1.4 PLANIFICACION DE APLICACIONES WEB

1. PLANIFICACIÓN Y DEFINICIÓN DE UNA APLICACIÓN WEB

Esta es la primera y más importante fase del desarrollo de una aplicación web. Aquí se establecen los objetivos, requisitos y estrategias antes de escribir una sola línea de código.

¿Por qué es importante la planificación?Una planificación sólida evita problemas futuros, reduce costos y garantiza que la aplicación cumpla con las expectativas de los usuarios y del negocio.

Pasos en la planificación y definición:

- Definir el propósito del proyecto
  - ✓ ¿Cuál es el problema que la aplicación resolverá?
  - ✓ ¿Qué beneficios ofrecerá a los usuarios?
  - ✓ ¿Cuál es el objetivo final del negocio?
- Identificar al público objetivo
  - ✓ ¿Quiénes serán los usuarios principales?
  - ✓ ¿Cuáles son sus necesidades y expectativas?
  - ✓ ¿En qué dispositivos accederán a la aplicación?
- Definir las funcionalidades principales. Aquí se establecen las características esenciales de la aplicación.
- Crear un boceto o wireframe inicial. Antes de pasar al desarrollo, se recomienda diseñar un prototipo visual que muestre la estructura de la aplicación. Herramientas útiles para prototipado.
  - ✓ Figma, Adobe XD, Sketch, Balsamiq.
- Elegir el modelo de negocio
  - ✓ Freemium.
  - ✓ Suscripción mensual o anual
  - ✓ Pago único por uso
  - ✓ Publicidad integrada
- Establecer un cronograma de desarrollo. Se define un plan de trabajo con plazos y entregas.

2. ELECCIÓN DE TECNOLOGÍAS PARA UNA APLICACIÓN WEB

Una vez que la aplicación ha sido planificada y definida, es momento de elegir las tecnologías adecuadas para su desarrollo. La selección correcta garantiza eficiencia, escalabilidad y buen rendimiento.

- Factores Claves al Elegir Tecnologías
  - ➢ Escalabilidad: ¿Puede crecer sin afectar el rendimiento?
  - ➢ Facilidad de desarrollo: ¿La tecnología es fácil de aprender y usar?
  - ➢ Comunidad y soporte: ¿Tiene una comunidad activa y documentación clara?
  - ➢ Seguridad: ¿Ofrece herramientas para proteger los datos del usuario?
  - ➢ Compatibilidad: ¿Funciona bien con otras tecnologías necesarias para el proyecto?
- Tecnologías Esenciales en el Desarrollo Web
  - ➢ Frontend (Interfaz de Usuario). El frontend es la parte visual con la que los usuarios interactúan.
    - HTML, CSS y JavaScript(Lenguajes base del frontend).

- React.js (Popular y flexible, creado por Facebook).
- Vue.js (Ligero y fácil de aprender).
- Angular (Escalable, desarrollado por Google).
- Tailwind CSS (Framework de CSS moderno y minimalista).
- Bootstrap (Componentes prediseñados para diseño responsivo).

➢ Backend (Lógica y Procesamiento del Servidor). El backend maneja la lógica de negocio, bases de datos y autenticación.

- Node.js con Express.js (Usa JavaScript en el backend).
- Python con Django o Flask (Rápido y con buenas prácticas).
- PHP con Laravel (Usado en sitios como WordPress).
- Ruby on Rails (Enfocado en rapidez de desarrollo).
- JSON Web Tokens (JWT) para manejar sesiones.
- OAuth 2.0 para integraciones con Google, Facebook, etc.

➢ Bases de Datos. Las bases de datos almacenan la información de la aplicación.

- MySQL (Popular, usado en WordPress).
- PostgreSQL (Robusto y escalable).
- MongoDB (Flexible y basado en documentos).
- Firebase Firestore (Usado en apps en tiempo real).

➢ Servidores y Hosting Para que la aplicación esté en línea, necesita un servidor donde alojarse.

- Vercel / Netlify (Ideal para frontend con React, Vue o Angular).
- Heroku (Plataforma sencilla para aplicaciones backend).
- AWS (Amazon Web Services) (Para aplicaciones grandes y escalables).
- Firebase Hosting (Ideal para aplicaciones en tiempo real).

➢ Herramientas Adicionales

- Git y GitHub/GitLab/Bitbucket para manejar cambios en el código.
- Stripe / PayPal (Pagos en línea).
- Google Maps API (Ubicación y mapas).
- Twilio (Mensajería y llamadas).

3. DESARROLLO DEL FRONTEND

El frontend es la parte visual e interactiva de una aplicación web. Es lo que los usuarios ven y con lo que interactúan. Su desarrollo implica crear interfaces atractivas, responsivas y funcionales.

