■ Extraer textos de la aplicación a archivos de traducción
  - ■ Implementar soporte para múltiples idiomas
  - ■ Adaptar formatos de fecha, moneda y números según región
- Utilizar librerías de i18n (i18next, vue-i18n, etc.) para gestionar las traducciones
  - ■ Configurar librerías de i18n
  - ■ Crear archivos de traducción para cada idioma
  - ■ Implementar cambio dinámico de idioma
- Asegurar que la interfaz se adapta correctamente a los distintos idiomas y culturas
  - ■ Validar que textos traducidos caben en la interfaz
  - ■ Adaptar direccionalidad (RTL para árabe, hebreo)
  - ■ Considerar diferencias culturales en diseño

■ **Medios y recursos**

- Librerías de i18n (i18next, vue-i18n, react-i18next)
- Herramientas de gestión de traducciones (Crowdin, Lokalise)
- Herramientas de testing de i18n
- Documentación de estándares de localización

■ **Resultados esperados**

- Aplicación soporta múltiples idiomas
- Traducciones gestionadas con i18n
- Interfaz adaptada a diferentes culturas y regiones

Función 12: Funcionalidades Avanzadas y Experiencia de Usuario

Actividad 12.3 Implementar Progressive Web Apps (PWA)

- Desarrollar funcionalidades de Progressive Web App como service workers y caché offline
  - ■ Crear service worker para interceptar solicitudes
  - ■ Implementar estrategias de caché (cache-first, network-first)
  - ■ Permitir funcionamiento offline
- Permitir la instalación de la aplicación en dispositivos móviles
  - ■ Crear manifest.json con metadatos de aplicación
  - ■ Implementar icono de instalación en navegador
  - ■ Permitir instalación en pantalla de inicio
- Optimizar la experiencia de usuario en contextos de conectividad limitada
  - ■ Implementar indicadores de estado de conexión
  - ■ Sincronizar datos cuando se restaura conexión
  - ■ Proporcionar experiencia degradada en modo offline

■ **Medios y recursos**

- APIs de Service Worker
- Herramientas de PWA (Workbox, PWA Builder)
- Herramientas de testing (Lighthouse, PWA Audit)
- Documentación de PWA

■ **Resultados esperados**

- Service worker implementado y funcionando
- Aplicación instalable en dispositivos móviles
- Funcionamiento offline implementado

ANEXO III Marco de competencias

Bloque 1 - Análisis y Diseño de Soluciones Web

Objetivo pedagógico

Capacitar al desarrollador para analizar requisitos, diseñar arquitecturas escalables y seleccionar tecnologías adecuadas en proyectos web complejos.

Actividades

Actividad 1.1 Recopilar y analizar requisitos funcionales con stakeholders
Actividad 1.2 Analizar requisitos técnicos y restricciones de infraestructura
Actividad 1.3 Diseñar la arquitectura global de la aplicación web
Actividad 1.4 Definir la estructura modular del front-end
Actividad 1.5 Definir la estructura modular del back-end
Actividad 1.6 Seleccionar tecnologías y frameworks adecuados

Competencias

Competencias	Indicadores
C1 - Analizar requisitos funcionales y técnicos de una solución web para identificar restricciones y viabilidad técnica	Documenta requisitos funcionales con criterios de aceptación claros y medibles Evalúa restricciones técnicas y presenta análisis de viabilidad con justificación de decisiones Comunica con stakeholders para validar comprensión de requisitos y obtener retroalimentación
C2 - Diseñar la arquitectura de una solución web completa considerando patrones de diseño y separación de responsabilidades	Documenta arquitectura con diagramas que muestran componentes, capas y flujos de datos Aplica patrones arquitectónicos (MVC, MVVM, microservicios) según requisitos técnicos Justifica decisiones de arquitectura considerando escalabilidad, mantenibilidad y rendimiento
C3 - Seleccionar tecnologías y frameworks adecuados para cada capa de la aplicación web basándose en requisitos y restricciones	Evalúa múltiples opciones de tecnologías comparando características, comunidad y compatibilidad Justifica selección de frameworks con argumentos técnicos documentados Valida que las tecnologías seleccionadas cumplen con requisitos funcionales y no funcionales

Saberes asociados

Análisis de Requisitos Funcionales y Técnicos

Análisis de requisitos funcionales

- Identificación de funcionalidades requeridas
- Documentación de casos de uso
- Criterios de aceptación medibles
- Validación con stakeholders
- Especificaciones de comportamiento esperado

Análisis de requisitos técnicos

- Evaluación de restricciones técnicas
- Análisis de limitaciones de infraestructura
- Evaluación de viabilidad técnica
- Identificación de riesgos técnicos
- Estimación de complejidad técnica

Comunicación con stakeholders

- Sesiones de validación de requisitos
- Documentación de cambios de requisitos
- Clarificación de expectativas
- Gestión de cambios de alcance
- Obtención de retroalimentación

Diseño de Arquitectura Web

Patrones de diseño arquitectónico

- Arquitectura en capas (MVC, MVVM)
- Patrones de separación de responsabilidades
- Arquitectura de microservicios
- Patrones de comunicación entre componentes
- Patrones de flujo de datos

Estructura de front-end

- Organización de componentes
- Gestión de estado de aplicación
- Enrutamiento y navegación
- Separación de capas de presentación
- Convenciones de nomenclatura

Estructura de back-end

- Organización de servicios
- Patrones de capas de negocio
- Diseño de APIs REST
- Gestión de acceso a datos
- Separación de responsabilidades en servidor

Documentación de arquitectura

- Diagramas de componentes
- Diagramas de flujo de datos
- Especificaciones de interfaces
- Justificación de decisiones arquitectónicas
- Documentación de patrones utilizados

Selección de Tecnologías y Frameworks

Evaluación de tecnologías

- Análisis comparativo de opciones
- Evaluación de características técnicas
- Análisis de comunidad y soporte

- Evaluación de compatibilidad
- Análisis de rendimiento y escalabilidad

Selección de frameworks front-end

- Evaluación de frameworks de UI (React, Vue, Angular)
- Selección de librerías CSS (Bootstrap, Tailwind)
- Evaluación de herramientas de estado
- Selección de librerías de enrutamiento
- Evaluación de herramientas de testing

Selección de frameworks back-end

- Evaluación de frameworks de servidor (Express, NestJS, Django)
- Selección de ORMs y drivers de base de datos
- Evaluación de herramientas de autenticación
- Selección de librerías de validación
- Evaluación de herramientas de logging y monitorización

Justificación de decisiones tecnológicas

- Documentación de criterios de selección
- Análisis de pros y contras
- Evaluación de riesgos técnicos
- Consideración de mantenibilidad a largo plazo
- Alineación con estándares del equipo

Principios de Diseño y Patrones

Principios SOLID

- Responsabilidad única
- Abierto/cerrado
- Sustitución de Liskov
- Segregación de interfaz
- Inversión de dependencia

Patrones de arquitectura

- Patrón MVC
- Patrón MVVM
- Patrón de capas
- Patrón de repositorio
- Patrón de inyección de dependencias

Escalabilidad y mantenibilidad

- Diseño para crecimiento futuro
- Consideraciones de rendimiento
- Facilidad de mantenimiento
- Documentación de decisiones
- Flexibilidad para cambios

Especificaciones Técnicas y Documentación

Documentación de especificaciones

- Especificaciones funcionales detalladas
- Especificaciones técnicas de implementación
- Diagramas de arquitectura
- Modelos de datos
- Flujos de procesos

Definición de interfaces

- Especificación de APIs REST

- Definición de contratos de datos
- Especificación de eventos
- Definición de protocolos de comunicación
- Documentación de formatos de datos

Criterios de aceptación

- Definición de criterios medibles
- Especificación de casos de prueba
- Definición de métricas de éxito
- Documentación de restricciones
- Especificación de requisitos no funcionales

Comunicación Técnica y Justificación de Decisiones

Justificación de decisiones arquitectónicas

- Presentación de argumentos técnicos
- Análisis de alternativas consideradas
- Documentación de trade-offs
- Explicación de restricciones
- Comunicación de riesgos identificados

Validación con stakeholders

- Presentación de propuestas de diseño
- Obtención de aprobación de arquitectura
- Comunicación de cambios de requisitos
- Gestión de expectativas
- Documentación de decisiones acordadas

Documentación clara y accesible

- Escritura de especificaciones comprensibles
- Creación de diagramas efectivos
- Uso de lenguaje técnico preciso
- Organización lógica de información
- Ejemplos y casos de uso

Habilidades actitudinales

- Documenta requisitos funcionales con criterios de aceptación claros y medibles
- Comunica con stakeholders para validar comprensión de requisitos y obtener retroalimentación
- Justifica decisiones de arquitectura considerando escalabilidad, mantenibilidad y rendimiento
- Evalúa múltiples opciones de tecnologías comparando características, comunidad y compatibilidad

Justificación

Este bloque agrupa las competencias fundamentales de planificación y diseño que preceden a toda implementación. Los requisitos funcionales y técnicos (C1) establecen el alcance del proyecto, la arquitectura (C2) define la estructura general del sistema, y la selección de tecnologías (C3) determina las herramientas concretas. Estas competencias son coherentes funcionalmente porque todas contribuyen a la toma de decisiones estratégicas que impactan el resto del desarrollo. Los entregables incluyen documentos de especificación, diagramas arquitectónicos y justificaciones técnicas. La evaluación se realiza mediante revisión de documentación de diseño y validación de decisiones técnicas con stakeholders.

Ubicación en la progresión

Este bloque constituye la base del desarrollo web full stack, donde el desarrollador aprende a traducir necesidades de negocio en soluciones técnicas viables. Se posiciona al inicio del ciclo de vida del proyecto, antes de cualquier implementación, permitiendo que decisiones tempranas sean bien fundamentadas. El dominio de estas competencias es esencial para que desarrolladores junior evolucionen hacia roles de arquitecto o líder técnico. La progresión continúa hacia la implementación específica de interfaces y servicios, donde las decisiones de diseño se materializan en código.

Bloque 2 - Desarrollo de Interfaces de Usuario y Componentes Front-End

Objetivo pedagógico

Desarrollar habilidades para crear interfaces responsivas, accesibles y de alto rendimiento, con componentes reutilizables y gestión de estado eficiente.

Actividades

Actividad 2.1 Desarrollar interfaces de usuario con HTML, CSS y frameworks de UI
Actividad 2.2 Diseñar e implementar componentes reutilizables
Actividad 2.3 Implementar lógica de negocio en el cliente
Actividad 2.4 Gestionar estado global y local de la aplicación
Actividad 2.5 Configurar enrutamiento y navegación entre vistas
Actividad 2.6 Optimizar rendimiento de carga y accesibilidad web

Competencias

Competencias	Indicadores
C4 - Desarrollar interfaces de usuario responsivas y accesibles que se adapten a múltiples dispositivos	Implementa diseño responsive con media queries y frameworks CSS que funciona en dispositivos móviles, tablets y desktop Cumple estándares WCAG de accesibilidad incluyendo navegación por teclado y atributos ARIA Verifica compatibilidad cross-browser en navegadores principales (Chrome, Firefox, Safari, Edge)
C5 - Implementar componentes reutilizables y modularizados con documentación clara para facilitar mantenimiento	Crea componentes con interfaz clara, props documentadas y ejemplos de uso Organiza componentes en estructura modular con convenciones de nomenclatura consistentes Asegura testabilidad de componentes mediante pruebas unitarias con cobertura mínima del 80%
C6 - Programar lógica de negocio en cliente validando datos y optimizando la experiencia de usuario	Implementa validación de datos en cliente con mensajes de error claros y accesibles Optimiza experiencia de usuario mediante feedback visual inmediato y manejo de estados de carga Aplica principios de usabilidad en la lógica de negocio del front-end
C15 - Gestionar estado de aplicación compleja utilizando patrones de gestión de estado y optimizando re-renderizados	Implementa gestión de estado con Redux, Zustand, Pinia o Context API según arquitectura Optimiza re-renderizados evitando actualizaciones innecesarias mediante memoización Estructura estado de forma normalizada facilitando actualizaciones y debugging

Competencias	Indicadores
C16 - Implementar navegación entre vistas con rutas protegidas y dinámicas gestionando historial del navegador	Configura enrutamiento con React Router, Vue Router u equivalente incluyendo rutas anidadas Implementa rutas protegidas verificando autenticación y autorización antes de acceso Gestiona historial del navegador permitiendo navegación atrás/adelante correctamente
C17 - Optimizar rendimiento de carga implementando lazy loading, code splitting y medición de métricas	Implementa lazy loading de componentes y recursos mejorando tiempo de carga inicial Aplica code splitting dividiendo bundle en chunks cargados bajo demanda Mide rendimiento con herramientas (Lighthouse, WebPageTest) alcanzando puntuaciones objetivo

Saberes asociados

Diseño Responsivo y Adaptativo

CSS Responsive y Media Queries

- Media queries para diferentes tamaños de pantalla
- Unidades relativas (em, rem, vw, vh)
- Flexbox y Grid para layouts adaptativos
- Breakpoints y estrategias mobile-first
- Pruebas de responsividad en múltiples dispositivos

Frameworks CSS

- Bootstrap: componentes y sistema de grid
- Tailwind CSS: utilidades y configuración
- Comparación de frameworks CSS
- Personalización de temas y estilos
- Optimización de CSS en producción

Compatibilidad Cross-Browser

- Pruebas en navegadores diferentes
- Prefijos de navegador (-webkit, -moz, -ms)
- Polyfills para características no soportadas
- Degradación elegante
- Herramientas de testing cross-browser

Arquitectura de Componentes y Modularidad

Componentes Reutilizables

- Diseño de componentes con interfaz clara
- Props y propiedades de componentes
- Composición de componentes
- Componentes controlados vs no controlados
- Patrones de componentes (presentacionales, contenedores)

Documentación de Componentes

- Storybook para documentación visual
- Ejemplos de uso de componentes
- Especificación de props y tipos
- Guías de estilo y patrones
- Versionado de componentes

Organización de Módulos Front-End

- Estructura de carpetas y convenciones
- Separación de responsabilidades
- Importaciones y dependencias
- Modularidad y reutilización
- Escalabilidad de arquitectura

Gestión de Estado y Lógica de Cliente

Gestión de Estado Compleja

- Patrones de gestión de estado (Redux, Zustand, Pinia)
- Actions, reducers y selectors
- Normalización de estado
- Middleware y efectos secundarios
- Debugging de estado

Optimización de Re-Renderizados

- Memoización de componentes (React.memo, useMemo)
- Optimización de callbacks (useCallback)
- Evitar prop drilling
- Context API para estado global
- Herramientas de profiling de rendimiento

Lógica de Negocio en Cliente

- Validación de datos en cliente
- Cálculos y transformaciones de datos
- Manejo de errores en cliente
- Sincronización de estado con servidor
- Optimistic updates

Enrutamiento y Navegación

Enrutamiento de Aplicaciones

- React Router, Vue Router, Next.js routing
- Rutas dinámicas y parámetros
- Rutas protegidas y autenticación
- Lazy loading de rutas
- Gestión del historial del navegador

Navegación Avanzada

- Transiciones entre vistas
- Scroll restoration
- Deep linking
- Breadcrumbs y navegación contextual
- Manejo de estado en navegación

Optimización de Rendimiento y Carga

Optimización de Rendimiento de Carga

- Lazy loading de componentes y imágenes
- Code splitting y bundle analysis
- Compresión de recursos
- Caché de navegador
- Service Workers y PWA

Medición de Métricas de Rendimiento

- Core Web Vitals (LCP, FID, CLS)
- Lighthouse y herramientas de análisis

- Performance API del navegador
- Monitorización en producción
- Análisis de waterfall de red

Accesibilidad Web y Estándares WCAG

Estándares WCAG

- Niveles de conformidad WCAG (A, AA, AAA)
- Criterios de éxito WCAG
- Auditorías de accesibilidad
- Herramientas de validación de accesibilidad
- Mejora continua de accesibilidad

Atributos ARIA y Semántica

- Roles ARIA
- Propiedades ARIA (aria-label, aria-describedby)
- Estados ARIA (aria-expanded, aria-selected)
- Semántica HTML5
- Landmarks y estructura de página

Navegación por Teclado

- Orden de tabulación (tabindex)
- Focus management
- Atajos de teclado
- Indicadores visuales de focus
- Testing con teclado

Habilidades actitudinales

- Implementa diseño responsive con media queries y frameworks CSS que funciona en dispositivos móviles, tablets y desktop
- Cumple estándares WCAG de accesibilidad incluyendo navegación por teclado y atributos ARIA
- Crea componentes con interfaz clara, props documentadas y ejemplos de uso
- Optimiza re-renderizados evitando actualizaciones innecesarias mediante memoización
- Implementa enrutamiento con rutas protegidas y dinámicas gestionando historial del navegador
- Implementa lazy loading de componentes y recursos mejorando tiempo de carga inicial

Justificación

Este bloque integra todas las competencias relacionadas con la capa de presentación y experiencia de usuario. Las interfaces responsivas (C4) y componentes reutilizables (C5) forman la base visual, la lógica de negocio en cliente (C6) añade interactividad, la gestión de estado (C15) coordina datos complejos, el enrutamiento (C16) organiza la navegación, y la optimización de rendimiento (C17) asegura experiencia fluida. Estas competencias son coherentes porque todas contribuyen a una experiencia de usuario cohesiva y eficiente. Los entregables incluyen componentes funcionales, interfaces responsivas y aplicaciones con buen rendimiento. La evaluación se realiza mediante testing en múltiples dispositivos, análisis de rendimiento y validación de accesibilidad.

Ubicación en la progresión

Este bloque representa la capa de presentación del desarrollo full stack, donde el desarrollador materializa el diseño en código funcional. Se construye sobre las decisiones arquitectónicas del bloque anterior, implementando la estructura de front-end definida. La progresión incluye desde interfaces básicas hasta aplicaciones complejas con gestión de estado sofisticada. Dominar este bloque permite al desarrollador

crear experiencias de usuario de calidad, siendo fundamental para roles especializados en front-end o full stack.

Bloque 3 - Desarrollo de Lógica de Negocio y APIs Back-End

Objetivo pedagógico

Capacitar al desarrollador para implementar lógica de negocio robusta en servidor, construir APIs REST bien diseñadas y gestionar integraciones con servicios externos.

Actividades

Actividad 3.1 Implementar autenticación y autorización
Actividad 3.2 Programar lógica de negocio en el servidor
Actividad 3.3 Diseñar e implementar APIs REST
Actividad 3.4 Integrar servicios externos mediante APIs
Actividad 3.5 Implementar validación de datos de entrada
Actividad 3.6 Proteger datos sensibles y credenciales

Competencias

Competencias	Indicadores
C7 - Programar lógica de negocio en servidor aplicando principios SOLID y garantizando integridad de datos	Implementa lógica de negocio siguiendo principios SOLID (responsabilidad única, abierto/cerrado, etc.) Garantiza integridad de datos mediante validaciones, transacciones y manejo de errores Documenta lógica de negocio compleja con comentarios y especificaciones técnicas
C8 - Construir APIs REST bien diseñadas con documentación clara y buenas prácticas de versionado	Diseña endpoints REST siguiendo convenciones (verbos HTTP, códigos de estado, estructura de recursos) Documenta APIs con Swagger/OpenAPI incluyendo parámetros, respuestas y ejemplos Implementa versionado de APIs manteniendo compatibilidad hacia atrás
C9 - Integrar servicios externos mediante APIs gestionando autenticación y errores de forma robusta	Implementa integración con APIs externas usando autenticación segura (OAuth, API keys, JWT) Maneja errores de integración con reintentos, timeouts y fallbacks apropiados Documenta integraciones externas con ejemplos de uso y configuración
C27 - Gestionar versionado de APIs comunicando cambios y manteniendo compatibilidad hacia atrás	Implementa versionado de APIs (URL, header) permitiendo múltiples versiones simultáneamente Comunica cambios de API a clientes con documentación de migración Mantiene compatibilidad hacia atrás durante períodos de transición

Saberes asociados

Lógica de Negocio en Servidor

Principios SOLID

- Responsabilidad única

- Abierto/cerrado
- Sustitución de Liskov
- Segregación de interfaz
- Inversión de dependencia

Integridad de Datos

- Validaciones en múltiples capas
- Transacciones de base de datos
- Manejo de errores robusto
- Consistencia de datos

Patrones de Capas

- Arquitectura MVC
- Arquitectura MVVM
- Separación de responsabilidades
- Capas de presentación, negocio y datos

Diseño de APIs REST

Convenciones REST

- Verbos HTTP (GET, POST, PUT, DELETE, PATCH)
- Códigos de estado HTTP
- Estructura de recursos
- Convenciones de nomenclatura de endpoints

Documentación de APIs

- Especificación Swagger/OpenAPI
- Parámetros y respuestas
- Ejemplos de uso
- Esquemas de datos

Buenas Prácticas de API

- Versionado de APIs
- Paginación y filtrado
- Manejo de errores
- Rate limiting

Integración de Servicios Externos

Autenticación con Servicios Externos

- OAuth 2.0
- API keys
- JWT (JSON Web Tokens)
- Autenticación básica

Manejo de Integraciones

- Llamadas a APIs externas
- Manejo de errores de integración
- Reintentos y timeouts
- Logging de integraciones

Sincronización de Datos

- Webhooks
- Polling
- Event-driven architecture
- Consistencia eventual

Versionado y Evolución de APIs

Estrategias de Versionado

- Versionado en URL
- Versionado en headers
- Versionado semántico
- Deprecación de versiones

Compatibilidad hacia Atrás

- Cambios no disruptivos
- Campos opcionales
- Valores por defecto
- Migración de clientes

Comunicación de Cambios

- Changelog de API
- Notificaciones a clientes
- Documentación de migraciones
- Período de transición

Frameworks y Tecnologías de Servidor

Frameworks Backend Populares

- Express.js (Node.js)
- Django (Python)
- Spring Boot (Java)
- Laravel (PHP)
- ASP.NET Core (.NET)

Herramientas de Desarrollo

- Postman/Insomnia para testing de APIs
- Swagger UI para documentación
- Herramientas de debugging de servidor
- Profilers de rendimiento

Patrones de Arquitectura

- Arquitectura monolítica
- Microservicios
- Arquitectura serverless
- Event sourcing

Seguridad en Backend

Validación y Sanitización

- Validación de entrada
- Prevención de inyección SQL
- Prevención de XSS
- Sanitización de datos

Autenticación y Autorización

- Sistemas de autenticación
- Control de acceso basado en roles
- Gestión de sesiones
- Tokens seguros

Prácticas de Seguridad

- OWASP Top 10
- Encriptación de datos
- Gestión de secretos
- Auditoría de seguridad

Habilidades actitudinales

- Implementa lógica de negocio siguiendo principios SOLID (responsabilidad única, abierto/cerrado, etc.)
- Garantiza integridad de datos mediante validaciones, transacciones y manejo de errores
- Diseña endpoints REST siguiendo convenciones (verbos HTTP, códigos de estado, estructura de recursos)
- Documenta APIs con Swagger/OpenAPI incluyendo parámetros, respuestas y ejemplos
- Implementa integración con APIs externas usando autenticación segura (OAuth, API keys, JWT)
- Implementa versionado de APIs manteniendo compatibilidad hacia atrás

Justificación

Este bloque agrupa las competencias de la capa de lógica de negocio del servidor. La lógica de negocio (C7) implementa reglas de negocio aplicando principios SOLID, las APIs REST (C8) exponen esta lógica de forma estándar, la integración de servicios externos (C9) extiende funcionalidad, y el versionado de APIs (C27) asegura evolución sostenible. Estas competencias son coherentes porque todas contribuyen a un backend robusto y mantenible. Los entregables incluyen endpoints funcionales, documentación de APIs y servicios integrados. La evaluación se realiza mediante testing de APIs, análisis de código y validación de integraciones.

Ubicación en la progresión

Este bloque representa la capa de lógica de negocio del servidor, donde el desarrollador implementa las reglas de negocio definidas en la arquitectura. Se construye sobre las decisiones de diseño del bloque de análisis y diseño, implementando la estructura de back-end. La progresión incluye desde APIs simples hasta sistemas complejos con múltiples integraciones. Dominar este bloque es esencial para desarrolladores full stack y especialistas en back-end.

Bloque 4 - Gestión de Datos y Bases de Datos

Objetivo pedagógico

Desarrollar habilidades para diseñar esquemas de base de datos, optimizar consultas, implementar operaciones CRUD seguras y sincronizar datos entre capas.

