Resúmenes Diapositivas SWAD por Orden

1. Arquitecturas Aplicaciones Internet

Aplicaciones Distribuidas Cliente-Servidor: La respuesta del servidor depende únicamente del request. No depende de request anteriores.

Arquitectura Aplicación WEB: El usuario pide datos al frontend. El frontend hace el request al backend, que incluye la lógica de la APP (Servidor WEB con base de datos y sistema de archivos). El backend hace el response al frontend. El frontend muestra el resultado al usuario.

Multiple Page Aplications (MPA) Lifecycle: Request inicial. El servidor devuelve HTML. Petición POST. Se recarga el HTML.

Single Page Aplications (SPA) Lifecycle: Request inicial. El servidor devuelve HTML. Petición POST. El servidor devuelve un JSON.

Ventajas de las SPAs:

1. Velocidad y capacidad de respuesta.
2. Experiencia de usuario mejorada.
3. Menor carga del servidor.
4. Compatibilidad multiplataforma.
5. Funcionalidad sin conexión.

Desventajas de las SPAs:

1. Desafíos de SEO.
2. Tiempo de carga inicial.
3. Historial y marcadores del navegador.
4. Desarrollo complejo.
5. Problemas de seguridad.

Client Side Rendering (CSR): El contenido de la página se genera completamente en el navegador del cliente mediante JavaScript, cargando una página inicial vacía y completándola dinámicamente.

Pros:

- Es rápido para el servidor entregar una página en blanco.

- Tiene soporte para SPAs.

- Es súper rápido una vez todo el Javascript ha sido cargado.

Contras:

- Entregas una página en blanco al usuario.

- Al entregar una página en vacía, no hay posibilidad de ser indexado por motores de búsqueda.

- Peligroso cuando tienes que cargar mucha información, provocando que el usuario solo vea una página blanca por más tiempo.

- Conexiones lentas pueden pueden provocar rebote de los usuarios.

Server Side Rendering (SSR): El contenido de la página se genera en el servidor en cada solicitud y se envía al cliente ya renderizado, lo que mejora el SEO y reduce el tiempo de carga inicial.

Pros:

- Es bueno para páginas que necesitan tener un buen SEO y que quieren ser indexadas por motores de búsqueda

- El contenido entregado por el servidor es inmediato.

- Funciona perfectamente en conexiones lentas.

- No hay peticiones a APIs en el lado del cliente.

Contras:

- Es más pesado para el servidor

- No puede ser cacheado debido a las peticiones que hace, por lo que se asume que esta información necesita ser nueva cada vez que el cliente quiere entrar a la página

- No funciona bien con librerías que necesitan acceder a la propiedad document.

- Genera problemas de hidratación si se accede a variables de localStorage o información que no puede ser obtenida en el lado del servidor.

Static Side Generation (SSG): Las páginas se renderizan en el momento de la compilación y se almacenan como archivos estáticos para ser servidos rápidamente, ideal para contenido que no cambia frecuentemente.

Pros:

- Las páginas son muy rápidas.

- Se genera una vez y luego el CDN la entrega inmediatamente al usuario.

- No tenemos peticiones por el lado del cliente.

- Perfecto para páginas que requieren un buen SEO.

- No tiene estado de carga, ya que nada necesita ser cargado, todo fue previamente cargado en el build.

- No hay necesidad de tener un servidor.

Contras:

- Tiene un build time más largo que el resto de los modelos.

- No funciona bien con librerías que necesitan acceder a la propiedad document.

- Genera problemas de hidratación si se accede a variables de localStorage o información que no puede ser obtenida previa a la generación de la página.

Incremental Static Generation (ISG): Una extensión de SSG que permite generar o actualizar páginas estáticas de manera incremental tras la compilación, combinando velocidad de carga y dinamismo.

Pros:

- Nos entrega la posibilidad de evitar recompilaciones si la información cambia seguido, pero no tanto.

- Si tenemos una gran cantidad de páginas no queremos tener que volver a compilar todas estas páginas cada vez que surga un cambio, esto lo soluciona.

- Tenemos todas las ventajas previamente mencionadas tanto por SSR y SSG sumando estas nuevas que acabo de mencionar.

- Sumamente fácil de agregar con solo una línea de código.

2. JavaScript ES6 y Programación Asíncrona

Síncrono: El cliente hace la petición y no continúa hasta que recibe la respuesta del servidor. En node.js se puede usar la librería unlinkSynk.

Asíncrono: El cliente hace la petición y continúa. Cuando recibe la respuesta, la emplea. En node.js se puede usar la librería unlink, con funciones callback. Callback Hell puede ser un problema si se anidan muchos Callback.

Solución: Promises de ES6 encadenables. Promises.resolve().then().then().catch();

Otra opción: Métodos async empleando await en las peticiones.