Pasos para el Desarrollo del Frontend

♥ Estructura del Proyecto. Antes de empezar a escribir código, se organiza el proyecto. Pasos clave:

➢ Definir la estructura de carpetas y archivos.
➢ Elegir un framework o librería (React, Vue, Angular, etc.).
➢ Configurar herramientas esenciales como Webpack o Vite.

♥ Creación de la Interfaz de Usuario (UI/UX). Objetivos:

- Crear una interfaz intuitiva y atractiva.
- Asegurar accesibilidad y experiencia de usuario óptima.
- Diseñar un layout responsivo para diferentes dispositivos.
- Herramientas y tecnologías para diseño:
  - CSS Frameworks: Tailwind CSS, Bootstrap, Material UI.
  - Preprocesadores: Sass, LESS.

```
/mi-proyecto
├── public/
├── src/
│   ├── components/   # Componentes reutilizables
│   ├── pages/        # Vistas principales
│   ├── styles/       # Archivos CSS
│   ├── App.js        # Componente principal
│   ├── index.js      # Punto de entrada
├── package.json      # Dependencias del proyecto
├── .gitignore        # Archivos a ignorar en Git
├── README.md         # Documentación del proyecto
```

- Programación y Creación de Componentes. Los frameworks modernos usan componentes, que son piezas de interfaz reutilizables.
- Gestión del Estado y Datos. En aplicaciones dinámicas, se necesita manejar datos de forma eficiente. Herramientas para gestión de estado:
  - React Context / Redux (Para manejar datos globales).
  - Vuex / Pinia en Vue.js.

```
function Tarjeta({ titulo, descripcion }) {
  return (
    <div className="border p-4 rounded-lg shadow-md">
      <h2 className="text-xl font-bold">{titulo}</h2>
      <p>{descripcion}</p>
    </div>
  );
}
```

- Conexión con el Backend (APIs). El frontend se comunica con el backend mediante APIs REST o GraphQL. Ejemplo de llamada a una API con Fetch:

```
fetch("https://api.ejemplo.com/datos")
  .then(response => response.json())
  .then(data => console.log(data));
```

- Ejemplo con Axios (librería recomendada para peticiones HTTP):

```
import axios from 'axios';

axios.get("https://api.ejemplo.com/datos")
  .then(response => console.log(response.data))
  .catch(error => console.error("Error:", error));
```

- Optimización del Rendimiento. Buenas prácticas para mejorar el rendimiento:
  - Minificar archivos CSS y JS.
  - Cargar imágenes en formatos optimizados (WebP).
  - Usar carga diferida (Lazy Loading).
  - Implementar almacenamiento en caché.

4. DESARROLLO DEL BACKEND

El backend es la parte de la aplicación que gestiona la lógica de negocio, la base de datos y la comunicación con el frontend. Se encarga de procesar solicitudes, almacenar datos y garantizar la seguridad del sistema.

Pasos para el Desarrollo del Backend

- Elección del Lenguaje y Framework Se debe elegir una tecnología que se adapte a las necesidades del proyecto. Lenguajes y frameworks populares para backend:
  - JavaScript (Node.js + Express.js, NestJS)
  - Python (Django, Flask, FastAPI)
  - PHP (Laravel, Symfony)
  - Ruby (Ruby on Rails)
  - Java (Spring Boot)

- Diseño de la Arquitectura del Backend. Existen diferentes enfoques para organizar la lógica del backend. Tipos de Arquitectura:
  - Monolítica: Todo el código en una sola aplicación.
  - Microservicios: La aplicación se divide en pequeños servicios independientes.
  - Serverless: Funciones que se ejecutan en la nube sin administrar servidores.
- Cada servicio tiene su propia base de datos y lógica de negocio. Ejemplo de arquitectura de backend basada en microservicios:
  - Servicio de autenticación.
  - Servicio de gestión de productos.
  - Servicio de pagos.
- Creación de la API (REST o GraphQL). El backend se comunica con el frontend a través de APIs. Diferencias clave:
  - REST API: Estructura tradicional basada en HTTP (GET, POST, PUT, DELETE).
  - GraphQL: Permite consultas flexibles y optimiza la transferencia de datos.
- Manejo de Base de Datos. El backend necesita una base de datos para almacenar y recuperar información. Tipos de bases de datos:
  - Relacionales (SQL) → MySQL, PostgreSQL, MariaDB.
  - NoSQL → MongoDB, Firebase, Cassandra.
- Seguridad y Autenticación. Buenas prácticas de seguridad en el backend:
  - Usar JSON Web Tokens (JWT) para autenticación.
  - Cifrar contraseñas con bcrypt
  - Implementar CORS para controlar el acceso desde otros dominios.
  - Usar HTTPS y proteger contra ataques de inyección SQL y XSS.
- Optimización y Escalabilidad. Técnicas para mejorar el rendimiento:
  - Implementar caché con Redis o Memcached.
  - Optimizar consultas a la base de datos.
  - Usar balanceadores de carga para distribuir tráfico.
  - Desplegar la aplicación en servicios escalables como AWS o Firebase.