Actividades

Actividad 4.1 Conectar la aplicación con bases de datos
Actividad 4.2 Diseñar y optimizar consultas a bases de datos
Actividad 4.3 Desarrollar operaciones CRUD
Actividad 4.4 Sincronizar datos entre cliente y servidor, y gestionar migraciones

Competencias

Competencias	Indicadores
C10 - Validar datos de entrada en múltiples capas previniendo vulnerabilidades de inyección y XSS	Implementa validación de datos en cliente y servidor con reglas consistentes Previene vulnerabilidades OWASP Top 10 (inyección SQL, XSS, CSRF) mediante sanitización y parametrización Proporciona mensajes de error informativos sin exponer detalles técnicos sensibles
C11 - Conectar aplicaciones con bases de datos utilizando ORMs y gestionar pools de conexiones eficientemente	Configura conexiones a bases de datos con ORMs (Sequelize, TypeORM, Mongoose) o drivers nativos Implementa gestión de pools de conexiones optimizando recursos y evitando fugas de memoria Maneja errores de conexión y reconexión automática
C12 - Diseñar y optimizar consultas a bases de datos utilizando índices y análisis de planes de ejecución	Escribe consultas SQL/NoSQL optimizadas analizando planes de ejecución (EXPLAIN) Crea índices estratégicos mejorando rendimiento de consultas frecuentes Identifica y previene problemas de rendimiento como N+1 queries y full table scans
C13 - Desarrollar operaciones CRUD manteniendo integridad referencial y transaccionalidad en bases de datos	Implementa operaciones CRUD (Create, Read, Update, Delete) con validaciones de integridad Utiliza transacciones para garantizar consistencia de datos en operaciones complejas Maneja restricciones de integridad referencial y conflictos de concurrencia
C14 - Sincronizar datos entre front-end y back-end mediante WebSockets y mecanismos de caché consistente	Implementa sincronización de datos cliente-servidor en tiempo real usando WebSockets Gestiona caché de cliente manteniendo consistencia con datos del servidor Maneja conflictos de sincronización y actualización de estado distribuido

Saberes asociados

Validación de Datos y Prevención de Vulnerabilidades

Validación en múltiples capas

- Validación en cliente con reglas de negocio
- Validación en servidor con sanitización
- Validación de tipos de datos
- Validación de rangos y formatos

Prevención de inyección SQL y XSS

- Parametrización de consultas
- Escape de caracteres especiales
- Sanitización de entrada de usuario
- Prevención de cross-site scripting

Mensajes de error seguros

- Comunicación clara sin exponer detalles técnicos
- Feedback de validación al usuario
- Logging de errores sensibles

Conexión y Gestión de Bases de Datos

Configuración de conexiones

- Conexión con ORMs (Sequelize, TypeORM, Mongoose)
- Conexión con drivers nativos
- Gestión de pools de conexiones
- Manejo de errores de conexión
- Variables de entorno y credenciales seguras

Ciclo de vida de conexiones

- Apertura y cierre de conexiones
- Reutilización de conexiones en pools
- Timeout y reconexión automática
- Monitorización de conexiones activas

Optimización de Consultas y Análisis de Rendimiento

Escritura de consultas eficientes

- Consultas SQL optimizadas
- Consultas NoSQL con proyecciones
- Evitar N+1 queries
- Uso de joins y agregaciones

Análisis de planes de ejecución

- EXPLAIN en SQL
- Interpretación de planes de ejecución
- Identificación de cuellos de botella
- Estimación de costo de consultas

Creación y gestión de índices

- Índices simples y compuestos
- Índices de texto completo
- Monitorización de uso de índices
- Impacto en escritura vs lectura

Operaciones CRUD y Integridad de Datos

Operaciones básicas de datos

- Create: inserción con validación

- Read: consultas con filtros
- Update: actualización con validación
- Delete: eliminación con cascada

Integridad referencial

- Claves foráneas y relaciones
- Restricciones de integridad
- Eliminación en cascada
- Actualización en cascada

Transaccionalidad

- Transacciones ACID
- Rollback en caso de error
- Niveles de aislamiento
- Manejo de conflictos de concurrencia

Sincronización de Datos y Comunicación en Tiempo Real

WebSockets y comunicación bidireccional

- Protocolo WebSocket
- Librerías Socket.io y ws
- Conexiones persistentes
- Manejo de desconexiones

Caché y consistencia

- Estrategias de caché (Redis, Memcached)
- Invalidación de caché
- Coherencia entre cliente y servidor
- Resolución de conflictos de datos

Patrones de sincronización

- Polling y long-polling
- Server-sent events
- Sincronización optimista
- Sincronización pesimista

Seguridad en Gestión de Datos

Protección de datos sensibles

- Encriptación de datos en tránsito
- Encriptación de datos en reposo
- Hashing de contraseñas
- Cumplimiento de GDPR y privacidad

Auditoría y trazabilidad

- Logging de operaciones de datos
- Trazabilidad de cambios
- Auditoría de acceso a datos
- Retención de logs

Habilidades actitudinales

- Implementa validación de datos en cliente y servidor con reglas consistentes
- Configura conexiones a bases de datos con ORMs (Sequelize, TypeORM, Mongoose) o drivers nativos
- Escribe consultas SQL/NoSQL optimizadas analizando planes de ejecución (EXPLAIN)
- Implementa operaciones CRUD (Create, Read, Update, Delete) con validaciones de integridad
- Utiliza transacciones para garantizar consistencia de datos en operaciones complejas

- Implementa sincronización de datos cliente-servidor en tiempo real usando WebSockets

Justificación

Este bloque integra todas las competencias relacionadas con la persistencia de datos. La validación de entrada (C10) protege la integridad desde el inicio, la conexión a bases de datos (C11) establece comunicación confiable, la optimización de consultas (C12) asegura rendimiento, las operaciones CRUD (C13) implementan cambios de datos, y la sincronización (C14) mantiene consistencia entre cliente y servidor. Estas competencias son coherentes porque todas contribuyen a un sistema de datos robusto y eficiente. Los entregables incluyen esquemas de base de datos, consultas optimizadas y operaciones transaccionales. La evaluación se realiza mediante análisis de planes de ejecución, testing de integridad y validación de sincronización.

Ubicación en la progresión

Este bloque representa la capa de persistencia de datos, donde el desarrollador gestiona la información crítica de la aplicación. Se construye sobre la arquitectura definida y la lógica de negocio implementada, asegurando que datos se almacenan y recuperan de forma segura y eficiente. La progresión incluye desde operaciones simples hasta sistemas complejos con sincronización en tiempo real. Dominar este bloque es fundamental para desarrolladores full stack que necesitan entender el ciclo completo de datos.

Bloque 5 - Testing, Depuración y Aseguramiento de Calidad

Objetivo pedagógico

Capacitar al desarrollador para escribir pruebas automatizadas, ejecutar pruebas funcionales, depurar errores sistemáticamente y revisar código de calidad.

Actividades

Actividad 5.1 Escribir y ejecutar pruebas unitarias e integración
Actividad 5.2 Ejecutar pruebas funcionales y end-to-end
Actividad 5.3 Depurar errores en cliente y servidor
Actividad 5.4 Revisar código y corregir incidencias de pruebas
Actividad 5.5 Aplicar estándares de calidad de código

Competencias

Competencias	Indicadores
C18 - Escribir pruebas unitarias e integración con cobertura de código adecuada y frameworks de testing	Escribe pruebas unitarias con Jest, Vitest o Mocha cubriendo lógica crítica Implementa pruebas de integración con Supertest, Cypress o Playwright validando flujos completos Mantiene cobertura de código mínima del 80% en código crítico
C19 - Ejecutar pruebas funcionales end-to-end y reportar defectos con información detallada para resolución	Ejecuta pruebas funcionales validando comportamiento de usuario en escenarios reales Reporta defectos con pasos de reproducción, logs y evidencia visual (screenshots/videos) Verifica que defectos resueltos cumplen criterios de aceptación
C20 - Depurar errores en front-end y back-end utilizando herramientas de debugging y análisis de logs	Utiliza DevTools del navegador para debugging de JavaScript, inspección de red y performance Analiza logs de servidor identificando causa raíz de errores mediante trazas de stack Depura llamadas a API verificando payloads, headers y respuestas
C21 - Corregir incidencias detectadas en pruebas analizando causa raíz y actualizando pruebas automatizadas	Analiza defectos identificando causa raíz mediante debugging sistemático Implementa correcciones que resuelven el problema sin introducir regresiones Actualiza pruebas automatizadas para prevenir recurrencia de defectos
C22 - Revisar código de otros desarrolladores proporcionando feedback técnico constructivo y verificando estándares	Realiza code review evaluando calidad, seguridad y adherencia a estándares Proporciona feedback constructivo con sugerencias de mejora y referencias a buenas prácticas Verifica que código cumple estándares de proyecto (linting, formatting, convenciones)

Saberes asociados

Frameworks y herramientas de testing automatizado

Pruebas unitarias

- Jest, Vitest, Mocha
- Escritura de casos de prueba
- Assertions y matchers
- Mocking y stubbing
- Cobertura de código

Pruebas de integración

- Supertest para APIs
- Testing de bases de datos
- Fixtures y datos de prueba
- Transacciones de prueba

Pruebas funcionales y end-to-end

- Cypress y Playwright
- Automatización de flujos de usuario
- Esperas y sincronización
- Captura de pantallas y videos

Herramientas y técnicas de depuración

Debugging en navegador

- DevTools del navegador
- Breakpoints y stepping
- Inspección de variables
- Análisis de red y performance
- Consola de JavaScript

Debugging en servidor

- Node.js debugger
- Herramientas de profiling
- Análisis de logs
- Trazas de error y stack traces

Análisis de rendimiento

- Lighthouse y WebPageTest
- Chrome DevTools Performance
- Métricas de Core Web Vitals
- Análisis de memoria y CPU

Estándares y prácticas de calidad de código

Principios de código limpio

- Legibilidad y mantenibilidad
- Convenciones de nomenclatura
- Funciones pequeñas y enfocadas
- Evitar código duplicado

Estándares de proyecto

- Linters (ESLint, Prettier)
- Guías de estilo
- Patrones arquitectónicos
- Documentación de código

Seguridad en código

- Validación de entrada

- Prevención de inyección SQL y XSS
- Gestión segura de credenciales
- Análisis de vulnerabilidades

Análisis y resolución de defectos

Análisis de causa raíz

- Técnica de los cinco por qué
- Reproducción sistemática de defectos
- Aislamiento de variables
- Documentación de hallazgos

Reporte de defectos

- Pasos de reproducción claros
- Información del entorno
- Logs y evidencia visual
- Severidad e impacto

Corrección y validación

- Implementación de soluciones
- Actualización de pruebas
- Regresión testing
- Verificación de corrección

Revisión de código y feedback técnico

Evaluación técnica

- Análisis de lógica y algoritmos
- Verificación de estándares
- Identificación de vulnerabilidades
- Evaluación de rendimiento

Comunicación constructiva

- Feedback respetuoso y claro
- Sugerencias de mejora
- Referencias a buenas prácticas
- Preguntas que invitan a reflexión

Colaboración en revisión

- Discusión de decisiones técnicas
- Aprendizaje mutuo
- Resolución de desacuerdos
- Documentación de decisiones

Cobertura de código y métricas de calidad

Métricas de cobertura

- Cobertura de líneas
- Cobertura de ramas
- Cobertura de funciones
- Objetivos de cobertura

Herramientas de medición

- Istanbul y nyc
- Reportes de cobertura
- Integración en CI/CD
- Análisis de tendencias

Mejora continua

- Identificación de código no cubierto
- Planificación de pruebas adicionales
- Refactorización para testabilidad
- Monitoreo de calidad

Habilidades actitudinales

- Escribe pruebas unitarias con Jest, Vitest o Mocha cubriendo lógica crítica
- Implementa pruebas de integración con Supertest, Cypress o Playwright validando flujos completos
- Ejecuta pruebas funcionales validando comportamiento de usuario en escenarios reales
- Reporta defectos con pasos de reproducción, logs y evidencia visual (screenshots/videos)
- Utiliza DevTools del navegador para debugging de JavaScript, inspección de red y performance
- Analiza defectos identificando causa raíz mediante debugging sistemático
- Realiza code review evaluando calidad, seguridad y adherencia a estándares

Justificación

Este bloque agrupa todas las competencias relacionadas con la calidad del software. Las pruebas unitarias e integración (C18) validan funcionalidad a nivel técnico, las pruebas funcionales (C19) validan desde perspectiva de usuario, el debugging (C20) identifica y resuelve problemas, la corrección de defectos (C21) implementa soluciones, y el code review (C22) asegura estándares. Estas competencias son coherentes porque todas contribuyen a un producto de alta calidad. Los entregables incluyen suites de pruebas, reportes de cobertura y código revisado. La evaluación se realiza mediante análisis de cobertura de código, ejecución de pruebas y validación de correcciones.

Ubicación en la progresión

Este bloque representa las prácticas de aseguramiento de calidad que acompañan todo el desarrollo. Se integra transversalmente con otros bloques, asegurando que cada componente, API y característica cumple estándares de calidad. La progresión incluye desde pruebas unitarias simples hasta estrategias complejas de testing end-to-end. Dominar este bloque es esencial para desarrolladores que buscan crear software confiable y mantenible.

Bloque 6 - Seguridad, Rendimiento y Optimización

Objetivo pedagógico

Desarrollar habilidades para implementar sistemas de autenticación y autorización seguros, aplicar medidas de seguridad web, optimizar rendimiento del servidor y gestionar dependencias.

Actividades

Actividad 6.1 Aplicar medidas de seguridad web
Actividad 6.2 Optimizar rendimiento del servidor
Actividad 6.3 Optimizar código para escalabilidad
Actividad 6.4 Optimizar rendimiento de carga en cliente

Competencias

Competencias	Indicadores
C23 - Gestionar autenticación y autorización implementando sistemas seguros de control de acceso	Implementa autenticación segura (JWT, OAuth, sesiones) con validación de credenciales Configura autorización basada en roles/permisos verificando acceso a recursos Protege endpoints y datos sensibles mediante mecanismos de autenticación y autorización
C24 - Aplicar medidas de seguridad web implementando cabeceras HTTP y auditorías de vulnerabilidades	Configura cabeceras de seguridad HTTP (CSP, X-Frame-Options, HSTS, etc.) protegiendo contra ataques Realiza auditorías de seguridad identificando vulnerabilidades OWASP Top 10 Implementa medidas correctivas documentando cambios de seguridad
C25 - Optimizar rendimiento del servidor implementando caché y monitorización de métricas de rendimiento	Implementa estrategias de caché (Redis, Memcached) reduciendo carga de servidor Configura monitorización y alertas de rendimiento (CPU, memoria, latencia) Analiza métricas de rendimiento identificando cuellos de botella y optimizando
C26 - Actualizar dependencias y librerías resolviendo conflictos de versiones y validando compatibilidad	Actualiza dependencias manteniendo compatibilidad y evitando breaking changes Resuelve conflictos de versiones analizando dependencias transitivas Valida que actualizaciones no introducen vulnerabilidades o regresiones

Saberes asociados

Autenticación y Autorización

Sistemas de autenticación

- JWT (JSON Web Tokens)
- OAuth 2.0
- Sesiones HTTP

- Almacenamiento seguro de credenciales
- Hashing de contraseñas (bcrypt, Argon2)

Control de acceso y autorización

- Control de acceso basado en roles (RBAC)
- Control de acceso basado en atributos (ABAC)
- Permisos y privilegios
- Rutas protegidas
- Validación de autorización en servidor

Gestión de sesiones

- Ciclo de vida de sesiones
- Tokens de refresco
- Expiración de sesiones
- Revocación de tokens

Seguridad Web y Vulnerabilidades

OWASP Top 10

- Inyección SQL
- Cross-Site Scripting (XSS)
- Cross-Site Request Forgery (CSRF)
- Autenticación rota
- Exposición de datos sensibles
- Control de acceso roto
- Deserialización insegura
- Uso de componentes con vulnerabilidades conocidas
- Registro y monitorización insuficientes
- Inyección de comandos

Cabeceras de seguridad HTTP

- Content-Security-Policy (CSP)
- X-Frame-Options
- X-Content-Type-Options
- Strict-Transport-Security (HSTS)
- X-XSS-Protection
- Referrer-Policy
- Permissions-Policy

Auditorías de seguridad

- Herramientas de análisis de seguridad
- Pruebas de penetración
- Análisis de vulnerabilidades
- Reportes de seguridad
- Remediación de vulnerabilidades

Optimización de Rendimiento del Servidor

Estrategias de caché

- Caché en memoria (Redis, Memcached)
- Caché HTTP
- Caché de base de datos
- Invalidación de caché
- Estrategias de evicción (LRU, LFU)
- Consistencia de caché

Monitorización de rendimiento

- Métricas de CPU y memoria
- Latencia de respuesta
- Throughput de servidor
- Análisis de logs de rendimiento
- Herramientas de monitorización (Prometheus, Grafana, New Relic)
- Alertas de rendimiento

Optimización de código servidor

- Identificación de cuellos de botella
- Profiling de código
- Optimización de algoritmos
- Gestión eficiente de memoria
- Concurrencia y paralelismo

Gestión de Dependencias y Versionado

Gestión de versiones de dependencias

- Versionado semántico (SemVer)
- Rangos de versión (^, ~, =)
- Archivos de bloqueo (package-lock.json, yarn.lock)
- Actualización de dependencias
- Auditoría de vulnerabilidades en dependencias

Resolución de conflictos

- Conflictos de versión
- Dependencias incompatibles
- Resolución manual de conflictos
- Herramientas de resolución (npm, yarn, pnpm)

Automatización de actualizaciones

- Herramientas de actualización automática (Dependabot, Renovate)
- Validación de compatibilidad
- Testing de actualizaciones
- Estrategias de actualización gradual

Implementación de Medidas de Seguridad

Validación y sanitización de entrada

- Validación en servidor
- Sanitización de datos
- Escapado de caracteres especiales
- Validación de tipos de datos
- Prevención de inyección

Encriptación y criptografía

- Encriptación de datos en tránsito (TLS/SSL)
- Encriptación de datos en reposo
- Algoritmos de hash criptográfico
- Certificados digitales
- Gestión de claves

Prácticas de seguridad en desarrollo

- Principio de menor privilegio
- Defensa en profundidad
- Logging de eventos de seguridad
- Gestión de secretos
- Variables de entorno seguras

Patrones de Arquitectura Segura

Diseño seguro de sistemas

- Separación de responsabilidades de seguridad
- Aislamiento de componentes sensibles
- Arquitectura de confianza cero
- Seguridad por diseño
- Evaluación de riesgos

Cultura de seguridad

- Conciencia de seguridad en equipo
- Revisión de código con enfoque en seguridad
- Capacitación en seguridad
- Reportes de vulnerabilidades
- Mejora continua de seguridad

Habilidades actitudinales

- Implementa autenticación segura (JWT, OAuth, sesiones) con validación de credenciales
- Configura autorización basada en roles/permisos verificando acceso a recursos
- Configura cabeceras de seguridad HTTP (CSP, X-Frame-Options, HSTS, etc.) protegiendo contra ataques
- Realiza auditorías de seguridad identificando vulnerabilidades OWASP Top 10
- Implementa estrategias de caché (Redis, Memcached) reduciendo carga de servidor
- Configura monitorización y alertas de rendimiento (CPU, memoria, latencia)
- Actualiza dependencias manteniendo compatibilidad y evitando breaking changes

Justificación

Este bloque integra competencias críticas de seguridad y rendimiento. La autenticación y autorización (C23) protegen acceso a recursos, las medidas de seguridad web (C24) previenen vulnerabilidades, la optimización de rendimiento (C25) asegura escalabilidad, y la gestión de dependencias (C26) mantiene sistema actualizado y seguro. Estas competencias son coherentes porque todas contribuyen a un sistema seguro, rápido y mantenible. Los entregables incluyen sistemas de autenticación, configuraciones de seguridad, optimizaciones de caché y dependencias actualizadas. La evaluación se realiza mediante auditorías de seguridad, análisis de rendimiento y validación de actualizaciones.

Ubicación en la progresión

Este bloque representa aspectos transversales de seguridad y rendimiento que deben considerarse en todo el desarrollo. Se integra con otros bloques, asegurando que cada componente implementa prácticas seguras y eficientes. La progresión incluye desde autenticación básica hasta sistemas complejos de autorización y optimización avanzada. Dominar este bloque es esencial para desarrolladores que trabajan en producción.

Bloque 7 - Despliegue, Infraestructura y Operaciones

Objetivo pedagógico

Capacitar al desarrollador para configurar entornos de desarrollo, implementar CI/CD, automatizar despliegues y gestionar infraestructura en plataformas cloud.

Actividades

Actividad 7.1 Configurar entornos de desarrollo y CI/CD
Actividad 7.2 Automatizar compilaciones y despliegues
Actividad 7.3 Desplegar aplicaciones en servidores y plataformas cloud
Actividad 7.4 Containerizar aplicaciones con Docker
Actividad 7.5 Monitorizar aplicaciones y analizar logs

Competencias

Competencias	Indicadores
C29 - Configurar entornos de desarrollo con herramientas y documentación para facilitar onboarding	Configura entorno de desarrollo con Docker, variables de entorno y dependencias Documenta pasos de configuración permitiendo que nuevos desarrolladores se incorporen rápidamente Automatiza configuración mediante scripts y herramientas (npm scripts, Makefile)
C31 - Automatizar compilaciones y despliegues utilizando herramientas de bundling y estrategias de despliegue	Configura herramientas de bundling (Webpack, Vite) optimizando artefactos de compilación Automatiza proceso de build y despliegue mediante scripts y herramientas CI/CD Implementa estrategias de despliegue (blue-green, canary) minimizando downtime
C32 - Desplegar aplicaciones en plataformas cloud configurando infraestructura y escalabilidad	Despliega aplicaciones en plataformas cloud (AWS, Azure, GCP, Vercel, Heroku) Configura infraestructura incluyendo servidores, bases de datos y CDN Implementa escalabilidad automática y monitorización en producción

Saberes asociados

Automatización de Integración y Despliegue Continuo

Configuración de pipelines CI/CD

- GitHub Actions
- GitLab CI
- Jenkins
- Definición de workflows
- Triggers de ejecución
- Gestión de secretos y variables de entorno

Automatización de validaciones

- Ejecución automática de pruebas unitarias
- Ejecución automática de pruebas de integración
- Análisis estático de código (linting)
- Verificación de seguridad (SAST)
- Generación de reportes de cobertura
- Validación de estándares de código

Automatización de compilación y empaquetado

- Herramientas de bundling (Webpack, Vite, Parcel)
- Optimización de artefactos
- Minificación y compresión
- Generación de mapas de fuentes
- Versionado de artefactos

Estrategias y Automatización de Despliegue

Estrategias de despliegue

- Despliegue blue-green
- Despliegue canary
- Despliegue rolling
- Despliegue con feature flags
- Rollback automático
- Minimización de downtime

Automatización de despliegue

- Scripts de despliegue
- Orquestación de despliegue
- Despliegue en múltiples entornos
- Sincronización de configuración
- Gestión de versiones de aplicación

Monitorización post-despliegue

- Verificación de salud de aplicación
- Alertas de despliegue
- Análisis de logs de despliegue
- Métricas de éxito de despliegue

Infraestructura y Plataformas Cloud

Plataformas cloud y servicios

- AWS (EC2, RDS, S3, CloudFront)
- Azure (App Service, SQL Database, Blob Storage)
- Google Cloud Platform (Compute Engine, Cloud SQL)
- Vercel y Netlify para front-end
- Heroku para aplicaciones full stack
- Selección de plataforma según requisitos

Configuración de infraestructura

- Provisión de servidores
- Configuración de bases de datos
- Configuración de redes y firewalls
- Configuración de CDN
- Gestión de certificados SSL/TLS
- Configuración de dominios y DNS

Escalabilidad y alta disponibilidad

- Escalado automático (auto-scaling)
- Balanceo de carga
- Replicación de bases de datos
- Redundancia geográfica
- Disaster recovery
- SLA y disponibilidad

Configuración de Entornos de Desarrollo

Herramientas de desarrollo local

- Node.js y npm/yarn/pnpm
- Docker y Docker Compose
- Variables de entorno (.env)
- Gestión de versiones de herramientas (nvm, asdf)
- Configuración de IDE y editores

Automatización de configuración

- Scripts de inicialización (npm scripts, Makefile)
- Instalación de dependencias
- Configuración de base de datos local
- Seed de datos de prueba
- Documentación de pasos de configuración

Onboarding de desarrolladores

- Documentación clara de requisitos
- Guías paso a paso de configuración
- Troubleshooting común
- Validación de configuración correcta
- Acceso a recursos compartidos

Containerización y Orquestación

Docker y contenedores

- Creación de Dockerfiles
- Construcción de imágenes Docker
- Gestión de contenedores
- Docker Compose para múltiples servicios
- Registros de imágenes (Docker Hub, ECR, GCR)
- Optimización de imágenes

Orquestación de contenedores

- Kubernetes básico
- Despliegue de contenedores
- Gestión de recursos
- Networking de contenedores
- Persistencia de datos

Prácticas Operacionales y Monitorización

Monitorización y observabilidad

- Métricas de aplicación (CPU, memoria, latencia)
- Logs centralizados
- Trazas distribuidas
- Alertas y notificaciones
- Dashboards de monitorización
- Herramientas (Prometheus, ELK, Datadog, New Relic)

Gestión de incidencias

- Detección de problemas
- Escalación de incidencias
- Respuesta a incidencias
- Postmortems y análisis de causa raíz
- Documentación de incidencias

Mantenimiento operacional

- Actualizaciones de sistema
- Parches de seguridad
- Limpieza de recursos
- Gestión de backups
- Recuperación ante desastres

Habilidades actitudinales

- Configura entorno de desarrollo con Docker, variables de entorno y dependencias
- Automatiza configuración mediante scripts y herramientas (npm scripts, Makefile)
- Configura CI/CD con GitHub Actions, GitLab CI o Jenkins ejecutando pruebas automáticamente
- Implementa validaciones (linting, testing, seguridad) en pipeline de CI
- Configura herramientas de bundling (Webpack, Vite) optimizando artefactos de compilación
- Automatiza proceso de build y despliegue mediante scripts y herramientas CI/CD
- Despliega aplicaciones en plataformas cloud (AWS, Azure, GCP, Vercel, Heroku)
- Configura infraestructura incluyendo servidores, bases de datos y CDN

Justificación

Este bloque agrupa competencias de infraestructura y operaciones. La configuración de entornos (C29) facilita desarrollo consistente, CI/CD (C30) automatiza validaciones, la automatización de build y despliegue (C31) acelera entrega, y el despliegue en cloud (C32) escala aplicaciones. Estas competencias son coherentes porque todas contribuyen a un pipeline de entrega eficiente y confiable. Los entregables incluyen entornos configurados, pipelines de CI/CD, scripts de despliegue e infraestructura en cloud. La evaluación se realiza mediante validación de pipelines, análisis de despliegues y monitorización en producción.

Ubicación en la progresión

Este bloque representa las prácticas de operaciones y despliegue que cierran el ciclo de desarrollo. Se construye sobre todo el trabajo anterior, asegurando que código se entrega de forma segura y eficiente a producción. La progresión incluye desde entornos locales hasta infraestructura cloud escalable. Dominar este bloque permite a desarrolladores participar en decisiones de infraestructura y operaciones.

Bloque 8 - Mantenimiento, Evolución y Gestión de Incidencias

Objetivo pedagógico

Desarrollar habilidades para implementar internacionalización y localización, permitiendo que aplicaciones sirvan a usuarios en múltiples idiomas y regiones.