3. Bases de Datos NO-SQL y Object Relational Mappers

Las Aplicaciones WEB se hacen para el CRUD: Create (Insert), Read (Select), Update y Delete. La mayoría de las bases de datos están en la nube.

Bases de datos SQL: Relacionales. Usan ORMs (Object Relational Mapping). Un Object Relational Mapping (ORM) es una técnica de programación que permite interactuar con bases de datos relacionales utilizando objetos en lugar de escribir consultas SQL directamente. Convierte tablas en clases y filas en objetos, facilitando la manipulación de datos con código en un lenguaje de programación.

Bases de datos NoSQL: Key-Value, Column-Family, Graph y Document. Objetivo las 3 Vs: Velocidad, Variedad y Volumen.

La diferencia principal entre bases de datos SQL y NoSQL radica en cómo estructuran y manejan los datos:

- **SQL**: Son bases de datos relacionales que usan tablas con un esquema fijo y estructurado (filas y columnas) y se acceden mediante lenguaje SQL. Son ideales para datos estructurados y transacciones complejas.
- **NoSQ**L: Son bases de datos no relacionales que almacenan datos de forma flexible (documentos, clave-valor, grafos, etc.) y no requieren un esquema fijo. Son adecuadas para datos no estructurados y aplicaciones escalables.

Neo4j: Base de datos NoSQL especializada en grafos.

Elasticsearch es una herramienta rápida y escalable para búsquedas y análisis de datos en tiempo real, ideal para manejar grandes volúmenes de información mediante indexación distribuida. Es especialmente útil en análisis de logs, monitoreo y sistemas que requieren búsquedas complejas.

MongoDB: Base de datos orientada a documentos BSON, formada por colecciones que son conjuntos de documentos. Los documentos tipo BSON (Binary JSON) son una representación binaria de JSON diseñada para ser más eficiente en almacenamiento y procesamiento, ampliamente utilizada en bases de datos como MongoDB. _id del Objeto autogenerado.

Mongoose: Driver para mongdb con esquema. Para validar con tipos, restricciones y valores por defecto.

4. MVC y Plantillas Framework Backends

- **M (Model)**: Interacciona con la Base de Datos (CRUD). Lógica relacionada con los datos. Comunica con el Controlador. Clases, validaciones, etc, con ORMs.

- **V (View)**: Interacción con el usuario. HTML, CSS y JavaScript. Motor de plantillas (template engine) Un template engine es una herramienta que permite generar páginas dinámicas combinando plantillas con datos, reemplazando marcadores o variables en el código HTML con contenido proporcionado en tiempo de ejecución. Comunica con el controlador.

- **C (Controller)**: Recibe URLs (routing). Procesa requests (CRUD). Interacciona con el modelo. Rellena y envía plantillas.

Express JS: Framework más utilizado en node. Es un framework minimalista y flexible para Node.js que facilita la creación de aplicaciones web y APIs, proporcionando herramientas y funcionalidades como enrutamiento, manejo de solicitudes HTTP y middleware para simplificar el desarrollo del backend.

El middleware de Express son funciones que tienen acceso a la solicitud (request), la respuesta (response) y el siguiente middleware en el ciclo de vida de la solicitud. Se usan para realizar tareas como procesar datos de entrada, autenticar usuarios, manejar errores, y modificar las respuestas antes de que lleguen al cliente, todo de manera modular y encadenada.

El uso de middleware en Express permite organizar y modularizar el código, facilitando tareas como la validación de datos, la autenticación, el manejo de errores y la gestión de solicitudes, todo de manera eficiente y reutilizable. Además, permite modificar las solicitudes y respuestas de forma flexible sin necesidad de alterar la lógica principal de las rutas, mejorando la mantenibilidad y escalabilidad de las aplicaciones.

Jinja templates es un motor de plantillas para Python que permite generar dinámicamente contenido HTML, XML u otros formatos. Utiliza una sintaxis similar a Python para insertar variables, realizar bucles y condicionales dentro de las plantillas, lo que facilita la creación de

páginas web dinámicas y personalizadas en aplicaciones web, como aquellas basadas en frameworks como Flask o Django.

Los **CSS frameworks**, como **Bootstrap**, son bibliotecas predefinidas que ofrecen un conjunto de estilos y componentes listos para usar, como botones, formularios, y diseño de rejillas. Facilitan el desarrollo web al proporcionar una estructura coherente y responsiva, permitiendo a los desarrolladores crear interfaces atractivas y funcionales de manera más rápida y sin tener que escribir CSS desde cero.

5. Protocolos HTTP, WebSockets

En un 99% de los casos, las APPs están distribuidas en Cliente-Servidor. Comunicadas con HTTP.

- **Cliente:** Proactivo, inicia la comunicación. Manda request. Se encarga de interaccionar con el usuario (interfaz).

- **Servidor:** Reactivo, espera peticiones. Responde response. Se encarga de la lógica de la APP y de las interacciones con la BD.