5. PRUEBAS EN EL DESARROLLO WEB

Las pruebas garantizan que la aplicación web funcione correctamente, sea segura y ofrezca una buena experiencia al usuario. Este paso es crucial antes del despliegue para detectar errores y optimizar el rendimiento.

Tipos de Pruebas en Aplicaciones Web:

- Pruebas Unitarias (Unit Testing)
  - Verifican el funcionamiento de partes individuales del código (funciones, métodos, componentes).
  - Se aseguran de que cada módulo de la aplicación haga lo que debe hacer.
- Herramientas:
  - Jest (JavaScript/React/Node.js).
  - Mocha + Chai (Node.js).
  - PyTest (Python).

- Pruebas de Integración (Integration Testing)
  - Evalúan cómo interactúan diferentes módulos del sistema.
  - Se prueban conexiones entre frontend y backend, llamadas a APIs, bases de datos, etc.
- Pruebas Funcionales
  - Simulan escenarios reales de usuario para comprobar que las funcionalidades cumplen su propósito.
  - Se centran en validar la lógica de negocio y los requerimientos del cliente.
  - Herramientas:
    - Cypress (Para pruebas de interfaz web).
    - Selenium (Automatización de navegadores).
- Pruebas de UI/UX (Pruebas de Interfaz y Experiencia de Usuario).
  - Evalúan el diseño, navegación y usabilidad.
  - Se hacen pruebas manuales o con herramientas de automatización.
  - Herramientas:
    - Lighthouse (Verifica accesibilidad y rendimiento).
    - Hotjar (Analiza interacciones de usuarios reales).
- Pruebas de Seguridad
  - Detectan vulnerabilidades en la aplicación.
  - Evitan ataques como inyección SQL, XSS, CSRF, etc.
  - Herramientas:
    - OWASP ZAP (Escaneo de vulnerabilidades).
    - Burp Suite (Pruebas de seguridad web).
- Pruebas de Carga y Rendimiento
  - Evalúan cómo responde la aplicación ante múltiples usuarios simultáneos.
  - Detectan cuellos de botella y optimizan tiempos de respuesta.
  - Herramientas:
    - Apache JMeter (Simulación de usuarios concurrentes).
    - K6 (Pruebas de carga en backend).

6. DESPLIEGUE DE LA APLICACIÓN WEB

El despliegue es el proceso de poner en funcionamiento la aplicación web en un servidor o plataforma en la nube, para que los usuarios puedan acceder a ella.

Pasos para el Despliegue de una Aplicación Web

- Elección del Hosting y Servidor. Dependiendo del tipo de aplicación, se puede elegir entre diferentes opciones: Tipos de Hosting:
  - Servidor Compartido: Económico, pero con recursos limitados. Ejemplo: Bluehost, Hostinger.
  - VPS (Servidor Privado Virtual): Más control y rendimiento. Ejemplo: DigitalOcean, Linode.
  - Cloud Computing: Escalabilidad automática. Ejemplo: AWS, Google Cloud, Azure.
  - Plataformas Serverless: Ideal para microservicios y APIs. Ejemplo: Vercel, Netlify, Firebase.
  - Ejemplo de elección:
    - Frontend React/Vue: Vercel, Netlify, Firebase Hosting

- Backend Node.js/Python: Render, Heroku, AWS EC2
- Base de Datos: MongoDB Atlas (NoSQL), Amazon RDS (SQL)

♥ Configuración del Servidor. Si el despliegue es en un VPS o servidor dedicado, se deben configurar ciertos servicios.

➢ Configuraciones esenciales:

- Sistema Operativo: Ubuntu, Debian, CentOS.
- Servidor Web: Nginx o Apache.
- Base de Datos: PostgreSQL, MySQL, MongoDB.
- Control de versiones: Git para actualizar código.

♥ Despliegue del Frontend. Si el frontend es una SPA (React, Vue, Angular), debe ser compilado y subido a un servidor estático.

♥ Despliegue del Backend. Si el backend está en Node.js, se puede desplegar en Heroku, Render, AWS EC2 o DigitalOcean.

♥ Configuración del Dominio y HTTPS. Para que la aplicación tenga un dominio propio y use HTTPS, se debe configurar un certificado SSL.

♥ Implementación de CI/CD (Integración y Entrega Continua). El uso de CI/CD permite desplegar actualizaciones automáticamente cuando se hacen cambios en el código.

➢ Herramientas de CI/CD:

- GitHub Actions
- GitLab CI/CD
- Jenkins