Actividades

Actividad 8.1 Mantener aplicaciones web en producción
Actividad 8.2 Gestionar incidencias críticas de producción
Actividad 8.3 Actualizar dependencias y gestionar versionado de APIs
Actividad 8.4 Gestionar evolución y versionado de APIs

Competencias

Competencias	Indicadores
C28 - Implementar internacionalización y localización adaptando contenido a múltiples idiomas y regiones	Implementa i18n usando librerías (i18next, vue-i18n) separando contenido de código Localiza contenido considerando formatos de fecha, moneda, plurales y direccionalidad Verifica que interfaz se adapta correctamente a diferentes idiomas y regiones

Saberes asociados

Internacionalización y Localización

Implementación de i18n

- Librerías de traducción (i18next, vue-i18n, react-i18next)
- Separación de contenido de código
- Gestión de archivos de traducción (JSON, YAML)
- Carga dinámica de idiomas
- Fallback de idiomas

Localización de contenido

- Formatos de fecha y hora según región
- Formatos de moneda y números
- Plurales y reglas gramaticales
- Direccionalidad de texto (RTL/LTR)
- Adaptación cultural de contenido

Testing multiidioma

- Pruebas en múltiples idiomas
- Validación de traducciones
- Testing de adaptación cultural
- Verificación de formatos localizados

Mantenimiento y Operaciones de Aplicaciones

Monitorización de aplicaciones

- Herramientas de monitorización (New Relic, Datadog, Prometheus)
- Métricas de rendimiento y disponibilidad
- Alertas y notificaciones
- Análisis de logs en producción

Gestión de incidencias

- Identificación de problemas en producción
- Escalación de incidencias
- Comunicación de estado
- Resolución y post-mortem

Evolución de aplicaciones

- Despliegues sin downtime
- Rollback de cambios
- Versionado de características
- Feature flags y canary deployments

Versionado y Evolución de APIs

Estrategias de versionado

- Versionado en URL (v1, v2)
- Versionado en headers
- Versionado semántico
- Deprecación de versiones

Compatibilidad hacia atrás

- Mantenimiento de múltiples versiones
- Migración de clientes
- Documentación de cambios
- Período de transición

Comunicación de cambios

- Changelog y release notes
- Notificación a clientes
- Documentación de migraciones
- Ejemplos de actualización

Gestión de Dependencias y Actualizaciones

Actualización de dependencias

- Herramientas de actualización (npm, yarn, pip)
- Análisis de vulnerabilidades
- Testing de compatibilidad
- Automatización de actualizaciones

Resolución de conflictos

- Conflictos de versiones
- Breaking changes
- Compatibilidad de dependencias
- Estrategias de resolución

Seguridad de dependencias

- Auditoría de vulnerabilidades
- Parches de seguridad
- Licencias de dependencias

- Proveedores confiables

Resolución de Defectos e Incidencias

Análisis de causa raíz

- Debugging sistemático
- Análisis de logs y trazas
- Reproducción de problemas
- Identificación de causa fundamental

Corrección de defectos

- Implementación de soluciones
- Testing de correcciones
- Prevención de recurrencia
- Documentación de soluciones

Actualización de pruebas

- Creación de casos de prueba
- Automatización de validaciones
- Cobertura de regresión
- Mantenimiento de suites de pruebas

Optimización de Rendimiento y Caché

Estrategias de caché

- Caché en cliente (localStorage, sessionStorage)
- Caché en servidor (Redis, Memcached)
- Caché HTTP (headers, ETags)
- Invalidación de caché

Monitorización de rendimiento

- Métricas de rendimiento (Core Web Vitals)
- Herramientas de análisis (Lighthouse, WebPageTest)
- Alertas de degradación
- Análisis de tendencias

Optimización del servidor

- Compresión de respuestas
- Optimización de consultas
- Pooling de conexiones
- Escalabilidad horizontal

Habilidades actitudinales

- Implementa i18n usando librerías (i18next, vue-i18n) separando contenido de código
- Localiza contenido considerando formatos de fecha, moneda, plurales y direccionalidad
- Verifica que interfaz se adapta correctamente a diferentes idiomas y regiones

Justificación

Este bloque agrupa la competencia de internacionalización y localización, que es fundamental para aplicaciones globales. La implementación de i18n (C28) separa contenido de código, permitiendo traducción y adaptación a diferentes regiones. Esta competencia es coherente con el objetivo de crear aplicaciones mantenibles y escalables. Los entregables incluyen aplicaciones con soporte multiidioma, contenido localizado y interfaz adaptada a diferentes regiones. La evaluación se realiza mediante validación de traducciones, pruebas en diferentes idiomas y verificación de adaptación cultural.

Ubicación en la progresión

Este bloque representa características avanzadas que mejoran la experiencia de usuario global. Se integra con otros bloques, asegurando que aplicaciones pueden servir a usuarios en múltiples idiomas y regiones. La progresión incluye desde soporte básico de idiomas hasta localización completa con consideraciones culturales. Dominar este bloque es esencial para desarrolladores que trabajan en aplicaciones internacionales.

Bloque 9 - Documentación, Comunicación y Colaboración

Objetivo pedagógico

Desarrollar habilidades transversales de comunicación, documentación y colaboración que facilitan trabajo en equipo y mantenimiento a largo plazo.

Actividades

Actividad 9.1 Documentar componentes, APIs y arquitectura
Actividad 9.2 Colaborar con diseñadores y equipos multidisciplinarios
Actividad 9.3 Comunicar con stakeholders y coordinar tareas de desarrollo

Competencias

Competencias	Indicadores

Saberes asociados

Comunicación Técnica y Feedback

Comunicación con stakeholders

- Sesiones de validación de requisitos
- Documentación de cambios de requisitos
- Presentación de decisiones técnicas
- Recopilación de retroalimentación

Feedback técnico constructivo

- Análisis de código para mejora
- Sugerencias basadas en buenas prácticas
- Comunicación respetuosa y clara
- Referencias a estándares de calidad

Explicación de conceptos técnicos

- Simplificación de conceptos complejos
- Uso de ejemplos y analogías
- Documentación clara y accesible
- Presentaciones técnicas efectivas

Documentación de Código y Arquitectura

Documentación de lógica de negocio

- Comentarios explicativos en código
- Especificaciones técnicas detalladas
- Justificación de decisiones complejas
- Ejemplos de uso y casos de prueba

Documentación de APIs

- Especificación Swagger/OpenAPI
- Descripción de parámetros y respuestas

- Ejemplos de solicitudes y respuestas
- Guías de autenticación y autorización

Documentación de integraciones

- Guías de configuración de servicios externos
- Ejemplos de integración paso a paso
- Troubleshooting y resolución de problemas
- Documentación de credenciales y secretos

Documentación de configuración

- Pasos de instalación y configuración
- Variables de entorno requeridas
- Dependencias y requisitos del sistema
- Guías de onboarding para nuevos desarrolladores

Colaboración y Trabajo en Equipo

Trabajo colaborativo

- Participación en sesiones de diseño
- Contribución a decisiones técnicas
- Apoyo a otros desarrolladores
- Compartición de conocimiento y experiencia

Comunicación efectiva en equipo

- Expresión clara de ideas
- Escucha activa de perspectivas
- Resolución constructiva de desacuerdos
- Documentación de decisiones compartidas

Mentoría y aprendizaje

- Ayuda a desarrolladores junior
- Compartición de mejores prácticas
- Revisión de código educativa
- Facilitación de aprendizaje continuo

Estándares de Calidad y Mejores Prácticas

Estándares de código

- Convenciones de nomenclatura
- Estructura y organización de código
- Principios SOLID
- Patrones de diseño

Mejores prácticas de desarrollo

- Principios de arquitectura limpia
- Separación de responsabilidades
- Reutilización de componentes
- Mantenibilidad a largo plazo

Buenas prácticas de documentación

- Documentación actualizada y precisa
- Ejemplos de código funcionales
- Diagramas y visualizaciones
- Accesibilidad de documentación

Gestión del Conocimiento y Transferencia

Documentación de decisiones

- Registro de decisiones arquitectónicas
- Justificación de alternativas rechazadas
- Contexto histórico de cambios
- Lecciones aprendidas

Transferencia de conocimiento

- Sesiones de capacitación
- Documentación de procesos
- Wikis y bases de conocimiento
- Sesiones de pair programming

Comunicación de cambios

- Notas de versión y changelog
- Comunicación de breaking changes
- Guías de migración
- Anuncios de deprecación

Habilidades actitudinales

- Comunica con stakeholders para validar comprensión de requisitos y obtener retroalimentación
- Proporciona feedback constructivo con sugerencias de mejora y referencias a buenas prácticas
- Documenta lógica de negocio compleja con comentarios y especificaciones técnicas
- Documenta APIs con Swagger/OpenAPI incluyendo parámetros, respuestas y ejemplos
- Documenta integraciones externas con ejemplos de uso y configuración
- Documenta pasos de configuración permitiendo que nuevos desarrolladores se incorporen rápidamente

Justificación

Este bloque transversal agrupa competencias de comunicación y documentación que son fundamentales para el trabajo en equipo. Aunque no está directamente asociado a una función específica, estas competencias son esenciales para que desarrolladores colaboren efectivamente, documenten su trabajo y comuniquen decisiones técnicas. Los entregables incluyen documentación clara, especificaciones técnicas y comunicación efectiva con stakeholders. La evaluación se realiza mediante revisión de documentación, feedback de equipo y validación de comunicación.

Ubicación en la progresión

Este bloque transversal acompaña todo el desarrollo, asegurando que comunicación y documentación son de alta calidad. Se integra con otros bloques, mejorando la colaboración y el mantenimiento a largo plazo. La progresión incluye desde documentación básica hasta comunicación estratégica con stakeholders. Dominar este bloque es esencial para desarrolladores que buscan crecer hacia roles de liderazgo.

Bloque 10 - Planificación, Estimación y Gestión de Proyectos

Objetivo pedagógico

Desarrollar habilidades transversales de planificación y gestión de proyectos que permiten al desarrollador participar en decisiones de alcance y cronograma.

Actividades

Actividad 10.1 Estimar esfuerzo y participar en planificación de sprints
Actividad 10.2 Priorizar tareas y gestionar deuda técnica

Competencias

Competencias	Indicadores

Saberes asociados

Estimación de Esfuerzo y Complejidad Técnica

Técnicas de estimación

- Estimación por puntos de historia
- Estimación por tiempo
- Estimación relativa
- Análisis de complejidad técnica
- Identificación de riesgos técnicos

Factores de complejidad

- Complejidad de requisitos funcionales
- Complejidad de requisitos técnicos
- Dependencias entre tareas
- Restricciones de infraestructura
- Curva de aprendizaje de tecnologías

Validación de estimaciones

- Comparación con proyectos anteriores
- Revisión por pares de estimaciones
- Análisis de desviaciones
- Ajuste de estimaciones basado en experiencia

Planificación y Cronograma de Proyectos

Estructura de desglose de trabajo

- Descomposición de requisitos en tareas
- Identificación de dependencias entre tareas
- Definición de hitos y entregables
- Asignación de recursos a tareas

Gestión de cronograma

- Planificación de sprints
- Estimación de duración de tareas

- Identificación de camino crítico
- Gestión de buffers y contingencias
- Seguimiento de progreso

Metodologías ágiles

- Scrum y sprints
- Kanban y flujo de trabajo
- Planificación iterativa
- Retrospectivas y mejora continua

Gestión de Riesgos Técnicos

Identificación de riesgos

- Riesgos técnicos comunes
- Riesgos de integración
- Riesgos de rendimiento
- Riesgos de seguridad
- Riesgos de disponibilidad de recursos

Análisis y mitigación de riesgos

- Evaluación de probabilidad e impacto
- Estrategias de mitigación
- Planes de contingencia
- Comunicación de riesgos
- Seguimiento de riesgos

Restricciones técnicas

- Limitaciones de infraestructura
- Restricciones de tecnología
- Limitaciones de recursos humanos
- Restricciones de tiempo y presupuesto

Comunicación con Stakeholders y Equipo

Comunicación de estimaciones

- Presentación de estimaciones realistas
- Justificación de estimaciones
- Comunicación de incertidumbre
- Negociación de plazos

Comunicación de riesgos

- Identificación temprana de riesgos
- Comunicación proactiva de problemas
- Escalada de riesgos críticos
- Propuesta de soluciones

Participación en planificación

- Contribución a decisiones de priorización
- Feedback sobre viabilidad técnica
- Colaboración en definición de alcance
- Participación en refinamiento de requisitos

Autonomía e Iniciativa en Gestión de Proyectos

Responsabilidad en estimaciones

- Propiedad de estimaciones proporcionadas
- Seguimiento de compromisos de tiempo
- Comunicación de desviaciones

- Aprendizaje de estimaciones anteriores

Iniciativa en identificación de riesgos

- Identificación proactiva de problemas potenciales
- Propuesta de mitigaciones
- Escalada oportuna de riesgos
- Contribución a mejora de procesos

Compromiso con objetivos del proyecto

- Alineación con cronograma del proyecto
- Cumplimiento de compromisos
- Flexibilidad ante cambios
- Contribución al éxito del equipo

Herramientas y Prácticas de Planificación

Herramientas de gestión de proyectos

- Jira y sistemas de seguimiento
- Tableros Kanban
- Herramientas de planificación ágil
- Herramientas de documentación de requisitos

Métricas y seguimiento

- Velocidad del equipo
- Burndown charts
- Cumplimiento de estimaciones
- Análisis de desviaciones
- Reportes de progreso

Documentación de planificación

- Especificaciones de requisitos
- Documentación de arquitectura
- Planes de proyecto
- Registros de riesgos
- Actas de reuniones

Habilidades actitudinales

- Participa en estimación de esfuerzo considerando complejidad técnica y riesgos
- Comunica riesgos técnicos y restricciones al equipo de forma proactiva
- Participa en planificación de sprints priorizando trabajo según impacto

Justificación

Este bloque transversal agrupa competencias de planificación y gestión que, aunque no están explícitamente detalladas en las competencias del contexto congelado, son fundamentales para el desarrollo profesional. Estas competencias permiten que desarrolladores participen en estimación de esfuerzo, planificación de sprints y gestión de riesgos. Los entregables incluyen estimaciones realistas, planes de proyecto y seguimiento de progreso. La evaluación se realiza mediante análisis de estimaciones, cumplimiento de cronogramas y feedback de equipo.

Ubicación en la progresión

Este bloque transversal acompaña todo el ciclo de desarrollo, permitiendo que desarrolladores participen en decisiones de planificación. Se integra con otros bloques, asegurando que trabajo se ejecuta de forma

ordenada y eficiente. La progresión incluye desde estimación de tareas simples hasta planificación de proyectos complejos. Dominar este bloque es esencial para desarrolladores que buscan roles de liderazgo técnico.

Bloque 11 - Control de Versiones y Gestión de Código

Objetivo pedagógico

Desarrollar habilidades transversales de control de versiones y gestión de código que facilitan colaboración y trazabilidad de cambios.

Actividades

Actividad 11.1 Gestionar control de versiones y estrategias de branching

Competencias

Competencias	Indicadores

Saberes asociados

Fundamentos de Git y Control de Versiones

Conceptos básicos de Git

- Repositorios locales y remotos
- Commits y historial de cambios
- Ramas y merging
- Tags y releases
- Staging area y working directory

Flujos de trabajo con Git

- Git Flow
- GitHub Flow
- Trunk-based development
- Feature branches
- Hotfix branches

Resolución de conflictos

- Merge conflicts
- Rebase conflicts
- Estrategias de resolución
- Herramientas de merge

Estándares de Calidad de Código

Convenciones de nomenclatura

- Nombres de variables y funciones
- Nombres de archivos y directorios
- Convenciones de commits
- Nomenclatura de ramas

Estándares de formato

- Indentación y espaciado
- Longitud de línea
- Organización de imports

- Comentarios y documentación

Principios de código limpio

- DRY (Don't Repeat Yourself)
- KISS (Keep It Simple, Stupid)
- YAGNI (You Aren't Gonna Need It)
- Legibilidad y mantenibilidad

Prácticas de Colaboración en Equipo

Gestión de ramas colaborativas

- Creación de ramas de feature
- Sincronización con rama principal
- Actualización de ramas locales
- Eliminación de ramas obsoletas

Pull requests y revisión de código

- Creación de pull requests
- Descripción de cambios
- Solicitud de revisión
- Respuesta a comentarios de revisión

Comunicación técnica

- Mensajes de commit descriptivos
- Documentación de cambios
- Explicación de decisiones técnicas
- Feedback constructivo

Herramientas de Control de Versiones

Plataformas de alojamiento

- GitHub
- GitLab
- Bitbucket
- Gitea

Herramientas de línea de comandos

- Comandos Git básicos
- Comandos Git avanzados
- Aliases y configuración
- Scripting con Git

Herramientas gráficas

- Clientes Git GUI
- Integración en IDEs
- Visualización de historial
- Herramientas de diff y merge

Organización y Estructura de Código

Estructura de directorios

- Organización por funcionalidad
- Organización por capa
- Separación de concerns
- Escalabilidad de estructura

Gestión de archivos

- Tamaño de archivos

- Cohesión de módulos
- Dependencias entre archivos
- Reutilización de código

Documentación de código

- Comentarios en línea
- Documentación de funciones
- README y guías
- Ejemplos de uso

Mentalidad Profesional y Responsabilidad

Responsabilidad en cambios

- Propiedad del código
- Trazabilidad de cambios
- Justificación de decisiones
- Impacto de cambios

Colaboración y comunicación

- Respeto por el trabajo de otros
- Feedback constructivo
- Apertura a crítica
- Trabajo en equipo

Mejora continua

- Aprendizaje de mejores prácticas
- Adopción de estándares
- Refactorización responsable
- Evolución de procesos

Habilidades actitudinales

- Utiliza control de versiones (Git) para gestionar cambios de código de forma ordenada
- Crea ramas de feature para trabajo aislado y realiza pull requests para revisión
- Escribe mensajes de commit claros y descriptivos facilitando trazabilidad

Justificación

Este bloque transversal agrupa competencias de control de versiones que son fundamentales para trabajo en equipo. Aunque no están explícitamente detalladas en las competencias del contexto congelado, estas prácticas son esenciales para mantener historial de cambios, facilitar colaboración y permitir rollback de cambios problemáticos. Los entregables incluyen repositorios bien organizados, commits claros y ramas de feature bien gestionadas. La evaluación se realiza mediante análisis de historial de commits, calidad de mensajes de commit y gestión de ramas.

Ubicación en la progresión

Este bloque transversal acompaña todo el desarrollo, asegurando que código se gestiona de forma ordenada. Se integra con otros bloques, facilitando colaboración y trazabilidad. La progresión incluye desde uso básico de Git hasta estrategias avanzadas de branching y merging. Dominar este bloque es esencial para desarrolladores que trabajan en equipos.

Bloque 12 - Funcionalidades Avanzadas y Experiencia de Usuario

Objetivo pedagógico

Desarrollar habilidades para mejorar la accesibilidad de interfaces, asegurando que aplicaciones son usables por personas con diferentes capacidades.

Actividades

Actividad 12.1 Implementar comunicación en tiempo real
Actividad 12.2 Implementar internacionalización y localización
Actividad 12.3 Implementar Progressive Web Apps (PWA)

Competencias

Competencias	Indicadores
C30 - Configurar integración continua automatizando pruebas y validaciones en cada commit	Configura CI/CD con GitHub Actions, GitLab CI o Jenkins ejecutando pruebas automáticamente Implementa validaciones (linting, testing, seguridad) en pipeline de CI Genera reportes de cobertura y calidad de código

Saberes asociados

Accesibilidad Web y Estándares WCAG

Estándares WCAG 2.1

- Niveles de conformidad (A, AA, AAA)
- Criterios de éxito WCAG
- Principios POUR (Perceptible, Operable, Comprensible, Robusto)

Evaluación de accesibilidad

- Herramientas de auditoría de accesibilidad
- Testing con lectores de pantalla
- Validación de contraste de color
- Análisis de estructura semántica

Atributos ARIA y Semántica HTML

Implementación de ARIA

- Roles ARIA (role, aria-label, aria-labelledby)
- Propiedades ARIA (aria-hidden, aria-disabled, aria-expanded)
- Estados ARIA (aria-checked, aria-selected, aria-pressed)
- Relaciones ARIA (aria-owns, aria-controls, aria-describedby)

Semántica HTML5

- Elementos semánticos (header, nav, main, article, section, footer)
- Etiquetas de formulario accesibles (label, fieldset, legend)
- Estructura de encabezados (h1-h6) jerárquica

Navegación por Teclado y Gestión del Foco

Implementación de navegación por teclado

- Orden de tabulación (tabindex)
- Gestión del foco (focus, blur, autofocus)
- Atajos de teclado accesibles
- Indicadores visuales de foco

Interactividad sin ratón

- Manejo de eventos de teclado (keydown, keyup, keypress)
- Soporte para navegación con Enter y Espacio
- Escape para cerrar modales y menús
- Navegación con Tab y Shift+Tab

Diseño Inclusivo y Experiencia de Usuario Accesible

Principios de diseño inclusivo

- Diseño para discapacidades visuales
- Diseño para discapacidades auditivas
- Diseño para discapacidades motoras
- Diseño para discapacidades cognitivas

Mejora de experiencia de usuario

- Mensajes de error claros y accesibles
- Confirmaciones de acciones importantes
- Retroalimentación visual y auditiva
- Navegación consistente y predecible

Herramientas y Frameworks de Testing de Accesibilidad

Herramientas de auditoría automática

- axe DevTools
- WAVE (Web Accessibility Evaluation Tool)
- Lighthouse auditoría de accesibilidad
- Pa11y y herramientas CLI

Testing manual y con usuarios

- Testing con lectores de pantalla (NVDA, JAWS, VoiceOver)
- Testing con navegación por teclado
- Testing con usuarios con discapacidades
- Validación de conformidad WCAG

Integración de Accesibilidad en CI/CD

Automatización de pruebas de accesibilidad

- Pruebas de accesibilidad en pipelines CI/CD
- Reportes de accesibilidad automatizados
- Validación de WCAG en cada commit
- Integración con herramientas de testing

Monitorización de accesibilidad

- Seguimiento de métricas de accesibilidad
- Alertas de regresiones de accesibilidad
- Documentación de cambios de accesibilidad
- Auditorías periódicas de accesibilidad

Habilidades actitudinales

- Implementa accesibilidad web (WCAG) cumpliendo estándares de conformidad
- Implementa atributos ARIA para semántica y estado de componentes dinámicos
- Implementa navegación por teclado completa permitiendo acceso sin ratón

Justificación

Este bloque agrupa la competencia de accesibilidad web, que es fundamental para crear aplicaciones inclusivas. La mejora de accesibilidad (C30) asegura que interfaces son usables por personas con discapacidades, cumpliendo estándares WCAG. Esta competencia es coherente con el objetivo de crear experiencias de usuario de calidad. Los entregables incluyen interfaces accesibles, navegación por teclado completa y atributos ARIA implementados. La evaluación se realiza mediante auditorías de accesibilidad, testing con lectores de pantalla y validación de conformidad WCAG.

Ubicación en la progresión

Este bloque representa características avanzadas que mejoran la experiencia de usuario para todos. Se integra con otros bloques de front-end, asegurando que accesibilidad se considera desde el inicio. La progresión incluye desde cumplimiento básico de WCAG hasta diseño inclusivo avanzado. Dominar este bloque es esencial para desarrolladores que buscan crear aplicaciones verdaderamente inclusivas.

ANEXO IV Marco de evaluación

ANEXO IV a. Exenciones de unidades

ANEXO IV b. Reglamento de examen

Pruebas	Unidad	Coef.	Forma	Duración
E1 Análisis y Diseño de Soluciones Web	**U1**	**2**	Prueba escrita única	3h
E2 Desarrollo de Interfaces de Usuario y Componentes Front-End	**U2**	**3**	Prueba práctica única	4h
E3 Desarrollo de Lógica de Negocio y APIs Back-End	**U3**	**3**	Prueba práctica única	4h
E4 Gestión de Datos y Bases de Datos	**U4**	**2**	Prueba escrita única	2h
E5 Testing, Depuración y Aseguramiento de Calidad	**U5**	**2**	Prueba escrita única	3h
E6 Seguridad, Rendimiento y Optimización	**U6**	**2**	Prueba escrita única	2h30
E7 Despliegue, Infraestructura y Operaciones	**U7**	**2**	Prueba escrita única	2h
E8 Mantenimiento, Evolución y Gestión de Incidencias	**U8**	**1**	Prueba escrita única	1h30
E9 Documentación, Comunicación y Colaboración	**U9**	**1**	Prueba escrita única	2h
E10 Planificación, Estimación y Gestión de Proyectos	**U10**	**1**	Prueba escrita única	1h30
E11 Control de Versiones y Gestión de Código	**U11**	**1**	Prueba escrita única	1h30
E12 Funcionalidades Avanzadas y Experiencia de Usuario	**U12**	**1**	Prueba escrita única	2h

*Pruebas optativas: solo se tienen en cuenta los puntos por encima de la media.

ANEXO IV c. Definición de las pruebas

E1 : Análisis y Diseño de Soluciones Web

Finalidades de la prueba

Evaluar la capacidad del desarrollador para analizar requisitos funcionales y técnicos, diseñar arquitecturas escalables y seleccionar tecnologías adecuadas en proyectos web complejos.

Contenido de la prueba

Análisis de requisitos funcionales con criterios de aceptación claros; análisis de requisitos técnicos evaluando restricciones y viabilidad; diseño de arquitectura web considerando patrones de diseño y separación de responsabilidades; evaluación comparativa de tecnologías y frameworks; documentación de decisiones arquitectónicas; comunicación con stakeholders para validación de requisitos.

Competencias evaluadas

C1 - Analizar requisitos funcionales y técnicos de una solución web para identificar restricciones y viabilidad técnica

C2 - Diseñar la arquitectura de una solución web completa considerando patrones de diseño y separación de responsabilidades

C3 - Seleccionar tecnologías y frameworks adecuados para cada capa de la aplicación web basándose en requisitos y restricciones

Criterios de evaluación

- El desarrollador produce especificaciones de requisitos con criterios de aceptación verificables y casos de uso documentados
- El desarrollador evalúa restricciones técnicas y viabilidad de soluciones propuestas
- El desarrollador presenta argumentos técnicos documentados para cada decisión arquitectónica
- El desarrollador crea matrices de comparación de tecnologías con análisis de pros y contras
- El desarrollador realiza sesiones de validación con stakeholders y documenta cambios de requisitos

Forma de evaluación

Vía escolar pública o privada concertada : Prueba escrita única — Análisis de caso de estudio con requisitos complejos; diseño de arquitectura; justificación de decisiones tecnológicas

Aprendizaje : Evaluación continua — Evaluación continua mediante proyectos reales; análisis de requisitos y diseño de arquitectura en contexto laboral

Validación de la experiencia profesional (VAE) : Prueba oral única — Presentación de proyectos anteriores; justificación de decisiones de arquitectura; demostración de experiencia en análisis y diseño

Reglas específicas

- La evaluación debe incluir análisis de al menos dos opciones tecnológicas diferentes
- El candidato debe demostrar comunicación efectiva con stakeholders
- La documentación de arquitectura debe incluir diagramas y justificaciones técnicas

E2 : Desarrollo de Interfaces de Usuario y Componentes Front-End

Finalidades de la prueba

Evaluar la capacidad del desarrollador para crear interfaces responsivas, accesibles y de alto rendimiento, con componentes reutilizables y gestión de estado eficiente.

Contenido de la prueba

Desarrollo de interfaces HTML/CSS responsivas con media queries; implementación de componentes reutilizables con documentación clara; gestión de estado de aplicación compleja; enrutamiento con rutas protegidas y dinámicas; optimización de rendimiento mediante lazy loading y code splitting; cumplimiento de estándares WCAG de accesibilidad; navegación por teclado y atributos ARIA; testing en múltiples dispositivos y navegadores.

Competencias evaluadas

C4 - Desarrollar interfaces de usuario responsivas y accesibles que se adapten a múltiples dispositivos

C5 - Implementar componentes reutilizables y modularizados con documentación clara para facilitar mantenimiento

C6 - Programar lógica de negocio en cliente validando datos y optimizando la experiencia de usuario

C15 - Gestionar estado de aplicación compleja utilizando patrones de gestión de estado y optimizando re-renderizados

C16 - Implementar navegación entre vistas con rutas protegidas y dinámicas gestionando historial del navegador

C17 - Optimizar rendimiento de carga implementando lazy loading, code splitting y medición de métricas

Criterios de evaluación

- El desarrollador implementa interfaces que se adaptan correctamente a múltiples tamaños de pantalla
- El desarrollador crea componentes con interfaz clara, props documentadas y ejemplos de uso
- El desarrollador implementa navegación por teclado completa y atributos ARIA donde sea necesario
- El desarrollador optimiza re-renderizados mediante técnicas de memoización
- El desarrollador configura rutas protegidas y verifica navegación atrás/adelante
- El desarrollador implementa lazy loading y verifica mejora de rendimiento con herramientas

Forma de evaluación

Vía escolar pública o privada concertada : Prueba práctica única — Desarrollo de aplicación front-end con requisitos de responsividad, accesibilidad y rendimiento

Aprendizaje : Evaluación continua — Evaluación continua mediante desarrollo de componentes y interfaces en proyectos reales

Validación de la experiencia profesional (VAE) : Prueba práctica única — Presentación de portfolio de componentes y interfaces desarrollados; demostración de accesibilidad y rendimiento

Reglas específicas

- La aplicación debe funcionar correctamente en al menos tres dispositivos diferentes (móvil, tablet, desktop)
- Debe cumplir con estándares WCAG nivel AA como mínimo
- La cobertura de código debe ser superior al 70%

E3 : Desarrollo de Lógica de Negocio y APIs Back-End

Finalidades de la prueba

Evaluar la capacidad del desarrollador para implementar lógica de negocio robusta en servidor, construir APIs REST bien diseñadas y gestionar integraciones con servicios externos.

Contenido de la prueba

Implementación de lógica de negocio siguiendo principios SOLID; garantía de integridad de datos mediante validaciones y transacciones; diseño de endpoints REST siguiendo convenciones; documentación de APIs con Swagger/OpenAPI; implementación de integración con APIs externas usando autenticación segura; versionado de APIs manteniendo compatibilidad hacia atrás; manejo robusto de errores en integraciones.

Competencias evaluadas

C7 - Programar lógica de negocio en servidor aplicando principios SOLID y garantizando integridad de datos

C8 - Construir APIs REST bien diseñadas con documentación clara y buenas prácticas de versionado

C9 - Integrar servicios externos mediante APIs gestionando autenticación y errores de forma robusta

C27 - Gestionar versionado de APIs comunicando cambios y manteniendo compatibilidad hacia atrás

Criterios de evaluación

- El desarrollador estructura código con responsabilidades bien separadas y bajo acoplamiento
- El desarrollador implementa validaciones en múltiples capas y utiliza transacciones para operaciones críticas
- El desarrollador implementa endpoints que siguen estándares REST y retornan códigos de estado apropiados
- El desarrollador proporciona documentación completa de APIs con ejemplos de uso
- El desarrollador configura autenticación correctamente y maneja errores de integración
- El desarrollador configura versionado de APIs y comunica cambios a clientes

Forma de evaluación

Vía escolar pública o privada concertada : Prueba práctica única — Desarrollo de API REST con lógica de negocio compleja, integraciones externas y documentación

Aprendizaje : Evaluación continua — Evaluación continua mediante desarrollo de APIs en proyectos reales con requisitos de negocio

Validación de la experiencia profesional (VAE) : Prueba oral única — Presentación de APIs desarrolladas; explicación de decisiones de diseño y patrones utilizados

Reglas específicas

- La API debe incluir al menos tres integraciones con servicios externos
- Debe implementarse versionado de API (v1, v2, etc.)
- La documentación debe incluir ejemplos de uso y casos de error

E4 : Gestión de Datos y Bases de Datos

Finalidades de la prueba

Evaluar la capacidad del desarrollador para diseñar esquemas de base de datos, optimizar consultas, implementar operaciones CRUD seguras y sincronizar datos entre capas.

Contenido de la prueba

Implementación de validación de datos en cliente y servidor; configuración de conexiones a bases de datos con ORMs o drivers nativos; escritura de consultas SQL/NoSQL optimizadas; análisis de planes de ejecución; implementación de operaciones CRUD con validaciones de integridad; uso de transacciones para garantizar consistencia; implementación de sincronización de datos cliente-servidor en tiempo real.

Competencias evaluadas

C10 - Validar datos de entrada en múltiples capas previniendo vulnerabilidades de inyección y XSS

C11 - Conectar aplicaciones con bases de datos utilizando ORMs y gestionar pools de conexiones eficientemente

C12 - Diseñar y optimizar consultas a bases de datos utilizando índices y análisis de planes de ejecución

C13 - Desarrollar operaciones CRUD manteniendo integridad referencial y transaccionalidad en bases de datos

C14 - Sincronizar datos entre front-end y back-end mediante WebSockets y mecanismos de caché consistente

Criterios de evaluación

- El desarrollador aplica validación en múltiples capas y previene vulnerabilidades OWASP
- El desarrollador establece conexiones confiables y maneja errores de conexión
- El desarrollador analiza planes de ejecución y crea índices estratégicos
- El desarrollador verifica que operaciones mantienen integridad referencial
- El desarrollador implementa rollback en caso de error y maneja conflictos de concurrencia
- El desarrollador verifica que datos se sincronizan correctamente entre capas

Forma de evaluación

Vía escolar pública o privada concertada : Prueba escrita única — Análisis de esquema de base de datos; optimización de consultas; diseño de operaciones CRUD

Aprendizaje : Evaluación continua — Evaluación continua mediante trabajo con bases de datos en proyectos reales

Validación de la experiencia profesional (VAE) : Prueba oral única — Presentación de esquemas de base de datos diseñados; explicación de optimizaciones realizadas

Reglas específicas

- El candidato debe demostrar optimización de al menos dos consultas complejas
- Debe implementarse sincronización en tiempo real mediante WebSockets
- Las operaciones CRUD deben incluir validaciones de integridad referencial

E5 : Testing, Depuración y Aseguramiento de Calidad

Finalidades de la prueba

Evaluar la capacidad del desarrollador para escribir pruebas automatizadas, ejecutar pruebas funcionales, depurar errores sistemáticamente y revisar código de calidad.

Contenido de la prueba

Escritura de pruebas unitarias con cobertura mínima del 80%; implementación de pruebas de integración validando flujos completos; ejecución de pruebas funcionales en escenarios reales; reporte de defectos con pasos de reproducción y logs; debugging de JavaScript utilizando DevTools; análisis de defectos identificando causa raíz; realización de code review evaluando calidad y seguridad.

Competencias evaluadas

C18 - Escribir pruebas unitarias e integración con cobertura de código adecuada y frameworks de testing

C19 - Ejecutar pruebas funcionales end-to-end y reportar defectos con información detallada para resolución

C20 - Depurar errores en front-end y back-end utilizando herramientas de debugging y análisis de logs

C21 - Corregir incidencias detectadas en pruebas analizando causa raíz y actualizando pruebas automatizadas

C22 - Revisar código de otros desarrolladores proporcionando feedback técnico constructivo y verificando estándares

Criterios de evaluación

- El desarrollador implementa pruebas unitarias con cobertura mínima del 80%
- El desarrollador valida que flujos completos funcionan correctamente
- El desarrollador verifica escenarios reales de uso y documenta resultados
- El desarrollador proporciona información detallada para resolución de defectos
- El desarrollador identifica y resuelve problemas usando herramientas de debugging
- El desarrollador documenta causa y solución de defectos
- El desarrollador proporciona feedback constructivo con referencias a buenas prácticas

Forma de evaluación

Vía escolar pública o privada concertada : Prueba escrita única — Escritura de pruebas unitarias e integración; análisis de cobertura de código; reporte de defectos

Aprendizaje : Evaluación continua — Evaluación continua mediante testing en proyectos reales

Validación de la experiencia profesional (VAE) : Prueba oral única — Presentación de estrategia de testing; demostración de debugging y análisis de defectos

Reglas específicas

- Cobertura de código mínima del 80%
- Debe incluirse testing de casos de error y edge cases
- Los reportes de defectos deben incluir pasos de reproducción y logs

E6 : Seguridad, Rendimiento y Optimización

Finalidades de la prueba

Evaluar la capacidad del desarrollador para implementar sistemas de autenticación y autorización seguros, aplicar medidas de seguridad web, optimizar rendimiento del servidor y gestionar dependencias.

Contenido de la prueba

Implementación de autenticación segura (JWT, OAuth, sesiones); configuración de autorización basada en roles/permisos; configuración de cabeceras de seguridad HTTP; realización de auditorías de seguridad identificando vulnerabilidades OWASP; implementación de estrategias de caché (Redis, Memcached); configuración de monitorización y alertas de rendimiento; actualización de dependencias manteniendo

compatibilidad.

Competencias evaluadas

C23 - Gestionar autenticación y autorización implementando sistemas seguros de control de acceso
C24 - Aplicar medidas de seguridad web implementando cabeceras HTTP y auditorías de vulnerabilidades
C25 - Optimizar rendimiento del servidor implementando caché y monitorización de métricas de rendimiento
C26 - Actualizar dependencias y librerías resolviendo conflictos de versiones y validando compatibilidad

Criterios de evaluación

- El desarrollador configura autenticación correctamente y almacena credenciales de forma segura
- El desarrollador implementa control de acceso y verifica que solo usuarios autorizados acceden
- El desarrollador implementa cabeceras de seguridad y verifica que se envían correctamente
- El desarrollador utiliza herramientas de análisis de seguridad y documenta hallazgos
- El desarrollador configura caché y verifica que mejora rendimiento
- El desarrollador implementa monitorización y responde a alertas
- El desarrollador valida que actualizaciones no introducen vulnerabilidades o regresiones

Forma de evaluación

Vía escolar pública o privada concertada : Prueba escrita única — Análisis de seguridad; diseño de sistema de autenticación; optimización de rendimiento

Aprendizaje : Evaluación continua — Evaluación continua mediante implementación de seguridad y optimización en proyectos reales

Validación de la experiencia profesional (VAE) : Prueba oral única — Presentación de medidas de seguridad implementadas; demostración de optimizaciones de rendimiento

Reglas específicas

- Debe implementarse autenticación con al menos dos métodos diferentes
- La auditoría de seguridad debe cubrir al menos cinco vulnerabilidades OWASP Top 10
- Debe demostrarse mejora de rendimiento mediante caché

E7 : Despliegue, Infraestructura y Operaciones

Finalidades de la prueba

Evaluar la capacidad del desarrollador para configurar entornos de desarrollo, implementar CI/CD, automatizar despliegues y gestionar infraestructura en plataformas cloud.

Contenido de la prueba

Configuración de entorno de desarrollo con Docker y variables de entorno; automatización de configuración mediante scripts; configuración de CI/CD con GitHub Actions, GitLab CI o Jenkins; implementación de validaciones en pipeline de CI; configuración de herramientas de bundling; automatización de build y despliegue; despliegue en plataformas cloud; configuración de infraestructura incluyendo escalabilidad automática.

Competencias evaluadas

C29 - Configurar entornos de desarrollo con herramientas y documentación para facilitar onboarding

C30 - Configurar integración continua automatizando pruebas y validaciones en cada commit

C31 - Automatizar compilaciones y despliegues utilizando herramientas de bundling y estrategias de despliegue

C32 - Desplegar aplicaciones en plataformas cloud configurando infraestructura y escalabilidad

Criterios de evaluación

- El desarrollador documenta pasos de configuración permitiendo onboarding rápido
- El desarrollador facilita configuración de entorno para nuevos desarrolladores
- El desarrollador implementa validaciones automáticas en pipeline de CI
- El desarrollador genera reportes de cobertura y calidad de código
- El desarrollador verifica que build se ejecuta correctamente
- El desarrollador implementa estrategias de despliegue minimizando downtime
- El desarrollador configura infraestructura y verifica que aplicación funciona en producción

Forma de evaluación

Vía escolar pública o privada concertada : Prueba escrita única — Diseño de pipeline CI/CD; configuración de infraestructura; documentación de despliegue

Aprendizaje : Evaluación continua — Evaluación continua mediante configuración de CI/CD y despliegue en proyectos reales

Validación de la experiencia profesional (VAE) : Prueba oral única — Presentación de pipelines CI/CD configurados; demostración de despliegue en cloud

Reglas específicas

- El pipeline de CI/CD debe incluir al menos tres validaciones diferentes
- Debe demostrarse despliegue en al menos una plataforma cloud
- La documentación debe incluir pasos de rollback en caso de error

E8 : Mantenimiento, Evolución y Gestión de Incidencias

Finalidades de la prueba

Evaluar la capacidad del desarrollador para implementar internacionalización y localización, permitiendo que aplicaciones sirvan a usuarios en múltiples idiomas y regiones.

Contenido de la prueba

Implementación de i18n usando librerías (i18next, vue-i18n); separación de contenido de código; localización de contenido considerando formatos de fecha, moneda, plurales y direccionalidad; verificación de adaptación a diferentes idiomas y regiones; testing en múltiples idiomas.

Competencias evaluadas

C28 - Implementar internacionalización y localización adaptando contenido a múltiples idiomas y regiones

Criterios de evaluación

- El desarrollador separa contenido de código permitiendo traducción
- El desarrollador adapta interfaz a diferentes idiomas y regiones
- El desarrollador prueba en múltiples idiomas y verifica adaptación cultural

Forma de evaluación

Vía escolar pública o privada concertada : Prueba escrita única — Implementación de i18n; diseño de estrategia de localización

Aprendizaje : Evaluación continua — Evaluación continua mediante implementación de i18n en proyectos reales

Validación de la experiencia profesional (VAE) : Prueba oral única — Presentación de aplicaciones con soporte multiidioma; demostración de localización

Reglas específicas

- Debe implementarse soporte para al menos tres idiomas diferentes
- Debe incluirse localización de formatos de fecha y moneda
- La interfaz debe adaptarse correctamente a idiomas RTL (derecha a izquierda)

E9 : Documentación, Comunicación y Colaboración

Finalidades de la prueba

Evaluar la capacidad del desarrollador para comunicar efectivamente, documentar su trabajo y colaborar en equipo.

Contenido de la prueba

Comunicación con stakeholders para validación de requisitos; proporcionar feedback constructivo; documentación de lógica de negocio compleja; documentación de APIs con Swagger/OpenAPI; documentación de integraciones externas; documentación de pasos de configuración para onboarding.

Competencias evaluadas

Criterios de evaluación

- El desarrollador realiza sesiones de validación y documenta cambios de requisitos
- El desarrollador comunica feedback de forma respetuosa y clara
- El desarrollador proporciona documentación clara y actualizada
- El desarrollador proporciona documentación completa de APIs
- El desarrollador proporciona guías de configuración y troubleshooting
- El desarrollador facilita onboarding mediante documentación clara

Forma de evaluación

Vía escolar pública o privada concertada : Prueba escrita única — Producción de documentación técnica; comunicación con stakeholders; feedback de código

Aprendizaje : Evaluación continua — Evaluación continua mediante documentación y comunicación en proyectos reales

Validación de la experiencia profesional (VAE) : Prueba oral única — Presentación de documentación producida; demostración de comunicación efectiva

Reglas específicas

- La documentación debe ser clara, completa y actualizada
- Debe incluirse documentación de APIs con ejemplos de uso

- El feedback debe ser constructivo y respetuoso

E10 : Planificación, Estimación y Gestión de Proyectos

Finalidades de la prueba

Evaluar la capacidad del desarrollador para participar en planificación, estimación y gestión de proyectos.

Contenido de la prueba

Participación en estimación de esfuerzo considerando complejidad técnica; comunicación de riesgos técnicos; participación en planificación de sprints; priorización de trabajo según impacto.

Competencias evaluadas

Criterios de evaluación

- El desarrollador proporciona estimaciones realistas basadas en experiencia
- El desarrollador identifica y comunica riesgos tempranamente
- El desarrollador contribuye a decisiones de priorización

Forma de evaluación

Vía escolar pública o privada concertada : Prueba escrita única — Análisis de proyecto; estimación de esfuerzo; identificación de riesgos

Aprendizaje : Evaluación continua — Evaluación continua mediante participación en planificación de proyectos reales

Validación de la experiencia profesional (VAE) : Prueba oral única — Presentación de experiencia en planificación; demostración de estimación de esfuerzo

Reglas específicas

- Las estimaciones deben incluir análisis de riesgos
- Debe demostrarse participación en decisiones de priorización
- La comunicación de riesgos debe ser proactiva y clara

E11 : Control de Versiones y Gestión de Código

Finalidades de la prueba

Evaluar la capacidad del desarrollador para utilizar control de versiones y gestionar código de forma ordenada.

Contenido de la prueba

Uso de control de versiones (Git) para gestionar cambios; creación de ramas de feature; realización de pull requests para revisión; escritura de mensajes de commit claros y descriptivos.

Competencias evaluadas

Criterios de evaluación

- El desarrollador mantiene historial claro de cambios con commits bien documentados
- El desarrollador sigue flujo de branching consistente
- El desarrollador proporciona mensajes de commit informativos

Forma de evaluación

Vía escolar pública o privada concertada : Prueba escrita única — Análisis de historial de commits; diseño de estrategia de branching

Aprendizaje : Evaluación continua — Evaluación continua mediante uso de Git en proyectos reales

Validación de la experiencia profesional (VAE) : Prueba oral única — Presentación de repositorios gestionados; demostración de flujo de branching

Reglas específicas

- Los commits deben tener mensajes descriptivos y claros
- Debe seguirse un flujo de branching consistente (git flow o similar)
- Los pull requests deben incluir descripción clara de cambios

E12 : Funcionalidades Avanzadas y Experiencia de Usuario

Finalidades de la prueba

Evaluar la capacidad del desarrollador para mejorar la accesibilidad de interfaces, asegurando que aplicaciones son usables por personas con diferentes capacidades.

Contenido de la prueba

Implementación de accesibilidad web (WCAG) cumpliendo estándares de conformidad; implementación de atributos ARIA para semántica; implementación de navegación por teclado completa; testing con lectores de pantalla; auditorías de accesibilidad.

Competencias evaluadas

C30 - Configurar integración continua automatizando pruebas y validaciones en cada commit

Criterios de evaluación

- El desarrollador verifica conformidad WCAG en nivel requerido
- El desarrollador incluye atributos ARIA donde sea necesario
- El desarrollador verifica que interfaz es navegable completamente por teclado

Forma de evaluación

Vía escolar pública o privada concertada : Prueba escrita única — Análisis de accesibilidad; diseño de interfaz accesible; auditoría WCAG

Aprendizaje : Evaluación continua — Evaluación continua mediante implementación de accesibilidad en proyectos reales

Validación de la experiencia profesional (VAE) : Prueba oral única — Presentación de interfaces accesibles; demostración de conformidad WCAG

Reglas específicas

- Debe cumplirse conformidad WCAG nivel AA como mínimo
- Debe demostrarse navegación por teclado completa
- Debe incluirse testing con lectores de pantalla

ANEXO IV d. Dispositivo de evaluación detallado por bloque

Este anexo describe, para cada bloque de competencias, el dispositivo de evaluación detallado: indicadores SMART por misión profesional, situaciones de evaluación (simulaciones, estudios de caso), pruebas esperadas (producciones documentales, acciones observables, elementos reflexivos), criterios de éxito agrupados en siete familias, y el umbral de validación del bloque. Este dispositivo constituye la referencia para evaluadores y tribunales.

Bloque 1 - Análisis y Diseño de Soluciones Web

Indicadores de desempeño por misión

Misión	Indicadores SMART
M11 — Analizar requisitos funcionales	Porcentaje de requisitos funcionales documentados con criterios de aceptación verificables (objetivo: 100%) Número de sesiones de validación realizadas con stakeholders antes de iniciar diseño (objetivo: mínimo 2) Completitud de especificaciones técnicas incluyendo casos de uso y restricciones (objetivo: 100%) Tiempo de clarificación de requisitos reducido mediante documentación inicial (objetivo: reducción del 30%)
M12 — Analizar requisitos técnicos	Evaluación documentada de restricciones técnicas identificadas (objetivo: 100% de restricciones evaluadas) Análisis de viabilidad técnica completado antes de diseño (objetivo: 100%) Identificación de riesgos técnicos con plan de mitigación (objetivo: mínimo 3 riesgos evaluados) Estimación de complejidad técnica realizada y validada (objetivo: estimación dentro del 20% de realidad)
M13 — Diseñar la arquitectura de la solución web	Documentación de arquitectura completa incluyendo diagramas y justificación (objetivo: 100%) Número de patrones arquitectónicos identificados y documentados (objetivo: mínimo 3) Validación de arquitectura con equipo técnico completada (objetivo: 100%) Claridad de documentación arquitectónica evaluada por pares (objetivo: puntuación mínima 8/10)
M14 — Definir la estructura de módulos front-end	Estructura de componentes front-end documentada con convenciones claras (objetivo: 100%) Organización de carpetas y módulos definida y comunicada (objetivo: 100%) Patrones de gestión de estado especificados (objetivo: 100%) Documentación de componentes reutilizables completada (objetivo: 100%)

Misión	Indicadores SMART
M15 — Definir la estructura de servicios back-end	Estructura de servicios back-end documentada con responsabilidades claras (objetivo: 100%) Patrones de capas implementados según especificación (objetivo: 100%) Separación de responsabilidades verificada en diseño (objetivo: 100%) Documentación de interfaces de servicios completada (objetivo: 100%)

Situaciones de evaluación

Análisis de Requisitos de Proyecto Web Complejo

Contexto : Se presenta un proyecto web de mediana complejidad (plataforma de e-commerce o sistema de gestión empresarial) con requisitos funcionales y técnicos iniciales. El desarrollador debe analizar estos requisitos, identificar ambigüedades, documentar criterios de aceptación y comunicar con stakeholders simulados para validar comprensión.

Restricciones : Tiempo límite de 4 horas, requisitos iniciales incompletos o ambiguos, necesidad de realizar al menos 2 sesiones de validación con stakeholders, documentación debe incluir casos de uso y criterios de aceptación medibles

Producciones esperadas :

- Documento de especificación funcional con casos de uso detallados
- Documento de análisis de requisitos técnicos con restricciones identificadas
- Matriz de criterios de aceptación para cada requisito funcional
- Registro de sesiones de validación con stakeholders
- Documento de cambios de requisitos identificados durante análisis

Competencias evaluadas : C1

Diseño de Arquitectura Web Escalable

Contexto : Basándose en los requisitos analizados en la situación anterior, el desarrollador debe diseñar una arquitectura web completa considerando patrones de diseño, separación de responsabilidades, escalabilidad y mantenibilidad. Debe documentar decisiones arquitectónicas y justificar selecciones de patrones.

Restricciones : Arquitectura debe soportar crecimiento futuro, debe considerar al menos 3 patrones arquitectónicos diferentes, documentación debe incluir diagramas de componentes y flujos de datos, justificación de decisiones debe ser técnicamente sólida

Producciones esperadas :

- Diagrama de arquitectura general con componentes principales
- Diagrama de flujo de datos entre capas
- Especificación de estructura de front-end con organización de componentes
- Especificación de estructura de back-end con servicios y capas
- Documento de justificación de decisiones arquitectónicas
- Documento de patrones arquitectónicos utilizados con explicación

Competencias evaluadas : C2

Evaluación Comparativa y Selección de Tecnologías

Contexto : El desarrollador debe evaluar múltiples opciones tecnológicas para cada capa de la aplicación (front-end, back-end, base de datos, herramientas de testing) basándose en requisitos específicos del proyecto. Debe crear matrices de comparación, analizar pros y contras, y justificar selecciones finales.

Restricciones : Mínimo 3 opciones evaluadas por cada capa, matriz de comparación debe incluir al menos 5 criterios de evaluación, justificación debe considerar comunidad, soporte, rendimiento y mantenibilidad, presentación de recomendaciones a stakeholders técnicos

Producciones esperadas :

- Matriz de comparación de frameworks front-end con criterios ponderados
- Matriz de comparación de frameworks back-end con criterios ponderados
- Matriz de comparación de bases de datos con criterios ponderados
- Análisis de herramientas de testing y CI/CD
- Documento de recomendaciones tecnológicas con justificación
- Presentación ejecutiva de selecciones tecnológicas

Competencias evaluadas : C3

Validación de Arquitectura con Equipo Técnico

Contexto : El desarrollador presenta la arquitectura diseñada y selecciones tecnológicas a un equipo técnico simulado (compañeros, arquitectos, líderes técnicos). Debe responder preguntas, justificar decisiones, considerar retroalimentación y documentar cambios acordados.

Restricciones : Presentación debe durar 30-45 minutos, debe responder al menos 5 preguntas técnicas desafiantes, debe documentar retroalimentación recibida, debe proponer cambios basados en feedback, debe obtener aprobación de arquitectura

Producciones esperadas :

- Presentación de arquitectura con diagramas y justificación
- Documento de preguntas y respuestas técnicas
- Registro de retroalimentación recibida del equipo
- Documento de cambios propuestos basados en feedback
- Acta de aprobación de arquitectura firmada por stakeholders técnicos

Competencias evaluadas : C1, C2, C3

Especificación Técnica Detallada para Implementación

Contexto : El desarrollador debe crear especificaciones técnicas detalladas que permitan a otros desarrolladores implementar la solución sin ambigüedades. Debe documentar interfaces, contratos de datos, flujos de procesos y criterios de aceptación técnicos.

Restricciones : Especificaciones deben ser lo suficientemente detalladas para implementación sin ambigüedades, debe incluir ejemplos de datos y flujos, debe documentar restricciones técnicas, debe ser validado por pares antes de entrega

Producciones esperadas :

- Especificación de APIs REST con ejemplos de solicitud/respuesta
- Especificación de modelos de datos con relaciones
- Especificación de flujos de procesos críticos
- Documento de restricciones técnicas y consideraciones de rendimiento
- Documento de criterios de aceptación técnicos
- Guía de configuración de entorno de desarrollo

Competencias evaluadas : C1, C2, C3

Pruebas esperadas

■ Pruebas documentales

- Documento de especificación funcional con casos de uso y criterios de aceptación
- Documento de análisis de requisitos técnicos con restricciones identificadas
- Diagramas de arquitectura (componentes, flujos de datos, secuencias)
- Documento de justificación de decisiones arquitectónicas
- Matriz de comparación de tecnologías con análisis de pros y contras
- Documento de especificación de APIs REST con ejemplos
- Especificación de modelos de datos y relaciones
- Documento de patrones arquitectónicos utilizados
- Registro de sesiones de validación con stakeholders
- Acta de aprobación de arquitectura

■ Pruebas de acción (en campo)

- Realización de sesiones de análisis de requisitos con stakeholders
- Creación de diagramas de arquitectura utilizando herramientas de modelado
- Evaluación comparativa de múltiples opciones tecnológicas
- Presentación de propuestas de arquitectura a equipo técnico
- Respuesta a preguntas técnicas desafiantes sobre decisiones de diseño
- Incorporación de retroalimentación en especificaciones
- Validación de arquitectura con pares mediante revisión
- Documentación de cambios de requisitos identificados
- Creación de prototipos o pruebas de concepto para validar decisiones

■ Pruebas reflexivas

- Análisis de trade-offs entre diferentes opciones arquitectónicas
- Reflexión sobre escalabilidad y mantenibilidad de decisiones
- Evaluación de riesgos técnicos identificados durante análisis
- Consideración de impacto de decisiones en ciclo de vida del proyecto
- Análisis de lecciones aprendidas de proyectos anteriores
- Reflexión sobre comunicación con stakeholders y claridad de requisitos
- Evaluación de alineación de arquitectura con objetivos de negocio
- Análisis de viabilidad técnica de soluciones propuestas

Criterios de éxito (7 familias)

Familia	Definición	Ponderación
Conformidad normativa	Especificaciones cumplen con estándares de documentación técnica y marcos de referencia de desarrollo web	Verificación de que documentación incluye todos los elementos requeridos (casos de uso, criterios de aceptación, restricciones técnicas)
Calidad técnica	Arquitectura diseñada es técnicamente sólida, escalable y sigue patrones reconocidos de la industria	Evaluación de arquitectura por pares con puntuación mínima de 8/10 en solidez técnica, escalabilidad y adherencia a patrones

Familia	Definición	Ponderación
Comunicación	Documentación es clara, completa y accesible para diferentes audiencias técnicas y no técnicas	Validación de claridad mediante revisión de pares, feedback de stakeholders sobre comprensión de especificaciones
Análisis y resolución	Análisis de requisitos identifica ambigüedades, restricciones y riesgos técnicos de forma sistemática	Número de requisitos ambiguos clarificados, restricciones técnicas identificadas, riesgos documentados con planes de mitigación
Autonomía e iniciativa	Desarrollador toma decisiones arquitectónicas fundamentadas sin depender constantemente de supervisión	Capacidad de justificar decisiones con argumentos técnicos sólidos, proactividad en identificación de problemas potenciales
Colaboración	Comunicación efectiva con stakeholders técnicos y no técnicos, incorporación de retroalimentación	Número de sesiones de validación realizadas, documentación de cambios basados en feedback, aprobación de arquitectura por stakeholders
Postura profesional	Documentación de decisiones, justificación de opciones consideradas, consideración de mantenibilidad a largo plazo	Completitud de documentación de decisiones, análisis de trade-offs documentado, consideración de impacto futuro en especificaciones

Umbral de validación : Promedio ≥ 10/20

Bloque 2 - Desarrollo de Interfaces de Usuario y Componentes Front-End

Indicadores de desempeño por misión

Misión	Indicadores SMART
M02 — Adaptar la interfaz a distintos dispositivos	Porcentaje de dispositivos en los que la interfaz se adapta correctamente (objetivo: 100%) Número de navegadores testeados y compatibilidad verificada Tiempo de carga en dispositivos móviles (objetivo: < 3 segundos) Puntuación de Lighthouse en dispositivos móviles (objetivo: > 80)
M03 — Aplicar diseño responsive	Número de media queries implementadas correctamente Cobertura de breakpoints (móvil, tablet, desktop) Validación de diseño responsive en herramientas de testing Cumplimiento de especificaciones de diseño adaptativo
M04 — Revisar código de otros desarrolladores	Número de comentarios de code review proporcionados Porcentaje de feedback constructivo vs crítica Tiempo de respuesta a pull requests Adherencia a estándares de código verificada
M05 — Corregir incidencias detectadas en pruebas	Número de defectos corregidos Tiempo promedio de resolución de defectos Porcentaje de defectos con análisis de causa raíz documentado Cobertura de pruebas después de corrección

Situaciones de evaluación

Desarrollo de Interfaz Responsiva para E-Commerce

Contexto : Se requiere desarrollar la interfaz de un sitio de e-commerce que debe funcionar perfectamente en móviles, tablets y desktops. La aplicación debe mostrar catálogo de productos, carrito de compras y proceso de checkout adaptándose a diferentes tamaños de pantalla.

Restricciones : Debe cumplir con estándares WCAG AA, soportar navegadores Chrome, Firefox, Safari y Edge, y alcanzar puntuación Lighthouse > 80 en móvil.

Producciones esperadas :

- Componentes HTML/CSS responsivos con media queries
- Interfaz funcional en al menos 3 tamaños de pantalla diferentes
- Documentación de breakpoints y estrategia responsive
- Reporte de testing cross-browser
- Validación de accesibilidad WCAG

Competencias evaluadas : C4, C5, C17

Implementación de Sistema de Gestión de Estado Complejo

Contexto : Desarrollar una aplicación de gestión de tareas con múltiples filtros, búsqueda, ordenamiento y sincronización en tiempo real. La aplicación debe mantener estado consistente entre cliente y servidor, permitir edición optimista y manejar conflictos de concurrencia.

Restricciones : Usar patrón de gestión de estado (Redux, Zustand o similar), implementar memoización para optimizar re-renderizados, cobertura de pruebas > 80%.

Producciones esperadas :

- Implementación de store de estado con actions y reducers
- Componentes optimizados con memoización
- Sincronización cliente-servidor funcionando
- Suite de pruebas unitarias e integración
- Documentación de flujos de estado

Competencias evaluadas : C6, C15, C16

Optimización de Rendimiento de Aplicación Front-End

Contexto : Una aplicación existente tiene problemas de rendimiento con puntuación Lighthouse de 45 en móvil. Se requiere implementar lazy loading, code splitting, optimización de imágenes y caché para mejorar métricas de Core Web Vitals.

Restricciones : Alcanzar Lighthouse > 80, LCP < 2.5s, FID < 100ms, CLS < 0.1. Mantener funcionalidad existente sin breaking changes.

Producciones esperadas :

- Implementación de lazy loading de componentes e imágenes
- Code splitting por rutas
- Optimización de bundle size
- Reporte de mejora de métricas de rendimiento
- Documentación de cambios de optimización

Competencias evaluadas : C17, C4, C5

Code Review y Feedback Técnico en Pull Request

Contexto : Revisar un pull request de un compañero que implementa un nuevo componente de formulario. El código debe cumplir con estándares de calidad, accesibilidad y rendimiento del proyecto.

Restricciones : Proporcionar feedback constructivo con referencias a buenas prácticas, verificar accesibilidad WCAG, validar rendimiento, revisar pruebas.

Producciones esperadas :

- Comentarios detallados en pull request
- Sugerencias de mejora con ejemplos
- Validación de estándares de código
- Verificación de accesibilidad
- Recomendaciones de optimización

Competencias evaluadas : C22, C4, C5

Corrección de Defectos de Interfaz y Testing

Contexto : Se reportan defectos en la interfaz: componentes no se adaptan correctamente en iPad, navegación por teclado no funciona en modal, y performance es lenta en dispositivos móviles antiguos.

Restricciones : Analizar causa raíz de cada defecto, implementar correcciones, actualizar pruebas automatizadas para prevenir recurrencia.

Producciones esperadas :

- Análisis de causa raíz documentado para cada defecto
- Correcciones implementadas y verificadas
- Pruebas automatizadas agregadas
- Reporte de testing en múltiples dispositivos
- Documentación de soluciones

Competencias evaluadas : C21, C4, C16

Pruebas esperadas

■ Pruebas documentales

- Especificación de componentes con Storybook
- Documentación de arquitectura de front-end
- Guía de estándares de código y convenciones
- Documentación de estrategia responsive
- Especificación de gestión de estado
- Documentación de rutas y navegación
- Reporte de auditoría de accesibilidad WCAG
- Análisis de rendimiento con Lighthouse
- Documentación de optimizaciones implementadas
- Especificación de breakpoints y media queries

■ Pruebas de acción (en campo)

- Implementación de componentes responsivos en HTML/CSS
- Desarrollo de sistema de gestión de estado
- Implementación de lazy loading y code splitting
- Configuración de enrutamiento con rutas protegidas
- Optimización de re-renderizados con memoización
- Implementación de navegación por teclado
- Agregación de atributos ARIA
- Testing en múltiples dispositivos y navegadores
- Code review de pull requests
- Corrección de defectos identificados en pruebas

■ Pruebas reflexivas

- Análisis de decisiones de arquitectura de componentes
- Reflexión sobre trade-offs entre rendimiento y funcionalidad
- Evaluación de estrategias de gestión de estado
- Análisis de impacto de optimizaciones en experiencia de usuario
- Reflexión sobre accesibilidad y diseño inclusivo
- Evaluación de feedback recibido en code review
- Análisis de causa raíz de defectos
- Reflexión sobre mejoras en procesos de testing

• Evaluación de compatibilidad cross-browser
• Análisis de métricas de rendimiento y mejoras

Criterios de éxito (7 familias)

Familia	Definición	Ponderación
Conformidad normativa	La interfaz cumple con estándares WCAG AA incluyendo navegación por teclado, atributos ARIA y semántica HTML correcta	Auditoría de accesibilidad con herramientas automatizadas y testing manual con lectores de pantalla
Calidad técnica	Componentes están bien estructurados, documentados y tienen cobertura de pruebas > 80%	Análisis de cobertura de código, revisión de documentación de componentes, validación de pruebas unitarias
Comunicación	Documentación de componentes es clara con ejemplos de uso y especificación de props	Revisión de documentación en Storybook, claridad de ejemplos, completitud de especificaciones
Análisis y resolución	Defectos se analizan identificando causa raíz y se implementan soluciones que previenen recurrencia	Documentación de análisis de causa raíz, validación de correcciones, cobertura de pruebas para defectos
Autonomía e iniciativa	Desarrollador identifica oportunidades de optimización y mejora sin esperar instrucciones	Iniciativas de optimización documentadas, mejoras de rendimiento implementadas, propuestas de refactoring
Colaboración	Feedback en code review es constructivo, específico y ayuda a mejorar calidad del código	Calidad de comentarios en pull requests, tono respetuoso, referencias a buenas prácticas
Postura profesional	Desarrollador mantiene estándares de calidad, busca mejora continua y se adapta a cambios de requisitos	Consistencia en aplicación de estándares, participación en mejora de procesos, adaptabilidad a cambios

Umbral de validación : Promedio ≥ 10/20

Bloque 3 - Desarrollo de Lógica de Negocio y APIs Back-End

Indicadores de desempeño por misión

Misión	Indicadores SMART
M01 — Gestionar autenticación y autorización	Porcentaje de endpoints con autenticación implementada correctamente Número de vulnerabilidades de seguridad identificadas en auditoría Tiempo de respuesta de validación de credenciales Cobertura de pruebas de autorización
M09 — Gestionar el versionado y la evolución de APIs	Número de versiones de API soportadas simultáneamente Porcentaje de clientes migrados a nueva versión Tiempo de deprecación de versiones antiguas Claridad de documentación de cambios de API
M06 — Reducir el tiempo de respuesta del servidor	Tiempo de respuesta promedio del servidor antes y después de optimización Porcentaje de reducción en uso de CPU/memoria Tasa de hit de caché Latencia de respuesta en percentil 95
M07 — Aplicar medidas de seguridad web	Número de vulnerabilidades OWASP Top 10 identificadas Porcentaje de endpoints con cabeceras de seguridad configuradas Resultado de auditoría de seguridad Tiempo de respuesta a incidentes de seguridad

Situaciones de evaluación

Implementación de API REST Completa con Autenticación

Contexto : Se requiere desarrollar una API REST para un sistema de gestión de tareas que permita a usuarios crear, leer, actualizar y eliminar tareas. La API debe incluir autenticación mediante JWT, validación de datos, manejo de errores y documentación completa con Swagger.

Restricciones : La API debe seguir convenciones REST, implementar validación en múltiples capas, incluir códigos de estado HTTP apropiados, y estar completamente documentada. Debe soportar múltiples usuarios con autorización basada en roles.

Producciones esperadas :

- Endpoints REST implementados (GET, POST, PUT, DELETE) con códigos de estado correctos
- Sistema de autenticación JWT funcional
- Validación de datos de entrada en servidor
- Documentación Swagger/OpenAPI completa
- Manejo de errores con mensajes claros
- Pruebas unitarias e integración con cobertura mínima 80%

Competencias evaluadas : C7, C8, C23

Integración con Servicio Externo y Manejo de Errores

Contexto : Se necesita integrar una API externa de pagos (como Stripe o PayPal) en una aplicación de e-commerce. La integración debe manejar autenticación segura, procesar pagos, sincronizar estado de transacciones y recuperarse de fallos de forma robusta.

Restricciones : Debe implementarse autenticación segura con API keys, manejo de reintentos, logging detallado de transacciones, y validación de respuestas. Debe garantizarse que no se pierdan transacciones en caso de fallos.

Producciones esperadas :

- Integración funcional con API de pagos
- Autenticación segura implementada
- Manejo de errores y reintentos
- Logging detallado de transacciones
- Sincronización de estado entre sistemas
- Pruebas de integración con casos de error

Competencias evaluadas : C9, C7

Versionado de API y Migración de Clientes

Contexto : Una API existente necesita evolucionar para soportar nuevas funcionalidades. Se requiere implementar versionado que permita que clientes antiguos sigan funcionando mientras se migran gradualmente a la nueva versión. Debe documentarse claramente el proceso de migración.

Restricciones : Debe mantenerse compatibilidad hacia atrás, implementarse versionado en URL o headers, documentarse cambios claramente, y proporcionarse período de transición. Debe verificarse que clientes antiguos siguen funcionando.

Producciones esperadas :

- Múltiples versiones de API funcionando simultáneamente
- Documentación clara de cambios entre versiones
- Guía de migración para clientes
- Deprecación clara de versiones antiguas
- Pruebas de compatibilidad hacia atrás
- Changelog detallado

Competencias evaluadas : C27, C8

Optimización de Rendimiento del Servidor

Contexto : Un servidor backend está experimentando problemas de rendimiento con tiempos de respuesta lentos y alto uso de CPU. Se requiere identificar cuellos de botella, implementar caché, optimizar consultas y monitorizar mejoras.

Restricciones : Debe analizarse rendimiento actual, implementarse estrategias de caché (Redis/Memcached), optimizarse código crítico, y verificarse mejoras. Debe configurarse monitorización para detectar regresiones.

Producciones esperadas :

- Análisis de rendimiento inicial documentado
- Implementación de caché estratégico
- Optimización de código crítico
- Reducción medible de tiempos de respuesta
- Configuración de monitorización
- Reporte de mejoras de rendimiento

Competencias evaluadas : C25, C7

Auditoría de Seguridad y Remediación de Vulnerabilidades

Contexto : Se realiza una auditoría de seguridad en una aplicación backend y se identifican vulnerabilidades OWASP Top 10. Se requiere implementar medidas de seguridad, configurar cabeceras HTTP, validar entrada y documentar cambios.

Restricciones : Debe implementarse validación en múltiples capas, configurarse cabeceras de seguridad (CSP, X-Frame-Options, HSTS), prevenirse inyección SQL y XSS, y verificarse remediación. Debe documentarse cada vulnerabilidad y su solución.

Producciones esperadas :

- Validación de entrada implementada
- Cabeceras de seguridad HTTP configuradas
- Prevención de inyección SQL y XSS
- Gestión segura de credenciales
- Documentación de vulnerabilidades y soluciones
- Auditoría de seguridad post-remediación

Competencias evaluadas : C24, C7, C10

Pruebas esperadas

■ Pruebas documentales

- Documentación Swagger/OpenAPI de endpoints implementados
- Especificación técnica de arquitectura de backend
- Documentación de versionado de API
- Guías de integración con servicios externos
- Documentación de estrategias de caché implementadas
- Reporte de auditoría de seguridad
- Changelog de API con cambios documentados
- Especificación de esquemas de datos
- Documentación de manejo de errores
- Guías de configuración de autenticación

■ Pruebas de acción (en campo)

- Implementación de endpoints REST con validación
- Configuración de autenticación JWT
- Integración con APIs externas
- Implementación de caché con Redis/Memcached
- Configuración de cabeceras de seguridad HTTP
- Optimización de consultas a base de datos
- Implementación de versionado de API
- Configuración de monitorización de rendimiento
- Implementación de reintentos y manejo de errores
- Configuración de logging detallado

■ Pruebas reflexivas

- Análisis de decisiones de diseño arquitectónico
- Evaluación de trade-offs entre rendimiento y mantenibilidad
- Reflexión sobre patrones SOLID aplicados
- Análisis de impacto de cambios de API en clientes
- Evaluación de estrategias de seguridad implementadas

- Reflexión sobre lecciones aprendidas en integraciones
- Análisis de mejoras de rendimiento logradas
- Evaluación de cobertura de pruebas
- Reflexión sobre manejo de errores y recuperación
- Análisis de feedback de usuarios sobre API

Criterios de éxito (7 familias)

Familia	Definición	Ponderación
Conformidad normativa	Cumplimiento de estándares REST en diseño de endpoints	100% de endpoints siguen convenciones REST (verbos HTTP correctos, códigos de estado apropiados)
Calidad técnica	Código backend implementa principios SOLID con bajo acoplamiento	Análisis de código verifica responsabilidad única, abierto/cerrado, y separación de responsabilidades
Calidad técnica	Integridad de datos garantizada mediante validaciones y transacciones	Cobertura de pruebas de integridad ≥ 80%, sin pérdida de datos en escenarios de error
Comunicación	Documentación de API completa y clara con ejemplos de uso	Documentación Swagger/OpenAPI incluye todos los endpoints, parámetros, respuestas y ejemplos
Análisis y resolución	Identificación y resolución de problemas de rendimiento	Reducción medible de tiempos de respuesta (≥ 20%), documentación de análisis realizado
Autonomía e iniciativa	Implementación de mejoras de seguridad proactivas	Configuración de cabeceras de seguridad, validación en múltiples capas, auditoría de vulnerabilidades
Colaboración	Integración robusta con servicios externos con manejo de errores	Integraciones funcionan correctamente, reintentos implementados, logging detallado de transacciones
Postura profesional	Versionado de API comunicado claramente con período de transición	Múltiples versiones soportadas, documentación de migración clara, clientes antiguos siguen funcionando

Umbral de validación : Promedio ≥ 10/20

Bloque 4 - Gestión de Datos y Bases de Datos

Indicadores de desempeño por misión

Misión	Indicadores SMART
M01 — Gestionar autenticación y autorización	Porcentaje de endpoints protegidos con autenticación y autorización implementadas correctamente Número de vulnerabilidades de control de acceso identificadas y corregidas en auditorías Tiempo de respuesta de validación de permisos en operaciones de datos
M06 — Reducir el tiempo de respuesta del servidor	Reducción porcentual del tiempo de respuesta del servidor tras implementación de caché Tasa de acierto de caché (hit rate) en operaciones de lectura frecuentes Latencia promedio de consultas a base de datos antes y después de optimización
M07 — Aplicar medidas de seguridad web	Número de vulnerabilidades OWASP Top 10 relacionadas con datos identificadas y corregidas Porcentaje de consultas protegidas contra inyección SQL Cobertura de validación de entrada en todas las capas de la aplicación
M08 — Actualizar dependencias y librerías	Número de dependencias de base de datos actualizadas sin introducir breaking changes Tiempo de resolución de conflictos de versiones en librerías de datos Porcentaje de pruebas que pasan después de actualización de dependencias

Situaciones de evaluación

Diseño e Implementación de Esquema de Base de Datos

Contexto : Se requiere diseñar e implementar un esquema de base de datos para un sistema de gestión de pedidos en una plataforma de comercio electrónico. El sistema debe almacenar información de clientes, productos, pedidos, líneas de pedido e historial de transacciones. Se requiere garantizar integridad referencial, rendimiento en consultas frecuentes y escalabilidad para millones de registros.

Restricciones : Debe utilizarse una base de datos relacional (PostgreSQL o MySQL). El esquema debe soportar al menos 1 millón de registros. Las consultas más frecuentes deben ejecutarse en menos de 100ms. Se requiere auditoría de cambios en datos sensibles.

Producciones esperadas :

- Diagrama entidad-relación (ER) documentado
- Script SQL de creación de tablas con restricciones de integridad
- Índices estratégicos con justificación de su necesidad
- Documentación de relaciones y cardinalidades
- Plan de optimización de consultas frecuentes

Competencias evaluadas : C11, C12, C13

Implementación de Operaciones CRUD Seguras y Transaccionales

Contexto : Se requiere implementar operaciones CRUD para un módulo de gestión de inventario donde múltiples usuarios pueden actualizar stock simultáneamente. Las operaciones deben garantizar que el stock nunca sea negativo, mantener historial de cambios y prevenir condiciones de carrera. Se utiliza Node.js con TypeORM y PostgreSQL.

Restricciones : Debe implementarse validación en cliente y servidor. Las transacciones deben ser ACID. Se requiere manejo de errores de concurrencia. El código debe incluir pruebas unitarias e integración con cobertura mínima del 80%.

Producciones esperadas :

- Código de operaciones CRUD con validaciones en múltiples capas
- Implementación de transacciones con rollback en caso de error
- Manejo de conflictos de concurrencia (optimista o pesimista)
- Suite de pruebas unitarias e integración
- Documentación de flujos transaccionales

Competencias evaluadas : C10, C11, C13, C14

Optimización de Consultas y Análisis de Rendimiento

Contexto : Una aplicación de análisis de datos tiene consultas que tardan más de 5 segundos en ejecutarse. Se requiere analizar planes de ejecución, crear índices estratégicos y reescribir consultas para mejorar rendimiento. La base de datos contiene 10 millones de registros en tablas principales.

Restricciones : Debe utilizarse EXPLAIN para analizar planes de ejecución. Se requiere reducir tiempo de ejecución a menos de 500ms. No se pueden modificar estructuras de datos existentes. Se debe documentar cada cambio y su impacto.

Producciones esperadas :

- Análisis de planes de ejecución con EXPLAIN
- Identificación de índices faltantes o ineficientes
- Consultas reescritas optimizadas
- Índices creados con justificación técnica
- Comparativa de rendimiento antes/después con métricas

Competencias evaluadas : C12, C25

Sincronización de Datos en Tiempo Real con WebSockets

Contexto : Se requiere implementar sincronización de datos en tiempo real para una aplicación colaborativa de edición de documentos. Múltiples usuarios pueden editar simultáneamente y los cambios deben reflejarse en todos los clientes en menos de 100ms. Se utiliza Node.js con Socket.io y MongoDB.

Restricciones : Debe implementarse WebSocket para comunicación bidireccional. Se requiere caché con Redis para optimizar rendimiento. Debe manejarse desconexión y reconexión de clientes. Se requiere validación de cambios en servidor.

Producciones esperadas :

- Implementación de servidor WebSocket con Socket.io
- Sincronización de estado entre cliente y servidor
- Implementación de caché con Redis
- Manejo de conflictos de edición simultánea
- Pruebas de sincronización bajo carga

Competencias evaluadas : C14, C25

Validación de Datos y Prevención de Vulnerabilidades

Contexto : Se requiere realizar auditoría de seguridad en una aplicación web existente enfocándose en validación de datos y prevención de inyección SQL y XSS. Se han identificado múltiples puntos donde entrada de usuario no se valida correctamente.

Restricciones : Debe implementarse validación en cliente y servidor. Se requiere parametrización de todas las consultas SQL. Se debe prevenir XSS mediante sanitización. Se requiere documentación de cambios de seguridad.

Producciones esperadas :

- Identificación de puntos vulnerables de validación
- Implementación de validación en múltiples capas
- Reescritura de consultas con parametrización
- Sanitización de entrada de usuario
- Reporte de auditoría de seguridad con hallazgos y correcciones

Competencias evaluadas : C10, C24

Pruebas esperadas

■ Pruebas documentales

- Diagrama entidad-relación (ER) con documentación de relaciones
- Scripts SQL de creación de esquema con restricciones de integridad
- Documentación de índices con análisis de impacto en rendimiento
- Planes de ejecución (EXPLAIN) de consultas optimizadas
- Especificación de operaciones CRUD con validaciones
- Documentación de transacciones y manejo de errores
- Reporte de auditoría de seguridad en gestión de datos
- Documentación de sincronización de datos en tiempo real
- Especificación de caché y estrategias de invalidación

■ Pruebas de acción (en campo)

- Crear tablas con restricciones de integridad referencial
- Implementar índices estratégicos en columnas frecuentemente consultadas
- Escribir consultas SQL/NoSQL optimizadas
- Implementar operaciones CRUD con validación en múltiples capas
- Configurar transacciones ACID con rollback en caso de error
- Implementar WebSocket para sincronización en tiempo real
- Configurar caché con Redis o Memcached
- Parametrizar consultas para prevenir inyección SQL
- Sanitizar entrada de usuario para prevenir XSS
- Crear índices y analizar planes de ejecución con EXPLAIN
- Implementar manejo de conflictos de concurrencia
- Configurar pools de conexiones a base de datos

■ Pruebas reflexivas

- Reflexión sobre decisiones de diseño de esquema y su impacto en rendimiento
- Análisis de trade-offs entre normalización y desnormalización
- Evaluación de estrategias de caché y su efectividad
- Reflexión sobre seguridad en validación de datos
- Análisis de impacto de índices en operaciones de escritura

• Evaluación de estrategias de sincronización de datos
• Reflexión sobre manejo de errores en operaciones transaccionales
• Análisis de escalabilidad de esquema para crecimiento futuro

Criterios de éxito (7 familias)

Familia	Definición	Ponderación
Conformidad normativa	Cumplimiento de estándares de integridad de datos (ACID) en operaciones transaccionales	100% de operaciones críticas implementadas con transacciones ACID verificadas en pruebas
Calidad técnica	Optimización de consultas a base de datos con tiempo de respuesta dentro de especificación	Todas las consultas frecuentes ejecutadas en menos de 100ms según análisis de EXPLAIN
Calidad técnica	Implementación correcta de integridad referencial mediante restricciones de base de datos	100% de relaciones entre tablas implementadas con claves foráneas y restricciones
Comunicación	Documentación clara de esquema de base de datos y operaciones CRUD	Documentación completa con diagramas ER, especificaciones de operaciones y ejemplos de uso
Análisis y resolución	Análisis sistemático de planes de ejecución para identificar y resolver cuellos de botella	Reducción de tiempo de ejecución de consultas en al menos 50% tras optimización
Autonomía e iniciativa	Identificación proactiva de vulnerabilidades de seguridad en gestión de datos	Detección y corrección de vulnerabilidades OWASP relacionadas con datos sin supervisión
Colaboración	Implementación de sincronización de datos consistente en entornos multiusuario	Sincronización de cambios en menos de 100ms en todos los clientes conectados

Umbral de validación : Promedio ≥ 10/20

Bloque 5 - Testing, Depuración y Aseguramiento de Calidad

Indicadores de desempeño por misión

Misión	Indicadores SMART
M02 — Adaptar la interfaz a distintos dispositivos	Porcentaje de dispositivos y navegadores donde la aplicación funciona correctamente (objetivo: 100%) Número de defectos relacionados con compatibilidad detectados en testing (objetivo: 0) Tiempo promedio de ejecución de pruebas de compatibilidad (objetivo: < 30 minutos) Cobertura de combinaciones dispositivo-navegador probadas (objetivo: >= 95%)
M04 — Revisar código de otros desarrolladores	Número de comentarios de code review por pull request (objetivo: >= 2) Tiempo promedio de resolución de comentarios de review (objetivo: < 24 horas) Porcentaje de defectos prevenidos por code review (objetivo: >= 30%) Satisfacción del equipo con feedback de review (objetivo: >= 4/5)
M05 — Corregir incidencias detectadas en pruebas	Tiempo promedio de resolución de defectos (objetivo: < 2 días) Porcentaje de defectos resueltos en primer intento (objetivo: >= 85%) Número de defectos reabiertos (objetivo: 0) Cobertura de pruebas para defectos corregidos (objetivo: 100%)

Situaciones de evaluación

Escritura de suite de pruebas unitarias para módulo de validación

Contexto : El desarrollador debe escribir pruebas unitarias para un módulo de validación de datos de usuario (email, contraseña, teléfono) que se utiliza en múltiples partes de la aplicación. El módulo contiene 5 funciones principales con diferentes casos de borde.

Restricciones : Debe alcanzar mínimo 80% de cobertura de código. Las pruebas deben ejecutarse en menos de 5 segundos. Debe usar Jest o Vitest. Debe incluir mocking de dependencias externas.

Producciones esperadas :

- Suite de pruebas unitarias con casos de éxito y error
- Reporte de cobertura de código
- Documentación de casos de prueba
- Configuración de Jest/Vitest

Competencias evaluadas : C18

Ejecución de pruebas funcionales end-to-end y reporte de defectos

Contexto : El desarrollador debe ejecutar un flujo completo de usuario en una aplicación de comercio electrónico: búsqueda de producto, adición al carrito, checkout y confirmación de pedido. Debe probar en al menos 3 navegadores diferentes (Chrome, Firefox, Safari) y 2 dispositivos (desktop, móvil).

Restricciones : Debe documentar cada paso con capturas de pantalla. Debe reportar cualquier defecto encontrado con pasos de reproducción claros. Debe completar pruebas en máximo 2 horas. Debe usar herramientas como Cypress o Playwright.

Producciones esperadas :

- Reporte de ejecución de pruebas
- Capturas de pantalla de cada paso
- Reportes de defectos con pasos de reproducción
- Matriz de compatibilidad navegador-dispositivo

Competencias evaluadas : C19, C4

Debugging y análisis de causa raíz de error en aplicación

Contexto : Se reporta que usuarios en ciertos navegadores experimentan error 'Cannot read property of undefined' al cargar la página de perfil. El error ocurre intermitentemente y solo afecta a usuarios con conexiones lentas. El desarrollador debe identificar y documentar la causa raíz.

Restricciones : Debe usar DevTools del navegador y herramientas de debugging. Debe simular conexiones lentas. Debe reproducir el error de forma consistente. Debe documentar hallazgos con evidencia (screenshots, logs). Debe proponer solución.

Producciones esperadas :

- Análisis detallado del error
- Logs y stack traces
- Documentación de causa raíz
- Propuesta de solución
- Plan de validación

Competencias evaluadas : C20, C21

Code review de pull request con feedback constructivo

Contexto : El desarrollador debe revisar un pull request de 300 líneas que implementa nueva funcionalidad de autenticación. El código incluye validaciones, manejo de errores, y llamadas a API externa. Debe evaluar calidad, seguridad, adherencia a estándares y sugerir mejoras.

Restricciones : Debe proporcionar mínimo 5 comentarios constructivos. Debe referenciar estándares de proyecto y buenas prácticas. Debe ser respetuoso y educativo. Debe verificar cobertura de pruebas. Debe revisar en máximo 1 hora.

Producciones esperadas :

- Comentarios de review en pull request
- Sugerencias de mejora con referencias
- Evaluación de seguridad
- Evaluación de cobertura de pruebas
- Aprobación o solicitud de cambios

Competencias evaluadas : C22

Corrección de defecto y actualización de pruebas automatizadas

Contexto : Se detectó defecto en módulo de cálculo de descuentos: cuando se aplican múltiples cupones, el cálculo es incorrecto. El desarrollador debe: 1) Reproducir el defecto, 2) Identificar causa raíz, 3) Implementar corrección, 4) Actualizar pruebas existentes, 5) Agregar nuevas pruebas para prevenir recurrencia.

Restricciones : Debe proporcionar pasos de reproducción claros. Debe documentar análisis de causa raíz. Debe implementar solución que no introduce regresiones. Debe alcanzar 100% de cobertura para código modificado. Debe validar corrección en múltiples escenarios.

Producciones esperadas :

- Documentación de defecto y reproducción
- Análisis de causa raíz
- Código corregido
- Pruebas unitarias actualizadas
- Nuevas pruebas para casos de borde
- Validación de corrección

Competencias evaluadas : C21, C18

Pruebas esperadas

■ Pruebas documentales

- Reportes de cobertura de código (Istanbul, nyc)
- Reportes de ejecución de pruebas (JUnit, HTML)
- Documentación de casos de prueba
- Reportes de defectos con pasos de reproducción
- Logs de debugging y stack traces
- Comentarios de code review en pull requests
- Documentación de análisis de causa raíz
- Especificaciones de pruebas actualizadas

■ Pruebas de acción (en campo)

- Escribir y ejecutar pruebas unitarias con Jest/Vitest
- Escribir y ejecutar pruebas de integración con Supertest
- Ejecutar pruebas funcionales con Cypress/Playwright
- Usar DevTools para debugging de JavaScript
- Analizar logs de servidor para identificar errores
- Reproducir defectos de forma sistemática
- Implementar correcciones de código
- Realizar code review de pull requests
- Actualizar pruebas automatizadas
- Validar correcciones en múltiples entornos

■ Pruebas reflexivas

- Reflexión sobre cobertura de pruebas y casos de borde no cubiertos
- Análisis de patrones de defectos recurrentes
- Evaluación de efectividad de estrategia de testing
- Reflexión sobre calidad de feedback en code review
- Análisis de lecciones aprendidas de defectos
- Evaluación de técnicas de debugging utilizadas

• Reflexión sobre mejoras en proceso de testing
• Análisis de impacto de correcciones en sistema

Criterios de éxito (7 familias)

Familia	Definición	Ponderación
Conformidad normativa	Las pruebas cumplen con estándares de calidad de código del proyecto (linting, formato, convenciones)	100% de pruebas pasan validación de linting y formato
Calidad técnica	Las pruebas tienen cobertura mínima de 80% y validan lógica crítica de forma exhaustiva	Cobertura de código >= 80%, mínimo 3 casos de prueba por función
Comunicación	Los reportes de defectos incluyen información clara y suficiente para reproducción y resolución	Cada reporte incluye pasos de reproducción, logs, y evidencia visual
Análisis y resolución	El análisis de causa raíz identifica la raíz del problema, no solo síntomas superficiales	Documentación de análisis incluye cadena causal completa y validación de solución
Autonomía e iniciativa	El desarrollador identifica proactivamente casos de prueba adicionales y mejoras de cobertura	Propone mínimo 2 casos de prueba adicionales por módulo probado
Colaboración	El feedback de code review es constructivo, respetuoso y facilita aprendizaje del equipo	Feedback incluye sugerencias de mejora con referencias a buenas prácticas
Postura profesional	El desarrollador mantiene rigor en testing y calidad incluso bajo presión de cronograma	No se omiten pruebas críticas, se documenta completamente cada defecto

Umbral de validación : Promedio >= 10/20

Bloque 6 - Seguridad, Rendimiento y Optimización

Indicadores de desempeño por misión

Misión	Indicadores SMART
M01 — Gestionar autenticación y autorización	Porcentaje de endpoints protegidos con autenticación válida (objetivo: 100%) Tiempo de respuesta de validación de autorización (objetivo: <50ms) Número de intentos de acceso no autorizado detectados y bloqueados Cobertura de pruebas de autenticación y autorización (objetivo: ≥85%)
M06 — Reducir el tiempo de respuesta del servidor	Reducción de latencia de respuesta del servidor (objetivo: -30% respecto a baseline) Tasa de acierto de caché (objetivo: ≥70%) Tiempo de respuesta promedio (objetivo: <200ms) Carga de CPU del servidor (objetivo: <70%)
M07 — Aplicar medidas de seguridad web	Número de vulnerabilidades OWASP Top 10 identificadas y remediadas Conformidad con cabeceras de seguridad HTTP (objetivo: 100%) Puntuación de auditoría de seguridad (objetivo: ≥90/100) Tiempo de respuesta a incidentes de seguridad (objetivo: <4 horas)
M08 — Actualizar dependencias y librerías	Número de dependencias actualizadas sin breaking changes Tiempo entre actualización disponible y aplicación (objetivo: <30 días) Vulnerabilidades de dependencias resueltas (objetivo: 100%) Tasa de éxito de actualizaciones automatizadas (objetivo: ≥90%)

Situaciones de evaluación

Implementación de Sistema de Autenticación y Autorización

Contexto : Se requiere implementar un sistema de autenticación seguro para una aplicación web que gestiona datos sensibles de usuarios. La aplicación debe permitir login con JWT, implementar refresh tokens, y controlar acceso basado en roles (admin, usuario, invitado).

Restricciones : Debe cumplir con estándares OWASP, almacenar contraseñas hasheadas, implementar expiración de tokens, y validar autorización en cada endpoint sensible.

Producciones esperadas :

- Endpoints de autenticación (login, logout, refresh) implementados y documentados
- Middleware de autorización que valida roles y permisos
- Pruebas unitarias de autenticación con cobertura ≥85%
- Documentación de flujo de autenticación con diagramas
- Configuración segura de variables de entorno

Competencias evaluadas : C23

Auditoría de Seguridad y Remediación de Vulnerabilidades

Contexto : Se realiza una auditoría de seguridad en una aplicación web existente utilizando herramientas de análisis automático (OWASP ZAP, Snyk) y pruebas manuales. Se identifican vulnerabilidades en validación de entrada, cabeceras HTTP y gestión de sesiones.

Restricciones : Debe identificar vulnerabilidades OWASP Top 10, documentar hallazgos con severidad, proporcionar recomendaciones de remediación, y validar que correcciones resuelven problemas sin introducir regresiones.

Producciones esperadas :

- Reporte de auditoría de seguridad con vulnerabilidades identificadas
- Plan de remediación con prioridades
- Código corregido con cabeceras de seguridad HTTP implementadas
- Pruebas de validación de correcciones
- Documentación de cambios de seguridad

Competencias evaluadas : C24

Optimización de Rendimiento del Servidor con Caché

Contexto : Una aplicación web tiene problemas de rendimiento con latencia alta en endpoints que consultan base de datos frecuentemente. Se requiere implementar estrategia de caché en Redis para reducir carga de base de datos y mejorar tiempo de respuesta.

Restricciones : Debe implementar caché con invalidación correcta, monitorizar métricas de rendimiento (latencia, CPU, memoria), validar que caché mejora rendimiento sin comprometer consistencia de datos.

Producciones esperadas :

- Implementación de caché en Redis con estrategia de invalidación
- Monitorización de métricas de rendimiento (Prometheus/Grafana)
- Pruebas de carga que demuestran mejora de rendimiento
- Documentación de estrategia de caché
- Alertas configuradas para anomalías de rendimiento

Competencias evaluadas : C25

Actualización de Dependencias y Resolución de Conflictos

Contexto : Se requiere actualizar dependencias de una aplicación Node.js que tiene vulnerabilidades conocidas en librerías. Algunas dependencias tienen conflictos de versión que requieren resolución manual. Se debe validar que actualizaciones no introducen breaking changes.

Restricciones : Debe auditar vulnerabilidades con herramientas (npm audit, Snyk), resolver conflictos de versión, ejecutar suite de pruebas completa, validar compatibilidad de APIs.

Producciones esperadas :

- Reporte de vulnerabilidades de dependencias
- Dependencias actualizadas sin breaking changes
- Archivo package-lock.json actualizado
- Suite de pruebas ejecutada exitosamente
- Documentación de cambios de dependencias

Competencias evaluadas : C26

Implementación Integral de Seguridad y Rendimiento

Contexto : Se desarrolla una nueva aplicación web que requiere implementar múltiples medidas de seguridad (autenticación, autorización, cabeceras HTTP) y optimizaciones de rendimiento (caché, monitorización). Se debe validar que todas las medidas funcionan correctamente juntas.

Restricciones : Debe cumplir con OWASP Top 10, implementar todas las cabeceras de seguridad HTTP, configurar caché eficientemente, monitorizar rendimiento, mantener cobertura de pruebas ≥85%.

Producciones esperadas :

- Aplicación con autenticación y autorización implementadas
- Cabeceras de seguridad HTTP configuradas correctamente
- Caché implementado y monitorizado
- Suite de pruebas de seguridad y rendimiento
- Documentación completa de medidas de seguridad
- Dashboard de monitorización de rendimiento

Competencias evaluadas : C23, C24, C25, C26

Pruebas esperadas

■ Pruebas documentales

- Especificación de sistema de autenticación con diagramas de flujo
- Reporte de auditoría de seguridad con vulnerabilidades identificadas
- Plan de remediación de vulnerabilidades con prioridades
- Documentación de cabeceras de seguridad HTTP implementadas
- Especificación de estrategia de caché con invalidación
- Reporte de vulnerabilidades de dependencias (npm audit, Snyk)
- Changelog de actualizaciones de dependencias
- Documentación de configuración de monitorización
- Especificación de alertas de rendimiento
- Matriz de compatibilidad de versiones

■ Pruebas de acción (en campo)

- Implementación de endpoints de autenticación (login, logout, refresh)
- Configuración de middleware de autorización
- Implementación de cabeceras de seguridad HTTP
- Configuración de caché en Redis
- Implementación de monitorización de rendimiento
- Actualización de dependencias vulnerables
- Resolución de conflictos de versión
- Ejecución de suite de pruebas de seguridad
- Ejecución de pruebas de carga
- Configuración de alertas de rendimiento

■ Pruebas reflexivas

- Análisis de decisiones de arquitectura de seguridad
- Evaluación de trade-offs entre seguridad y rendimiento
- Reflexión sobre lecciones aprendidas en auditorías de seguridad
- Análisis de impacto de actualizaciones de dependencias
- Evaluación de efectividad de medidas de caché
- Reflexión sobre mejoras continuas de seguridad

- Análisis de incidentes de seguridad y prevención
- Evaluación de cobertura de pruebas de seguridad

Criterios de éxito (7 familias)

Familia	Definición	Ponderación
Conformidad normativa	Cumplimiento con estándares OWASP Top 10 y buenas prácticas de seguridad web	Auditoría de seguridad con puntuación ≥90/100 y cero vulnerabilidades críticas
Calidad técnica	Implementación de autenticación, autorización y cabeceras de seguridad HTTP correctas	100% de endpoints sensibles protegidos, cabeceras de seguridad presentes en todas las respuestas
Comunicación	Documentación clara de medidas de seguridad, flujos de autenticación y configuración	Documentación completa con diagramas, ejemplos y guías de configuración
Análisis y resolución	Identificación sistemática de vulnerabilidades y análisis de causa raíz de problemas de rendimiento	Reporte de auditoría con vulnerabilidades categorizadas, planes de remediación con justificación
Autonomía e iniciativa	Proactividad en identificación de riesgos de seguridad y optimización de rendimiento	Implementación de monitorización y alertas sin requerimiento explícito, mejoras sugeridas
Colaboración	Comunicación efectiva de hallazgos de seguridad y coordinación de remediación con equipo	Feedback constructivo en code review, participación en decisiones de seguridad
Postura profesional	Compromiso con seguridad como responsabilidad compartida y mejora continua	Participación en capacitación de seguridad, promoción de buenas prácticas en equipo

Umbral de validación : Promedio ≥ 10/20

Bloque 7 - Despliegue, Infraestructura y Operaciones

Indicadores de desempeño por misión

Misión	Indicadores SMART
M06 — Reducir el tiempo de respuesta del servidor	Tiempo de respuesta promedio del servidor reducido en al menos 30% después de implementar caché Número de solicitudes al servidor reducido en al menos 40% mediante implementación de caché Latencia P95 de respuestas de API mejorada en al menos 25% Utilización de CPU del servidor reducida en al menos 20% después de optimizaciones
M08 — Actualizar dependencias y librerías	100% de dependencias actualizadas sin introducir vulnerabilidades críticas Cero breaking changes no documentados en actualizaciones de dependencias Tiempo de resolución de conflictos de versiones reducido a menos de 2 horas Cobertura de pruebas mantenida o mejorada después de actualizaciones
M09 — Gestionar el versionado y la evolución de APIs	Documentación de cambios de API completada antes de despliegue Cero clientes afectados por cambios de API no comunicados Compatibilidad hacia atrás mantenida en al menos 2 versiones anteriores Tiempo de migración de clientes a nueva versión de API reducido en al menos 50%
M10 — Implementar internacionalización y localización	Aplicación soporta al menos 3 idiomas diferentes sin duplicación de código Tiempo de traducción de nuevas características reducido en al menos 40% Cero errores de localización reportados en producción Cobertura de traducción de al menos 95% de contenido de usuario

Situaciones de evaluación

Configuración de entorno de desarrollo con Docker

Contexto : Un nuevo desarrollador se incorpora al equipo y necesita configurar su entorno de desarrollo local. El proyecto utiliza Node.js, PostgreSQL y Redis. Se requiere que el desarrollador configure Docker y Docker Compose para que todos los servicios funcionen correctamente.

Restricciones : El desarrollador debe completar la configuración en menos de 30 minutos. Debe documentar cualquier paso adicional necesario. La configuración debe ser reproducible en diferentes máquinas.

Producciones esperadas :

- Dockerfile funcional para la aplicación
- docker-compose.yml que define todos los servicios
- Documentación actualizada de pasos de configuración
- Script de inicialización que valida la configuración
- Verificación de que aplicación funciona correctamente

Competencias evaluadas : C29

Implementación de pipeline CI/CD con GitHub Actions

Contexto : El equipo necesita automatizar pruebas y despliegue. Se requiere configurar un pipeline de CI/CD que ejecute pruebas unitarias, análisis de código, y despliegue automático a staging en cada push a rama develop.

Restricciones : El pipeline debe ejecutarse en menos de 10 minutos. Debe fallar si cobertura de código cae por debajo del 80%. Debe notificar al equipo de fallos.

Producciones esperadas :

- Archivo .github/workflows/ci-cd.yml configurado
- Pruebas ejecutándose automáticamente en cada commit
- Reportes de cobertura generados y publicados
- Despliegue automático a staging después de pruebas exitosas
- Notificaciones de fallos configuradas

Competencias evaluadas : C30

Automatización de build y despliegue a producción

Contexto : Se necesita automatizar el proceso de compilación y despliegue de la aplicación a AWS. La aplicación debe compilarse con Webpack, optimizarse, y desplegarse a un bucket S3 con invalidación de CloudFront.

Restricciones : El despliegue debe completarse en menos de 5 minutos. Debe incluir validaciones de seguridad. Debe permitir rollback rápido si es necesario.

Producciones esperadas :

- Scripts de build optimizados
- Configuración de despliegue a S3
- Invalidación de CloudFront automatizada
- Versionado de artefactos implementado
- Documentación de proceso de despliegue
- Capacidad de rollback implementada

Competencias evaluadas : C31

Despliegue de aplicación full stack en plataforma cloud

Contexto : Se necesita desplegar una aplicación full stack (Node.js + React + PostgreSQL) en AWS. Se requiere configurar EC2 para el servidor, RDS para la base de datos, S3 para archivos estáticos, y CloudFront como CDN. Debe incluir escalado automático y monitorización.

Restricciones : La aplicación debe estar disponible 24/7 con SLA de 99.9%. Debe soportar al menos 1000 usuarios concurrentes. Debe incluir backups automáticos de base de datos.

Producciones esperadas :

- Instancias EC2 configuradas y ejecutándose
- Base de datos RDS creada y configurada
- Bucket S3 configurado para archivos estáticos
- CloudFront distribuido configurado
- Auto-scaling groups configurados
- Monitorización y alertas implementadas
- Backups automáticos configurados
- Documentación de infraestructura

Competencias evaluadas : C32

Optimización de rendimiento del servidor con caché

Contexto : La aplicación tiene problemas de rendimiento con tiempo de respuesta promedio de 2 segundos. Se necesita implementar caché con Redis para reducir carga de base de datos y mejorar rendimiento. Se requiere medir mejora de rendimiento antes y después.

Restricciones : Tiempo de respuesta debe reducirse a menos de 500ms. Caché debe invalidarse correctamente cuando datos cambian. Debe incluir monitorización de hit rate de caché.

Producciones esperadas :

- Redis configurado y ejecutándose
- Estrategia de caché implementada
- Invalidación de caché implementada
- Monitorización de caché configurada
- Métricas de rendimiento antes y después
- Documentación de estrategia de caché

Competencias evaluadas : C25

Pruebas esperadas

■ Pruebas documentales

- Documentación de configuración de entorno de desarrollo
- Archivos de configuración (Dockerfile, docker-compose.yml, .env.example)
- Definición de pipelines CI/CD (.github/workflows, .gitlab-ci.yml, Jenkinsfile)
- Scripts de build y despliegue
- Documentación de infraestructura (diagramas, especificaciones)
- Guías de despliegue y rollback
- Documentación de monitorización y alertas
- Registros de despliegue y cambios
- Especificaciones de escalabilidad
- Planes de disaster recovery

■ Pruebas de acción (en campo)

- Configuración exitosa de entorno de desarrollo local
- Ejecución exitosa de pipeline CI/CD
- Despliegue exitoso a staging y producción
- Validación de aplicación en entorno cloud
- Pruebas de escalado automático
- Pruebas de failover y recuperación
- Implementación de caché y validación de mejora de rendimiento
- Actualización de dependencias sin regresiones
- Despliegue de nueva versión de API con compatibilidad hacia atrás
- Implementación de soporte multiidioma

■ Pruebas reflexivas

- Análisis de decisiones de infraestructura y justificación
- Evaluación de estrategias de despliegue según requisitos
- Reflexión sobre mejoras de rendimiento logradas
- Análisis de incidentes de despliegue y lecciones aprendidas
- Evaluación de efectividad de monitorización
- Reflexión sobre escalabilidad de solución
- Análisis de impacto de cambios de API en clientes
- Evaluación de efectividad de estrategia de caché
- Reflexión sobre experiencia de onboarding de nuevos desarrolladores
- Análisis de mejoras en tiempo de despliegue

Criterios de éxito (7 familias)

Familia	Definición	Ponderación
Conformidad normativa	Configuración de infraestructura cumple con estándares de seguridad (HTTPS, firewalls, encriptación de datos en tránsito y en reposo)	Auditoría de seguridad de infraestructura sin hallazgos críticos
Calidad técnica	Pipeline CI/CD ejecuta todas las validaciones (pruebas, linting, análisis de seguridad) sin fallos	100% de validaciones pasadas en cada ejecución de pipeline
Calidad técnica	Despliegue automatizado se ejecuta sin errores y aplicación funciona correctamente en producción	Cero fallos de despliegue en últimas 10 ejecuciones
Calidad técnica	Infraestructura está configurada correctamente con escalado automático y monitorización activa	Aplicación mantiene rendimiento bajo carga con escalado automático funcionando
Comunicación	Documentación de configuración de entorno es clara y permite que nuevos desarrolladores se incorporen rápidamente	Nuevo desarrollador configura entorno exitosamente en menos de 30 minutos
Comunicación	Cambios de API están documentados y comunicados a clientes antes de despliegue	100% de cambios de API documentados con guías de migración
Análisis y resolución	Problemas de despliegue se identifican y resuelven rápidamente mediante análisis de logs y monitorización	Tiempo medio de resolución de incidentes de despliegue menor a 15 minutos
Análisis y resolución	Optimizaciones de rendimiento se validan con métricas antes y después	Mejora de rendimiento medida y documentada para cada optimización

Familia	Definición	Ponderación
Autonomía e iniciativa	Desarrollador configura y mantiene infraestructura de forma independiente sin supervisión constante	Desarrollador resuelve problemas de infraestructura sin escalación
Autonomía e iniciativa	Desarrollador identifica oportunidades de mejora en pipeline CI/CD y las implementa proactivamente	Al menos una mejora de pipeline implementada por trimestre
Colaboración	Documentación de infraestructura es compartida y accesible a todo el equipo	100% del equipo puede acceder y entender documentación de infraestructura
Colaboración	Cambios de infraestructura se comunican al equipo y se validan antes de despliegue	Cero cambios de infraestructura no comunicados
Postura profesional	Desarrollador sigue mejores prácticas de seguridad en configuración de infraestructura	Cero vulnerabilidades de seguridad introducidas en infraestructura
Postura profesional	Desarrollador mantiene documentación actualizada y procesos de operación claros	Documentación actualizada en cada cambio de infraestructura

Umbral de validación : Promedio ≥ 10/20

Bloque 8 - Mantenimiento, Evolución y Gestión de Incidencias

Indicadores de desempeño por misión

Misión	Indicadores SMART
M05 — Corregir incidencias detectadas en pruebas	Porcentaje de defectos corregidos en primera iteración (objetivo: ≥90%) Tiempo promedio de resolución de defectos (objetivo: <2 días) Cobertura de pruebas automatizadas para defectos corregidos (objetivo: ≥95%) Número de regresiones introducidas por correcciones (objetivo: 0)
M06 — Reducir el tiempo de respuesta del servidor	Reducción de tiempo de respuesta del servidor (objetivo: ≥30%) Tasa de acierto de caché (objetivo: ≥80%) Reducción de carga de servidor (CPU/memoria) (objetivo: ≥25%) Mejora en Core Web Vitals (objetivo: ≥20%)
M08 — Actualizar dependencias y librerías	Porcentaje de dependencias actualizadas (objetivo: ≥90%) Vulnerabilidades de seguridad resueltas (objetivo: 100%) Compatibilidad mantenida después de actualizaciones (objetivo: 100%) Tiempo de resolución de conflictos de versiones (objetivo: <1 día)
M09 — Gestionar el versionado y la evolución de APIs	Número de versiones de API mantenidas simultáneamente (objetivo: ≥2) Período de deprecación comunicado (objetivo: ≥3 meses) Clientes migrados a nueva versión (objetivo: ≥95%) Documentación de cambios completada (objetivo: 100%)
M10 — Implementar internacionalización y localización	Número de idiomas soportados (objetivo: ≥3) Cobertura de traducción (objetivo: ≥95%) Idiomas testeados (objetivo: 100%) Adaptación cultural verificada (objetivo: 100%)

Situaciones de evaluación

Corrección de defecto crítico en producción

Contexto : Se ha reportado un defecto crítico en producción que afecta a usuarios. El desarrollador debe identificar la causa raíz, implementar una corrección, validarla mediante pruebas y desplegar la solución minimizando impacto.

Restricciones : Tiempo limitado para resolución, necesidad de validar que corrección no introduce regresiones, documentación de causa y solución requerida

Producciones esperadas :

- Análisis detallado de causa raíz con evidencia de debugging
- Código corregido con explicación de cambios
- Casos de prueba que validan corrección
- Documentación de solución y prevención futura

Competencias evaluadas : C20, C21

Optimización de rendimiento de API lenta

Contexto : Una API está respondiendo lentamente afectando experiencia de usuario. El desarrollador debe analizar métricas de rendimiento, identificar cuellos de botella, implementar optimizaciones (caché, índices, etc.) y validar mejora.

Restricciones : Debe mantener funcionalidad existente, validar que caché no causa inconsistencias, documentar cambios de arquitectura

Producciones esperadas :

- Análisis de rendimiento con métricas antes/después
- Implementación de estrategias de caché
- Optimización de consultas o código
- Validación de mejora con herramientas de monitorización

Competencias evaluadas : C25

Actualización de dependencias con resolución de conflictos

Contexto : Múltiples dependencias tienen actualizaciones disponibles incluyendo parches de seguridad. El desarrollador debe evaluar compatibilidad, resolver conflictos de versiones, validar que no hay regresiones y documentar cambios.

Restricciones : Debe validar compatibilidad con código existente, resolver conflictos de versiones, ejecutar suite de pruebas completa

Producciones esperadas :

- Análisis de vulnerabilidades y compatibilidad
- Resolución de conflictos de versiones
- Validación mediante pruebas automatizadas
- Documentación de cambios y breaking changes

Competencias evaluadas : C26

Implementación de versionado de API con compatibilidad hacia atrás

Contexto : Se necesita introducir cambios en una API existente manteniendo compatibilidad con clientes antiguos. El desarrollador debe diseñar estrategia de versionado, implementar múltiples versiones, comunicar cambios y facilitar migración.

Restricciones : Debe mantener compatibilidad hacia atrás, documentar cambios claramente, proporcionar período de transición

Producciones esperadas :

- Diseño de estrategia de versionado
- Implementación de múltiples versiones de API
- Documentación de cambios y guía de migración
- Comunicación a clientes sobre cambios

Competencias evaluadas : C27

Implementación de internacionalización en aplicación existente

Contexto : Una aplicación debe expandirse a múltiples mercados internacionales. El desarrollador debe implementar i18n, localizar contenido para diferentes regiones, adaptar formatos y validar en múltiples idiomas.

Restricciones : Debe separar contenido de código, adaptar formatos según región, validar en múltiples idiomas, considerar direccionalidad RTL

Producciones esperadas :

- Implementación de librería de i18n
- Archivos de traducción para múltiples idiomas
- Adaptación de formatos (fecha, moneda, plurales)
- Validación en múltiples idiomas y regiones

Competencias evaluadas : C28

Pruebas esperadas

■ Pruebas documentales

- Análisis de causa raíz documentado con evidencia de debugging
- Reportes de rendimiento antes/después de optimizaciones
- Matriz de compatibilidad de dependencias
- Documentación de estrategia de versionado de API
- Changelog y release notes comunicados a clientes
- Archivos de traducción y configuración de i18n
- Documentación de formatos localizados
- Especificación de período de deprecación

■ Pruebas de acción (en campo)

- Ejecución de debugging sistemático para identificar causa raíz
- Implementación de correcciones de defectos
- Configuración de estrategias de caché (Redis, Memcached)
- Optimización de consultas y código
- Actualización de dependencias con resolución de conflictos
- Implementación de múltiples versiones de API
- Implementación de librería de i18n
- Localización de contenido para múltiples regiones
- Despliegue de cambios en producción
- Configuración de monitorización y alertas

■ Pruebas reflexivas

- Reflexión sobre causa raíz y prevención futura
- Análisis de impacto de optimizaciones en arquitectura
- Evaluación de estrategia de versionado elegida
- Análisis de experiencia de usuario en diferentes idiomas
- Reflexión sobre lecciones aprendidas en resolución de incidencias
- Evaluación de efectividad de estrategias de caché
- Análisis de feedback de usuarios sobre cambios de API
- Reflexión sobre adaptación cultural de contenido

Criterios de éxito (7 familias)

Familia	Definición	Ponderación
Conformidad normativa	Cumplimiento de estándares de seguridad en actualizaciones de dependencias y resolución de vulnerabilidades	100% de vulnerabilidades críticas resueltas, auditoría de seguridad aprobada
Calidad técnica	Calidad técnica de correcciones, optimizaciones e implementaciones de i18n	Cobertura de pruebas ≥95%, sin regresiones introducidas, rendimiento mejorado ≥30%
Comunicación	Comunicación clara de cambios de API, deprecaciones y actualizaciones a stakeholders	Documentación completa, changelog actualizado, período de transición comunicado
Análisis y resolución	Análisis sistemático de causa raíz en defectos e identificación de cuellos de botella	Causa raíz documentada, soluciones implementadas resuelven problema, prevención de recurrencia
Autonomía e iniciativa	Capacidad de identificar y resolver problemas de forma proactiva sin supervisión constante	Defectos resueltos en primera iteración ≥90%, optimizaciones implementadas sin requerimiento
Colaboración	Colaboración efectiva con equipo en resolución de incidencias y comunicación de cambios	Feedback positivo de equipo, documentación compartida, participación en post-mortem
Postura profesional	Responsabilidad en mantenimiento de aplicaciones, consideración de impacto en usuarios y compromiso con calidad	Cero incidencias causadas por cambios, usuarios satisfechos con mejoras, documentación actualizada

Umbral de validación : Promedio ≥ 10/20

Bloque 9 - Documentación, Comunicación y Colaboración

Indicadores de desempeño por misión

Misión	Indicadores SMART
M04 — Revisar código de otros desarrolladores	Número de comentarios de code review proporcionados por sesión de revisión Porcentaje de feedback técnico que incluye referencias a estándares o buenas prácticas Tiempo promedio de respuesta a solicitudes de revisión de código Satisfacción del equipo con la calidad y constructividad del feedback recibido
M09 — Gestionar el versionado y la evolución de APIs	Completitud de documentación de API (cobertura de endpoints, parámetros, respuestas) Claridad de comunicación de cambios de API a clientes Número de migraciones exitosas de clientes a nuevas versiones de API Tiempo de resolución de problemas de compatibilidad reportados
M11 — Analizar requisitos funcionales	Número de sesiones de validación de requisitos realizadas Completitud de especificaciones técnicas documentadas Porcentaje de requisitos clarificados antes de implementación Satisfacción de stakeholders con comprensión de requisitos

Situaciones de evaluación

Revisión de código y feedback técnico

Contexto : El desarrollador debe revisar código de un compañero que implementa una nueva característica de autenticación. El código incluye algunos problemas de seguridad, falta de documentación y oportunidades de mejora en estructura.

Restricciones : Debe proporcionar feedback en máximo 2 horas, siendo constructivo y educativo. Debe incluir referencias a estándares de seguridad y mejores prácticas.

Producciones esperadas :

- Comentarios de revisión detallados en pull request
- Identificación de vulnerabilidades de seguridad
- Sugerencias de mejora con justificación técnica
- Referencias a documentación o estándares relevantes
- Tono respetuoso y constructivo en toda la comunicación

Competencias evaluadas : C22

Documentación de API REST y comunicación de cambios

Contexto : El desarrollador debe documentar una nueva versión de API que incluye cambios en estructura de respuestas y deprecación de algunos endpoints. Debe comunicar estos cambios a clientes existentes.

Restricciones : Documentación debe ser completa en Swagger/OpenAPI. Comunicación debe ser clara sobre breaking changes y proporcionar guía de migración.

Producciones esperadas :

- Especificación Swagger/OpenAPI completa
- Ejemplos de solicitudes y respuestas
- Guía de migración para clientes
- Comunicación clara de cambios y timeline de deprecación
- Documentación de compatibilidad hacia atrás

Competencias evaluadas : C8, C27

Análisis de requisitos y comunicación con stakeholders

Contexto : El desarrollador participa en sesión de recopilación de requisitos con cliente para nueva funcionalidad de reportes. Debe documentar requisitos funcionales y técnicos, validar comprensión y obtener aprobación.

Restricciones : Debe producir especificación clara con criterios de aceptación. Debe realizar sesión de validación y documentar cambios solicitados.

Producciones esperadas :

- Documento de especificación de requisitos
- Criterios de aceptación medibles y verificables
- Casos de uso documentados
- Notas de sesión de validación
- Matriz de trazabilidad de requisitos

Competencias evaluadas : C1, C11

Documentación de configuración e integración

Contexto : El desarrollador debe documentar proceso de configuración de entorno de desarrollo y integración con servicio externo de pagos. Documentación será utilizada por nuevos desarrolladores.

Restricciones : Documentación debe ser clara, paso a paso, con ejemplos funcionales. Debe incluir troubleshooting común.

Producciones esperadas :

- Guía de instalación y configuración
- Variables de entorno documentadas
- Ejemplos de integración con servicio de pagos
- Guía de troubleshooting
- Checklist de verificación de configuración

Competencias evaluadas : C9, C29

Comunicación de decisiones arquitectónicas

Contexto : El desarrollador debe documentar y comunicar decisión de cambiar de arquitectura monolítica a microservicios. Debe justificar decisión, explicar impacto y proporcionar plan de migración.

Restricciones : Debe incluir análisis de alternativas, justificación técnica y plan de implementación. Debe ser comunicado a todo el equipo.

Producciones esperadas :

- Documento de decisión arquitectónica (ADR)
- Análisis de alternativas consideradas
- Justificación técnica de decisión
- Diagrama de arquitectura nueva
- Plan de migración con timeline
- Presentación a equipo

Competencias evaluadas : C2, C13

Pruebas esperadas

■ Pruebas documentales

- Especificaciones técnicas documentadas con criterios de aceptación
- Documentación de APIs en formato Swagger/OpenAPI
- Comentarios de code review en pull requests
- Documentación de configuración e instalación
- Guías de integración con servicios externos
- Documentos de decisión arquitectónica (ADR)
- Notas de sesiones de validación con stakeholders
- Changelog y notas de versión
- Guías de migración y compatibilidad
- Wikis y bases de conocimiento del proyecto

■ Pruebas de acción (en campo)

- Participación en sesiones de validación de requisitos
- Realización de code review constructivo
- Comunicación de cambios de API a clientes
- Configuración de entornos de desarrollo
- Documentación de integraciones con servicios externos
- Presentación de decisiones técnicas al equipo
- Sesiones de capacitación para nuevos desarrolladores
- Pair programming para transferencia de conocimiento
- Participación en reuniones de diseño arquitectónico
- Actualización de documentación ante cambios

■ Pruebas reflexivas

- Reflexión sobre efectividad de comunicación con stakeholders
- Análisis de feedback recibido en code review
- Evaluación de claridad de documentación producida
- Reflexión sobre decisiones arquitectónicas tomadas
- Análisis de impacto de cambios comunicados
- Evaluación de efectividad de sesiones de capacitación
- Reflexión sobre mejoras en procesos de documentación

- Análisis de lecciones aprendidas en proyectos
- Evaluación de colaboración con equipo
- Reflexión sobre evolución de habilidades de comunicación

Criterios de éxito (7 familias)

Familia	Definición	Ponderación
Conformidad normativa	Documentación cumple con estándares de especificación técnica requeridos por la organización	100% de documentación incluye elementos obligatorios (descripción, ejemplos, criterios de aceptación)
Calidad técnica	Feedback de code review identifica problemas técnicos reales y propone soluciones viables	80% mínimo de comentarios de revisión son técnicamente precisos y constructivos
Comunicación	Comunicación es clara, precisa y accesible para audiencia objetivo (técnica o no técnica)	Feedback de stakeholders indica comprensión clara de requisitos y decisiones comunicadas
Análisis y resolución	Análisis de requisitos identifica restricciones técnicas y propone soluciones viables	100% de especificaciones incluyen análisis de viabilidad técnica
Autonomía e iniciativa	Desarrollador toma iniciativa en documentación sin esperar instrucciones explícitas	Documentación se mantiene actualizada proactivamente ante cambios de código
Colaboración	Comunicación facilita colaboración efectiva y transferencia de conocimiento en equipo	Equipo reporta que documentación y feedback facilitan su trabajo
Postura profesional	Comunicación es respetuosa, constructiva y orientada al aprendizaje mutuo	Feedback de equipo indica tono profesional y actitud colaborativa en todas las interacciones

Umbral de validación : Promedio ≥ 10/20

Bloque 10 - Planificación, Estimación y Gestión de Proyectos

Indicadores de desempeño por misión

Misión	Indicadores SMART
M11 — Analizar requisitos funcionales	Porcentaje de requisitos funcionales documentados con criterios de aceptación verificables Número de sesiones de validación realizadas con stakeholders Completitud de especificaciones técnicas (cobertura de casos de uso) Tasa de cambios de requisitos identificados tempranamente vs. tardíamente
M12 — Analizar requisitos técnicos	Número de restricciones técnicas identificadas y documentadas Porcentaje de evaluaciones de viabilidad técnica completadas antes de inicio de desarrollo Precisión de evaluaciones de viabilidad (coincidencia con realidad de implementación) Tiempo dedicado a análisis de restricciones técnicas
M13 — Diseñar la arquitectura de la solución web	Completitud de documentación de arquitectura (diagramas, componentes, flujos) Número de patrones arquitectónicos identificados y justificados Alineación de arquitectura con requisitos funcionales y técnicos Participación en revisiones de arquitectura

Situaciones de evaluación

Estimación de Esfuerzo para Característica Compleja

Contexto : En una reunión de planificación de sprint, el equipo necesita estimar el esfuerzo para implementar una característica que requiere cambios en front-end, back-end y base de datos. La característica incluye integración con un servicio externo y requiere optimización de rendimiento.

Restricciones : Tiempo limitado para estimación (máximo 30 minutos), información incompleta sobre requisitos exactos, necesidad de consenso del equipo

Producciones esperadas :

- Estimación en puntos de historia o días de trabajo
- Desglose de tareas técnicas necesarias
- Identificación de riesgos técnicos y dependencias
- Justificación de estimación basada en complejidad
- Comunicación clara de incertidumbre y supuestos

Competencias evaluadas : C1, C2, C3

Identificación y Comunicación de Riesgos Técnicos

Contexto : Durante el análisis de requisitos para un nuevo proyecto, el desarrollador identifica varios riesgos técnicos potenciales: tecnología nueva para el equipo, integración compleja con sistema legacy, y requisitos de rendimiento muy exigentes. Debe comunicar estos riesgos al equipo y proponer mitigaciones.

Restricciones : Necesidad de ser constructivo y no alarmista, basarse en hechos técnicos, proponer soluciones viables

Producciones esperadas :

- Documento de registro de riesgos identificados
- Análisis de probabilidad e impacto de cada riesgo
- Propuestas de mitigación técnica
- Plan de contingencia para riesgos críticos
- Presentación clara a stakeholders

Competencias evaluadas : C1, C2, C3

Participación en Planificación de Sprint y Priorización

Contexto : El equipo está planificando el próximo sprint. Hay más trabajo solicitado que capacidad disponible. El desarrollador debe contribuir a decisiones de priorización considerando viabilidad técnica, dependencias y valor de negocio.

Restricciones : Múltiples perspectivas en conflicto, presión para incluir más trabajo, necesidad de mantener calidad

Producciones esperadas :

- Análisis de viabilidad técnica de historias de usuario propuestas
- Identificación de dependencias entre historias
- Recomendaciones de priorización basadas en riesgos técnicos
- Estimaciones de esfuerzo para historias priorizadas
- Comunicación clara de trade-offs técnicos

Competencias evaluadas : C1, C2, C3

Seguimiento y Comunicación de Desviaciones de Cronograma

Contexto : Durante la ejecución de un sprint, el desarrollador se da cuenta de que una tarea técnica está tomando más tiempo del estimado debido a complejidad inesperada. Debe comunicar la desviación, analizar causa raíz y proponer ajustes.

Restricciones : Necesidad de comunicación temprana, evitar sorpresas al final del sprint, mantener transparencia

Producciones esperadas :

- Comunicación temprana de desviación
- Análisis de causa raíz de la desviación
- Estimación revisada de tiempo restante
- Propuestas de ajuste (reducción de alcance, extensión de plazo, recursos adicionales)
- Lecciones aprendidas para futuras estimaciones

Competencias evaluadas : C1, C2, C3

Evaluación de Viabilidad Técnica de Requisitos Nuevos

Contexto : El cliente solicita una nueva característica que requiere tecnología no utilizada anteriormente en el proyecto. El desarrollador debe evaluar viabilidad técnica, esfuerzo de implementación, riesgos y opciones alternativas.

Restricciones : Información limitada sobre requisitos exactos, necesidad de investigación rápida, presión de cliente por respuesta rápida

Producciones esperadas :

- Análisis de viabilidad técnica detallado
- Evaluación de opciones tecnológicas
- Estimación de esfuerzo de implementación
- Identificación de riesgos técnicos
- Recomendación con justificación técnica
- Propuesta de plan de implementación

Competencias evaluadas : C1, C2, C3

Pruebas esperadas

■ Pruebas documentales

- Documento de estimación de esfuerzo con desglose de tareas
- Registro de riesgos técnicos identificados y mitigaciones propuestas
- Plan de proyecto con cronograma y hitos
- Especificaciones técnicas de requisitos
- Documentación de arquitectura de solución
- Actas de reuniones de planificación
- Reportes de seguimiento de progreso
- Análisis de desviaciones de cronograma
- Documentación de decisiones técnicas y justificaciones
- Matriz de priorización de requisitos

■ Pruebas de acción (en campo)

- Participación en reuniones de estimación y planificación
- Comunicación proactiva de riesgos técnicos identificados
- Contribución a decisiones de priorización basadas en viabilidad técnica
- Seguimiento de tareas asignadas y comunicación de desviaciones
- Propuesta de mitigaciones para riesgos identificados
- Colaboración en refinamiento de requisitos con stakeholders
- Análisis de causa raíz de desviaciones de estimación
- Participación en retrospectivas para mejora de estimaciones
- Comunicación clara de restricciones técnicas y limitaciones
- Escalada oportuna de riesgos críticos

■ Pruebas reflexivas

- Reflexión sobre precisión de estimaciones realizadas
- Análisis de factores que afectaron desviaciones de cronograma
- Evaluación de efectividad de mitigaciones de riesgos implementadas
- Identificación de patrones en estimaciones (tendencia a sobrestimar o subestimar)
- Reflexión sobre comunicación de riesgos y su impacto
- Análisis de lecciones aprendidas en planificación de proyectos

- Evaluación de participación en decisiones de priorización
- Reflexión sobre mejora en habilidades de estimación
- Análisis de impacto de decisiones técnicas en cronograma
- Evaluación de autonomía e iniciativa en gestión de proyectos

Criterios de éxito (7 familias)

Familia	Definición	Ponderación
Conformidad normativa	Las estimaciones y planes de proyecto cumplen con estándares de documentación y trazabilidad requeridos por la organización	100% de estimaciones documentadas con justificación técnica, 100% de riesgos registrados en sistema de seguimiento
Calidad técnica	Las estimaciones de esfuerzo son realistas y basadas en análisis técnico profundo de complejidad	Desviación promedio de estimaciones ≤ 20%, precisión de evaluaciones de viabilidad técnica ≥ 90%
Comunicación	La comunicación de estimaciones, riesgos y restricciones técnicas es clara, oportuna y comprensible para stakeholders técnicos y no técnicos	Feedback positivo de stakeholders sobre claridad de comunicación, 100% de riesgos críticos comunicados antes de inicio de implementación
Análisis y resolución	El análisis de requisitos, riesgos y viabilidad técnica es profundo, considerando múltiples perspectivas y opciones	Mínimo 3 opciones técnicas evaluadas para decisiones significativas, análisis de causa raíz completado para 100% de desviaciones
Autonomía e iniciativa	El desarrollador identifica proactivamente riesgos técnicos, propone mitigaciones y toma iniciativa en planificación sin esperar instrucciones	Mínimo 2 riesgos identificados proactivamente por sprint, participación activa en 100% de reuniones de planificación
Colaboración	La participación en planificación y estimación es colaborativa, considerando perspectivas del equipo y contribuyendo a consenso	Participación en 100% de sesiones de estimación, feedback positivo de equipo sobre colaboración, consenso alcanzado en 100% de decisiones de priorización
Postura profesional	El desarrollador demuestra responsabilidad en estimaciones, transparencia en comunicación de desviaciones y compromiso con objetivos del proyecto	Cumplimiento de 100% de compromisos de cronograma o comunicación temprana de desviaciones, actitud constructiva en gestión de cambios

Umbral de validación : Promedio ≥ 10/20

Bloque 11 - Control de Versiones y Gestión de Código

Indicadores de desempeño por misión

Misión	Indicadores SMART
M04 — Revisar código de otros desarrolladores	Número de comentarios constructivos proporcionados en code reviews por semana Porcentaje de pull requests revisados dentro de 24 horas Tasa de aceptación de feedback en pull requests (cambios implementados / comentarios recibidos) Consistencia en aplicación de estándares de código identificados en reviews
M08 — Actualizar dependencias y librerías	Número de dependencias actualizadas sin introducir breaking changes Tiempo promedio para resolver conflictos de versiones Porcentaje de actualizaciones validadas con suite de pruebas Reducción de vulnerabilidades de seguridad tras actualización de dependencias

Situaciones de evaluación

Gestión de rama de feature con múltiples commits

Contexto : El desarrollador debe crear una rama de feature para implementar una nueva funcionalidad, realizar múltiples commits con cambios incrementales, mantener la rama sincronizada con la rama principal, y finalmente crear un pull request para revisión.

Restricciones : La rama debe estar basada en la rama principal más reciente, los commits deben ser atómicos y descriptivos, no debe haber conflictos sin resolver, y el pull request debe incluir descripción clara de cambios.

Producciones esperadas :

- Rama de feature creada con nombre descriptivo
- Mínimo 3 commits con mensajes claros y descriptivos
- Rama sincronizada con cambios recientes de rama principal
- Pull request creado con descripción detallada de cambios
- Historial de commits limpio sin commits de merge innecesarios

Competencias evaluadas : C22, C26

Resolución de conflictos de merge en colaboración

Contexto : Dos desarrolladores han realizado cambios en el mismo archivo en ramas diferentes. El desarrollador debe resolver los conflictos de merge manteniendo la integridad del código y comunicando cambios al equipo.

Restricciones : Los conflictos deben resolverse sin perder cambios válidos, el código resultante debe ser funcional, debe haber comunicación clara sobre decisiones de resolución, y se debe validar que no se introducen regresiones.

Producciones esperadas :

- Conflictos identificados y resueltos correctamente
- Código resultante funcional y sin errores de sintaxis
- Mensaje de commit de merge explicando decisiones
- Validación de que cambios de ambas ramas se preservan
- Comunicación con otros desarrolladores sobre resolución

Competencias evaluadas : C22, C26

Code review de pull request con identificación de problemas

Contexto : El desarrollador debe revisar un pull request de un compañero, identificar problemas de calidad, seguridad o adherencia a estándares, y proporcionar feedback constructivo con sugerencias de mejora.

Restricciones : El feedback debe ser específico y accionable, debe incluir referencias a estándares o buenas prácticas, debe ser respetuoso y constructivo, y debe ayudar al desarrollador a mejorar su código.

Producciones esperadas :

- Identificación de al menos 3 problemas o áreas de mejora
- Comentarios específicos en líneas de código problemáticas
- Sugerencias de mejora con referencias a estándares
- Aprobación o solicitud de cambios justificada
- Tono respetuoso y constructivo en todos los comentarios

Competencias evaluadas : C22

Actualización de dependencias con validación de compatibilidad

Contexto : El desarrollador debe actualizar una o más dependencias del proyecto, resolver conflictos de versiones si existen, validar que la aplicación sigue funcionando correctamente, y documentar cambios realizados.

Restricciones : Las actualizaciones deben validarse con suite de pruebas completa, no debe haber breaking changes sin justificación, debe haber comunicación clara sobre cambios, y se debe verificar que no se introducen vulnerabilidades de seguridad.

Producciones esperadas :

- Dependencias actualizadas a versiones compatibles
- Conflictos de versiones resueltos
- Suite de pruebas ejecutada exitosamente
- Documentación de cambios en dependencias
- Verificación de seguridad de nuevas versiones
- Commit con mensaje claro describiendo actualizaciones

Competencias evaluadas : C26

Organización y documentación de estructura de código

Contexto : El desarrollador debe organizar el código de un módulo siguiendo convenciones del proyecto, documentar la estructura de directorios y archivos, y asegurar que otros desarrolladores puedan entender y navegar el código fácilmente.

Restricciones : La estructura debe ser escalable y mantenible, debe seguir convenciones del proyecto, debe incluir documentación clara, y debe facilitar reutilización de código.

Producciones esperadas :

- Estructura de directorios bien organizada
- Archivos con nombres descriptivos y consistentes
- Documentación de estructura en README o guía
- Ejemplos de uso de módulos principales
- Comentarios en código complejo explicando lógica
- Separación clara de responsabilidades entre archivos

Competencias evaluadas : C22, C26

Pruebas esperadas

■ Pruebas documentales

- Historial de commits en repositorio con mensajes descriptivos
- Pull requests con descripciones detalladas de cambios
- Documentación de estructura de código y convenciones
- Comentarios de code review proporcionados a otros desarrolladores
- Registros de resolución de conflictos de merge
- Documentación de actualizaciones de dependencias
- README y guías de configuración del proyecto
- Especificaciones técnicas de módulos y componentes

■ Pruebas de acción (en campo)

- Crear ramas de feature con nombres descriptivos
- Realizar commits atómicos con mensajes claros
- Sincronizar ramas con rama principal regularmente
- Crear pull requests con descripción completa
- Revisar código de otros desarrolladores
- Proporcionar feedback constructivo en code reviews
- Resolver conflictos de merge manteniendo integridad
- Actualizar dependencias validando compatibilidad
- Ejecutar suite de pruebas después de cambios
- Documentar decisiones técnicas en commits

■ Pruebas reflexivas

- Reflexión sobre calidad de mensajes de commit y cómo mejorar claridad
- Análisis de feedback recibido en code reviews y aplicación de mejoras
- Evaluación de decisiones de resolución de conflictos y alternativas
- Reflexión sobre impacto de cambios en otros desarrolladores
- Análisis de patrones en actualizaciones de dependencias
- Evaluación de efectividad de comunicación en pull requests
- Reflexión sobre mejoras en organización de código
- Análisis de lecciones aprendidas en colaboración con equipo

Criterios de éxito (7 familias)

Familia	Definición	Ponderación
Conformidad normativa	El desarrollador sigue políticas de control de versiones del proyecto incluyendo convenciones de branching, naming y commit	Auditoría de historial de Git verificando adherencia a políticas documentadas
Calidad técnica	Los commits son atómicos, con mensajes descriptivos que explican el qué y el por qué de los cambios	Análisis de historial de commits evaluando claridad y completitud de mensajes
Comunicación	El desarrollador comunica cambios claramente en pull requests y proporciona feedback constructivo en code reviews	Evaluación de descripciones de pull requests y comentarios de code review por claridad y utilidad
Análisis y resolución	El desarrollador identifica y resuelve conflictos de merge manteniendo integridad del código	Validación de resoluciones de conflictos verificando que cambios válidos se preservan
Autonomía e iniciativa	El desarrollador gestiona ramas de forma independiente, sincroniza cambios proactivamente y resuelve problemas sin supervisión	Observación de capacidad para trabajar con Git sin asistencia y resolver problemas de forma autónoma
Colaboración	El desarrollador colabora efectivamente en code reviews, respeta el trabajo de otros y proporciona feedback respetuoso	Evaluación de interacciones en pull requests y feedback de compañeros sobre colaboración
Postura profesional	El desarrollador demuestra responsabilidad por sus cambios, documenta decisiones y mantiene estándares de calidad consistentemente	Análisis de patrones en commits, documentación y adherencia a estándares a lo largo del tiempo

Umbral de validación : Promedio ≥ 10/20

Bloque 12 - Funcionalidades Avanzadas y Experiencia de Usuario

Indicadores de desempeño por misión

Misión	Indicadores SMART
M02 — Adaptar la interfaz a distintos dispositivos	Porcentaje de dispositivos testeados donde la interfaz es completamente accesible (objetivo: 100%) Número de errores de accesibilidad detectados por herramientas automáticas (objetivo: 0) Tiempo de navegación por teclado completa sin bloqueos (objetivo: < 2 minutos para flujo crítico) Conformidad WCAG alcanzada (objetivo: AA mínimo)
M03 — Aplicar diseño responsive	Porcentaje de componentes con atributos ARIA implementados correctamente (objetivo: 100%) Validación de estructura semántica HTML (objetivo: 0 errores) Cobertura de navegación por teclado en componentes interactivos (objetivo: 100%) Puntuación de accesibilidad en Lighthouse (objetivo: >= 90/100)
M05 — Corregir incidencias detectadas en pruebas	Número de defectos de accesibilidad corregidos (objetivo: 100% de defectos reportados) Tiempo promedio de resolución de defectos de accesibilidad (objetivo: < 3 días) Pruebas de regresión de accesibilidad ejecutadas (objetivo: 100% de cambios) Validación de correcciones con herramientas de auditoría (objetivo: 0 errores residuales)

Situaciones de evaluación

Auditoría de Accesibilidad de Aplicación Existente

Contexto : Se proporciona una aplicación web existente con múltiples páginas y componentes interactivos. El desarrollador debe realizar una auditoría completa de accesibilidad utilizando herramientas automáticas y testing manual para identificar problemas de conformidad WCAG.

Restricciones : Debe completarse en 4 horas. Debe cubrir al menos 5 páginas principales. Debe utilizar al menos 2 herramientas de auditoría diferentes. Debe incluir testing con navegación por teclado.

Producciones esperadas :

- Reporte de auditoría detallado con errores categorizados por severidad
- Lista de problemas WCAG identificados con referencias a criterios específicos
- Capturas de pantalla de errores de accesibilidad
- Recomendaciones de corrección con prioridad
- Matriz de conformidad WCAG (A, AA, AAA)

Competencias evaluadas : C30

Implementación de Atributos ARIA en Componentes Complejos

Contexto : Se proporciona un conjunto de componentes interactivos complejos (modales, menús desplegables, carruseles, pestañas) sin atributos ARIA. El desarrollador debe implementar ARIA correctamente para que sean accesibles a lectores de pantalla.

Restricciones : Debe implementar al menos 5 componentes diferentes. Debe validar con herramientas de auditoría. Debe testear con lector de pantalla (NVDA o VoiceOver). Debe mantener funcionalidad existente.

Producciones esperadas :

- Código con atributos ARIA implementados correctamente
- Documentación de roles, propiedades y estados ARIA utilizados
- Validación de auditoría sin errores ARIA
- Grabación de testing con lector de pantalla
- Guía de uso de componentes accesibles

Competencias evaluadas : C30

Validación de Navegación por Teclado Completa

Contexto : Se proporciona una aplicación web con múltiples flujos de usuario. El desarrollador debe verificar que todos los elementos interactivos son accesibles mediante navegación por teclado sin necesidad de ratón.

Restricciones : Debe completarse sin usar ratón. Debe cubrir todos los flujos críticos de usuario. Debe documentar orden de tabulación. Debe validar indicadores visuales de foco.

Producciones esperadas :

- Documento de validación de navegación por teclado
- Mapa de orden de tabulación (tabindex)
- Identificación de elementos no accesibles por teclado
- Recomendaciones de corrección
- Evidencia de testing (video o capturas)

Competencias evaluadas : C30

Corrección de Defectos de Accesibilidad Reportados

Contexto : Se proporciona una lista de defectos de accesibilidad identificados en testing previo. El desarrollador debe analizar cada defecto, implementar correcciones y validar que se resuelven sin introducir regresiones.

Restricciones : Debe corregir al menos 10 defectos. Debe validar cada corrección con herramientas automáticas. Debe ejecutar pruebas de regresión. Debe documentar cambios realizados.

Producciones esperadas :

- Código corregido con cambios documentados
- Reporte de validación post-corrección
- Pruebas de regresión ejecutadas
- Documentación de causa raíz de cada defecto
- Actualización de pruebas automatizadas para prevenir recurrencia

Competencias evaluadas : C30

Integración de Testing de Accesibilidad en Pipeline CI/CD

Contexto : Se proporciona un proyecto con pipeline CI/CD existente. El desarrollador debe integrar herramientas de testing de accesibilidad automático para validar conformidad WCAG en cada commit.

Restricciones : Debe implementar al menos 2 herramientas de auditoría automática. Debe generar reportes de accesibilidad. Debe fallar el build si hay errores críticos. Debe documentar configuración.

Producciones esperadas :

- Configuración de herramientas de auditoría en CI/CD
- Scripts de validación de accesibilidad
- Reportes de accesibilidad generados automáticamente
- Documentación de configuración y uso
- Ejemplos de ejecución exitosa y fallida

Competencias evaluadas : C30

Pruebas esperadas

■ Pruebas documentales

- Reportes de auditoría de accesibilidad con errores categorizados
- Documentación de atributos ARIA implementados
- Mapas de navegación por teclado y orden de tabulación
- Especificaciones de conformidad WCAG alcanzada
- Guías de accesibilidad para componentes reutilizables
- Configuración de herramientas de testing de accesibilidad
- Reportes de validación post-corrección
- Documentación de cambios de accesibilidad en commits

■ Pruebas de acción (en campo)

- Ejecución de auditorías automáticas con axe, WAVE o Lighthouse
- Implementación de atributos ARIA en componentes interactivos
- Testing manual con navegación por teclado (Tab, Shift+Tab, Enter, Escape)
- Testing con lectores de pantalla (NVDA, JAWS, VoiceOver)
- Corrección de defectos de accesibilidad identificados
- Validación de indicadores visuales de foco
- Integración de herramientas de auditoría en pipelines CI/CD
- Ejecución de pruebas de regresión de accesibilidad

■ Pruebas reflexivas

- Análisis de impacto de cambios de accesibilidad en experiencia de usuario
- Reflexión sobre barreras de accesibilidad identificadas y soluciones implementadas
- Evaluación de efectividad de atributos ARIA implementados
- Análisis de cobertura de accesibilidad alcanzada vs. objetivos
- Identificación de patrones de errores de accesibilidad recurrentes
- Reflexión sobre mejoras en proceso de desarrollo para prevenir problemas de accesibilidad
- Evaluación de herramientas de auditoría utilizadas y su efectividad
- Análisis de feedback de usuarios con discapacidades sobre accesibilidad

Criterios de éxito (7 familias)

Familia	Definición	Ponderación
Conformidad normativa	La aplicación cumple con estándares WCAG 2.1 nivel AA mínimo en todas las páginas evaluadas	Auditoría de accesibilidad sin errores críticos o mayores de WCAG
Calidad técnica	Los atributos ARIA están implementados correctamente sin conflictos con HTML semántico	Validación de estructura ARIA sin errores en herramientas de auditoría
Comunicación	La navegación por teclado es clara y predecible con indicadores visuales de foco evidentes	100% de elementos interactivos accesibles por teclado con foco visible
Análisis y resolución	Los defectos de accesibilidad se analizan identificando causa raíz y se implementan soluciones sostenibles	Tasa de recurrencia de defectos de accesibilidad < 10%
Autonomía e iniciativa	El desarrollador implementa proactivamente mejoras de accesibilidad más allá de requisitos mínimos	Conformidad WCAG alcanzada superior al nivel requerido (ej: AAA vs AA)
Colaboración	Se comunica claramente sobre problemas de accesibilidad y se colabora con equipo para resolverlos	Reportes de accesibilidad claros y feedback constructivo en code reviews
Postura profesional	Se demuestra compromiso con inclusión y accesibilidad como responsabilidad profesional	Documentación de aprendizaje sobre accesibilidad y participación en mejora continua

Umbral de validación : Promedio >= 10/20

ANEXO VI Glosario y terminología

Este glosario presenta la terminología del referencial. Los términos siguientes deben utilizarse sistemáticamente en la enseñanza y la evaluación.

1. Términos profesionales obligatorios

- Accesibilidad web
- Actualización de pruebas automatizadas
- Análisis de causa raíz
- Análisis de errores JavaScript
- Análisis de logs y trazas de error
- Análisis de requisitos funcionales
- Análisis de requisitos técnicos
- Arquitectura de back-end
- Arquitectura de front-end
- Atributos ARIA
- Auditorías de seguridad
- Autenticación con servicios externos
- Automatización de actualizaciones de dependencias
- Automatización de build y despliegue
- Automatización de pruebas y validaciones
- Buenas prácticas de API
- Cobertura de código
- Code review y análisis de código
- Compatibilidad cross-browser
- Comunicación con stakeholders
- Comunicación de cambios de API
- Comunicación técnica y feedback
- Conexión a bases de datos
- Configuración de cabeceras de seguridad HTTP
- Configuración de CI/CD
- Configuración de entornos de desarrollo
- Configuración de infraestructura
- Context API
- Convenciones de nomenclatura
- CSS responsive y media queries
- Debugging en navegador
- Debugging en servidor
- Debugging y resolución de defectos
- Depuración de llamadas a API
- Desarrollo de componentes reutilizables
- Desarrollo de interfaces HTML/CSS
- Desarrollo Web Full Stack
- Despliegue en plataformas cloud
- Diseño adaptativo
- Diseño de APIs REST
- Diseño de arquitectura web
- Docker y containerización

- Documentación de APIs
- Documentación de arquitectura
- Documentación de componentes
- Documentación de configuración
- Enrutamiento de aplicaciones
- Especificaciones técnicas
- Estrategias de despliegue
- Estándares de calidad de código
- Evaluación de restricciones técnicas
- Evaluación de tecnologías
- Frameworks CSS
- Frameworks de testing
- Frameworks de UI
- Gestión de autorización y control de acceso
- Gestión de caché y consistencia
- Gestión de compatibilidad hacia atrás
- Gestión de dependencias y versionado
- Gestión de estado
- Gestión de pools de conexiones
- Gestión del historial del navegador
- Herramientas de bundling
- Herramientas de CI/CD
- Herramientas de depuración de servidor
- Herramientas de medición de rendimiento
- Herramientas de testing
- Implementación de caché
- Implementación de sistemas de autenticación
- Integración de APIs externas
- Integridad de datos
- Integridad referencial
- Internacionalización
- Lazy loading y code splitting
- Librerías de traducción
- Localización de contenido
- Lógica de negocio en back-end
- Lógica de negocio en front-end
- Manejo de errores en integraciones
- Mensajes de error de usuario
- Modularidad y testabilidad
- Monitorización y alertas de rendimiento
- Navegación por teclado
- Operaciones CRUD
- Optimización de consultas SQL/NoSQL
- Optimización de experiencia de usuario
- Optimización de re-renderizados
- Optimización de rendimiento de carga
- Optimización de rendimiento del servidor
- Organización de componentes
- ORMs y drivers de base de datos
- OWASP Top 10 y vulnerabilidades web
- Patrones de capas

- Patrones de diseño arquitectónico
- Plataformas cloud
- Prevención de inyección SQL y XSS
- Prevención de problemas de rendimiento
- Principios SOLID
- Pruebas de integración
- Pruebas end-to-end
- Pruebas funcionales
- Pruebas unitarias
- Reporte de defectos
- Resolución de conflictos de versiones
- Rutas protegidas y dinámicas
- Seguridad en aplicaciones web
- Selección de frameworks
- Separación de responsabilidades
- Sincronización de datos cliente-servidor
- Testing de APIs y bases de datos
- Testing en múltiples dispositivos
- Transacciones de base de datos
- Usabilidad e interfaz de usuario
- Validación de datos
- Validación de datos en cliente
- Versionado de APIs
- Viabilidad técnica
- WebSockets y comunicación en tiempo real
- Índices y explain plans

2. Sinónimos normalizados (forma canónica en negrita)

- Context API → **API de contexto**
- Enrutamiento de aplicaciones → **Routers de aplicación**
- Frameworks CSS → **Librerías CSS**
- Frameworks de UI → **Librerías de componentes de interfaz**
- Gestión de estado → **Gestores de estado**
- Herramientas de bundling → **Bundlers de aplicación**
- Herramientas de CI/CD → **Plataformas de integración continua**
- Herramientas de depuración de servidor → **Debuggers de servidor**
- Herramientas de medición de rendimiento → **Herramientas de análisis de rendimiento**
- Herramientas de testing → **Frameworks de testing automatizado**
- Librerías de traducción → **Herramientas de internacionalización**
- Operaciones CRUD → **Operaciones de datos básicas**
- ORMs y drivers de base de datos → **Herramientas de acceso a datos**
- Patrones de capas → **Arquitectura en capas**
- Patrones de diseño arquitectónico → **Patrones arquitectónicos**
- Plataformas cloud → **Servicios de infraestructura en la nube**

3. Glosario de términos

Término	Definición
Accesibilidad web	Conjunto de prácticas y estándares (WCAG) que garantizan que las aplicaciones web sean usables por personas con discapacidades.
Actualización de pruebas automatizadas	Modificación de pruebas existentes para reflejar cambios en código o prevenir recurrencia de defectos.
Análisis de causa raíz	Investigación sistemática para identificar la causa fundamental de un problema, no solo síntomas.
Análisis de errores JavaScript	Identificación y resolución de errores en código JavaScript utilizando stack traces y herramientas de debugging.
Análisis de logs y trazas de error	Examen de registros de servidor para identificar causa raíz de errores y problemas de rendimiento.
Análisis de requisitos funcionales	Proceso de identificación, documentación y validación de las funcionalidades que debe cumplir una solución web desde la perspectiva del usuario final.
Análisis de requisitos técnicos	Evaluación de las restricciones técnicas, limitaciones de infraestructura y viabilidad tecnológica para implementar una solución web.
Arquitectura de back-end	Diseño de la capa de servidor incluyendo servicios, APIs, lógica de negocio, acceso a datos y patrones de separación de responsabilidades.
Arquitectura de front-end	Organización estructural de la capa de presentación incluyendo componentes, gestión de estado, enrutamiento y comunicación con el servidor.
Atributos ARIA	Atributos HTML que proporcionan información semántica adicional a lectores de pantalla y herramientas de accesibilidad.
Auditorías de seguridad	Evaluación sistemática de una aplicación para identificar vulnerabilidades de seguridad.
Autenticación con servicios externos	Implementación de mecanismos seguros (OAuth, API keys, JWT) para autenticarse con servicios externos.
Automatización de actualizaciones de dependencias	Uso de herramientas para actualizar automáticamente dependencias manteniendo compatibilidad.
Automatización de build y despliegue	Proceso automatizado de compilar, empaquetar y desplegar aplicaciones a servidores.
Automatización de pruebas y validaciones	Ejecución automática de pruebas y validaciones de código en pipelines de CI/CD.
Buenas prácticas de API	Conjunto de convenciones y patrones para diseñar APIs robustas, seguras y fáciles de usar.
Cobertura de código	Métrica que indica el porcentaje de código que es ejecutado por pruebas automatizadas.
Code review y análisis de código	Evaluación sistemática de código escrito por otros desarrolladores para verificar calidad, seguridad y adherencia a estándares.

Término	Definición
Compatibilidad cross-browser	Garantía de que la aplicación funciona correctamente en diferentes navegadores web.
Comunicación con stakeholders	Interacción efectiva con clientes, usuarios y otros interesados para entender y validar requisitos.
Comunicación de cambios de API	Documentación y notificación clara a usuarios de API sobre cambios, deprecaciones y migraciones.
Comunicación técnica y feedback	Capacidad de explicar conceptos técnicos complejos y proporcionar retroalimentación constructiva.
Conexión a bases de datos	Establecimiento de comunicación entre la aplicación y el servidor de base de datos utilizando drivers u ORMs.
Configuración de cabeceras de seguridad HTTP	Establecimiento de headers HTTP (CSP, X-Frame-Options, HSTS) que protegen contra ataques web comunes.
Configuración de CI/CD	Establecimiento de pipelines automatizados que ejecutan pruebas y despliegues en cada cambio de código.
Configuración de entornos de desarrollo	Establecimiento de herramientas, variables de entorno y dependencias necesarias para desarrollar localmente.
Configuración de infraestructura	Establecimiento de servidores, bases de datos, redes y otros recursos necesarios para ejecutar la aplicación.
Context API	Mecanismo de React para pasar datos a través de la jerarquía de componentes sin prop drilling.
Convenciones de nomenclatura	Reglas consistentes para nombrar variables, funciones, componentes y archivos en el proyecto.
CSS responsive y media queries	Técnicas CSS que permiten que estilos se adapten a diferentes tamaños de pantalla.
Debugging en navegador	Uso de DevTools del navegador para inspeccionar, depurar y analizar código JavaScript y red.
Debugging en servidor	Uso de herramientas de depuración en el servidor para inspeccionar ejecución de código y estado.
Debugging y resolución de defectos	Proceso de identificar, analizar y corregir errores en código.
Depuración de llamadas a API	Inspección de solicitudes y respuestas HTTP para identificar problemas en comunicación cliente-servidor.
Desarrollo de componentes reutilizables	Creación de unidades de interfaz independientes, documentadas y testables que pueden ser utilizadas en múltiples contextos.
Desarrollo de interfaces HTML/CSS	Implementación de la estructura y estilos visuales de la interfaz de usuario utilizando HTML y CSS.
Desarrollo Web Full Stack	Conjunto de competencias y actividades para diseñar, desarrollar, desplegar y mantener aplicaciones web completas que integran componentes de front-end, back-end, bases de datos e infraestructura.

Término	Definición
Despliegue en plataformas cloud	Publicación de aplicaciones en servicios cloud como AWS, Azure, GCP, Vercel o Heroku.
Diseño adaptativo	Técnica de diseño que permite que una interfaz se adapte automáticamente a diferentes tamaños de pantalla y dispositivos.
Diseño de APIs REST	Creación de interfaces de programación siguiendo principios REST incluyendo recursos, verbos HTTP y códigos de estado.
Diseño de arquitectura web	Definición de la estructura general de una aplicación web incluyendo componentes, capas, patrones de comunicación y flujos de datos.
Docker y containerización	Tecnología para empaquetar aplicaciones con sus dependencias en contenedores aislados y portables.
Documentación de APIs	Especificación clara de endpoints, parámetros, respuestas y ejemplos de uso de una API, típicamente usando Swagger/OpenAPI.
Documentación de arquitectura	Especificación clara de la estructura de la aplicación incluyendo diagramas, componentes y flujos.
Documentación de componentes	Especificación clara de propósito, props, ejemplos de uso y comportamiento de componentes reutilizables.
Documentación de configuración	Especificación clara de pasos y requisitos para configurar entornos de desarrollo y producción.
Enrutamiento de aplicaciones	Implementación de navegación entre vistas en aplicaciones de una sola página usando librerías como React Router o Vue Router.
Especificaciones técnicas	Documentación detallada de requisitos técnicos, restricciones y decisiones de diseño.
Estrategias de despliegue	Enfoques para desplegar nuevas versiones (blue-green, canary, rolling) minimizando downtime.
Estándares de calidad de código	Conjunto de reglas y convenciones que definen cómo debe escribirse el código en un proyecto.
Evaluación de restricciones técnicas	Análisis de limitaciones de infraestructura, tecnología y recursos que afectan la solución.
Evaluación de tecnologías	Análisis comparativo de opciones tecnológicas considerando características, comunidad, compatibilidad, rendimiento y costo.
Frameworks CSS	Librerías predefinidas de estilos y componentes CSS como Bootstrap o Tailwind que aceleran el desarrollo de interfaces.
Frameworks de testing	Librerías como Jest, Vitest o Mocha que facilitan la escritura y ejecución de pruebas automatizadas.
Gestión de autorización y control de acceso	Implementación de sistemas que controlan qué recursos puede acceder cada usuario basándose en roles y permisos.

Término	Definición
Gestión de caché y consistencia	Almacenamiento temporal de datos en cliente o servidor manteniendo coherencia con la fuente de verdad.
Gestión de compatibilidad hacia atrás	Garantía de que cambios en APIs o código no rompen aplicaciones o clientes existentes.
Gestión de dependencias y versionado	Administración de librerías externas incluyendo actualización, versionado y resolución de conflictos.
Gestión de estado	Administración centralizada del estado de la aplicación en el cliente utilizando patrones como Redux, Zustand o Pinia.
Gestión de pools de conexiones	Administración eficiente de un conjunto de conexiones reutilizables a la base de datos para optimizar recursos.
Gestión del historial del navegador	Control de la pila de navegación permitiendo que botones atrás/adelante funcionen correctamente en aplicaciones de una sola página.
Herramientas de bundling	Utilidades como Webpack o Vite que empaquetan y optimizan código y recursos para producción.
Herramientas de CI/CD	Plataformas como GitHub Actions, GitLab CI o Jenkins que automatizan integración y despliegue continuo.
Herramientas de medición de rendimiento	Utilidades como Lighthouse, WebPageTest o DevTools que miden y analizan métricas de rendimiento.
Herramientas de testing	Utilidades como Supertest, Cypress o Playwright para automatizar pruebas de integración y end-to-end.
Implementación de caché	Almacenamiento temporal de datos frecuentemente accedidos en memoria (Redis, Memcached) para mejorar rendimiento.
Implementación de sistemas de autenticación	Desarrollo de mecanismos seguros para verificar la identidad de usuarios (JWT, OAuth, sesiones).
Integración de APIs externas	Proceso de conectar una aplicación con servicios externos mediante sus APIs para acceder a funcionalidades o datos.
Integridad de datos	Garantía de que los datos en la base de datos son precisos, consistentes y no se corrompen durante operaciones.
Integridad referencial	Garantía de que las relaciones entre tablas en una base de datos se mantienen consistentes.
Internacionalización	Proceso de preparar una aplicación para soportar múltiples idiomas y regiones.
Lazy loading y code splitting	Técnicas que cargan componentes y código bajo demanda en lugar de todo al inicio, mejorando rendimiento inicial.
Librerías de traducción	Herramientas como i18next o vue-i18n que facilitan la implementación de soporte multiidioma.
Localización de contenido	Adaptación de contenido, formatos y estilos a idiomas y regiones específicas.

Término	Definición
Lógica de negocio en back-end	Implementación de reglas de negocio en el servidor garantizando integridad, seguridad y consistencia de datos.
Lógica de negocio en front-end	Implementación de reglas de negocio en el cliente incluyendo validaciones, cálculos y transformaciones de datos.
Manejo de errores en integraciones	Implementación de mecanismos robustos para manejar fallos en integraciones con servicios externos.
Mensajes de error de usuario	Comunicación clara y útil de errores a usuarios sin exponer detalles técnicos sensibles.
Modularidad y testabilidad	Principios de diseño que permiten que componentes sean independientes, fáciles de probar y mantenibles.
Monitorización y alertas de rendimiento	Seguimiento continuo de métricas de rendimiento (CPU, memoria, latencia) con notificaciones de anomalías.
Operaciones CRUD	Conjunto de operaciones básicas sobre datos: Create (crear), Read (leer), Update (actualizar) y Delete (eliminar).
Optimización de consultas SQL/NoSQL	Mejora del rendimiento de consultas a bases de datos mediante reescritura, índices y análisis de planes de ejecución.
Optimización de experiencia de usuario	Mejora de la interacción del usuario con la aplicación mediante feedback visual, rendimiento y usabilidad.
Optimización de re-renderizados	Técnicas para evitar actualizaciones innecesarias de componentes mejorando rendimiento mediante memoización.
Optimización de rendimiento de carga	Conjunto de técnicas para reducir el tiempo que tarda una aplicación en cargar inicialmente.
Optimización de rendimiento del servidor	Mejora de la velocidad y eficiencia del servidor mediante optimización de código, caché y recursos.
Organización de componentes	Estructura y disposición de componentes en la aplicación facilitando mantenimiento y reutilización.
ORMs y drivers de base de datos	Herramientas que facilitan la interacción con bases de datos mapeando objetos a registros (ORMs) o proporcionando acceso directo (drivers).
OWASP Top 10 y vulnerabilidades web	Lista de las diez vulnerabilidades web más críticas y técnicas para prevenirlas.
Patrones de capas	Arquitecturas que organizan código en capas (presentación, negocio, datos) con responsabilidades claras.
Patrones de diseño arquitectónico	Soluciones reutilizables para problemas comunes de arquitectura como MVC, MVVM, microservicios o arquitectura por capas.
Plataformas cloud	Servicios de infraestructura en la nube que proporcionan servidores, bases de datos y herramientas de despliegue.

Término	Definición
Prevención de inyección SQL y XSS	Técnicas de sanitización y parametrización que previenen ataques de inyección SQL y cross-site scripting.
Prevención de problemas de rendimiento	Identificación proactiva y resolución de cuellos de botella que afectan rendimiento.
Principios SOLID	Conjunto de cinco principios de diseño (responsabilidad única, abierto/cerrado, sustitución de Liskov, segregación de interfaz, inversión de dependencia) que mejoran la calidad del código.
Pruebas de integración	Pruebas que verifican que múltiples componentes funcionan correctamente juntos.
Pruebas end-to-end	Pruebas que verifican flujos completos de usuario desde inicio hasta fin en un entorno similar a producción.
Pruebas funcionales	Pruebas que verifican que la aplicación cumple con requisitos funcionales desde la perspectiva del usuario.
Pruebas unitarias	Pruebas que verifican el comportamiento correcto de unidades individuales de código (funciones, métodos).
Reporte de defectos	Documentación detallada de problemas encontrados incluyendo pasos de reproducción, logs y evidencia visual.
Resolución de conflictos de versiones	Proceso de resolver incompatibilidades cuando múltiples dependencias requieren versiones diferentes de una librería.
Rutas protegidas y dinámicas	Rutas que requieren autenticación/autorización o se generan dinámicamente basadas en datos.
Seguridad en aplicaciones web	Conjunto de prácticas y medidas para proteger aplicaciones web contra ataques y vulnerabilidades.
Selección de frameworks	Proceso de elegir frameworks específicos para cada capa de la aplicación basándose en requisitos funcionales y restricciones técnicas.
Separación de responsabilidades	Principio de diseño que asigna responsabilidades específicas a diferentes componentes o capas.
Sincronización de datos cliente-servidor	Proceso de mantener los datos en el cliente y servidor actualizados y consistentes en tiempo real.
Testing de APIs y bases de datos	Pruebas que verifican el comportamiento correcto de APIs y operaciones de base de datos.
Testing en múltiples dispositivos	Verificación de que la aplicación funciona correctamente en diferentes dispositivos (móviles, tablets, desktops).
Transacciones de base de datos	Unidades de trabajo que se ejecutan completamente o no se ejecutan en absoluto, garantizando consistencia de datos.
Usabilidad e interfaz de usuario	Principios y prácticas para crear interfaces intuitivas, eficientes y agradables de usar.

Término	Definición
Validación de datos	Verificación de que los datos de entrada cumplen con reglas de formato, tipo y rango esperadas.
Validación de datos en cliente	Verificación de la integridad y formato de datos en el navegador antes de enviarlos al servidor.
Versionado de APIs	Estrategia de mantener múltiples versiones de una API simultáneamente para compatibilidad hacia atrás.
Viabilidad técnica	Evaluación de si una solución propuesta es técnicamente posible con recursos y tecnologías disponibles.
WebSockets y comunicación en tiempo real	Protocolo y técnicas que permiten comunicación bidireccional en tiempo real entre cliente y servidor.
Índices y explain plans	Estructuras de base de datos que aceleran búsquedas y herramientas que analizan cómo se ejecutan las consultas.