- **Frontend:** Es la parte visible de una aplicación web o móvil, encargada de la interfaz de usuario y la interacción, desarrollada con HTML, CSS y JavaScript.

- **Backend:** Es la parte del servidor que gestiona la lógica de la aplicación, el almacenamiento de datos y la comunicación con el frontend, utilizando lenguajes como Python, Node.js o Java.

- **AJAX (Asynchronous JavaScript and XML):** Es una técnica que permite realizar solicitudes HTTP de manera asíncrona sin recargar la página, lo que permite actualizar partes específicas de la interfaz sin interrumpir la experiencia del usuario.

- **Fetch:** Es una API moderna de JavaScript que reemplaza a AJAX para hacer solicitudes asíncronas. Utiliza Promesas, lo que hace el código más limpio y fácil de manejar para las respuestas y errores de las solicitudes HTTP.

HTTP es un protocolo sin estado porque cada solicitud que se realiza entre el cliente y el servidor es independiente, es decir, el servidor no guarda información sobre las solicitudes anteriores. Cada **request** es tratado de manera aislada, sin que el servidor dependa de ninguna interacción previa para generar una respuesta, lo que lo hace más simple y escalable pero también requiere mecanismos adicionales (como cookies o sesiones) para gestionar el estado cuando es necesario.

- **HTTP (HyperText Transfer Protocol):** Es el protocolo de comunicación utilizado en la web para intercambiar datos entre el cliente (navegador) y el servidor. Funciona de manera sin estado, lo que significa que cada solicitud es independiente y no depende de solicitudes anteriores.

- **Requests (Solicitudes):** Son mensajes enviados por el cliente (como un navegador) al servidor para solicitar recursos, datos o servicios. Contienen información como el método HTTP, encabezados y, a veces, un cuerpo de datos.
- **Responses (Respuestas):** Son los mensajes que el servidor envía al cliente en respuesta a una solicitud HTTP, que incluyen el código de estado (como 200 OK), encabezados y el cuerpo con el recurso solicitado.
- **GET:** Es un verbo de HTTP utilizado para solicitar recursos del servidor, como páginas web o datos, sin modificar nada en el servidor.
- **PUT:** Es un verbo de HTTP utilizado para actualizar un recurso existente en el servidor o crear uno nuevo si no existe, reemplazando su contenido actual.
- **POST:** Es un verbo de HTTP utilizado para enviar datos al servidor para crear un nuevo recurso, como enviar un formulario o cargar un archivo.
- **Cookies:** Son pequeños archivos de datos que el servidor envía al navegador y que se almacenan en el dispositivo del usuario. Se utilizan para mantener el estado de la sesión, guardar preferencias o realizar seguimiento de la actividad del usuario en un sitio web.

    Los **tipos de cookies** principales son:

    1. **Cookies de sesión:** Son temporales y se eliminan cuando el navegador se cierra. Se utilizan para almacenar información durante una sesión de navegación, como el estado de inicio de sesión.
    2. **Cookies persistentes:** Permanecen en el dispositivo del usuario durante un período definido, incluso después de cerrar el navegador. Se usan para recordar preferencias, configuraciones o datos entre sesiones.
    3. **Cookies de terceros:** Son establecidas por un dominio distinto al del sitio web que el usuario está visitando, y generalmente se usan para seguimiento, publicidad y análisis.
    4. **Cookies seguras (Secure cookies):** Solo se transmiten a través de una conexión HTTPS segura, protegiendo la cookie contra interceptaciones en redes no seguras.
    5. **Cookies HTTP Only:** Solo pueden ser accedidas por el servidor a través de HTTP, lo que previene que sean manipuladas o leídas por JavaScript en el navegador, aumentando la seguridad frente a ataques como XSS.

Las **librerías de sesiones** gestionan cookies de sesión, asociando un identificador único al usuario para almacenar datos temporales en el servidor. Facilitan la autenticación y el seguimiento del usuario entre solicitudes, asegurando la seguridad de las cookies mediante opciones como **HTTP Only** y **Secure**.

Un **reverse proxy** actúa como intermediario entre el cliente y el servidor, gestionando solicitudes entrantes. Puede cifrar conexiones HTTPS, servir archivos estáticos para reducir la carga del servidor, mitigar ataques DDoS distribuyendo tráfico, y registrar las solicitudes para análisis y monitoreo. Un **forward proxy** actúa como intermediario entre el cliente y el servidor, mientras que

un **reverse proxy** lo hace entre el servidor y el cliente, gestionando las solicitudes hacia un servidor específico.

El **plugin mod_php** para Apache permite ejecutar código PHP directamente dentro del servidor web, integrando el motor PHP con el servidor para procesar dinámicamente las páginas PHP sin necesidad de configuraciones adicionales en la infraestructura del servidor.

El **full duplex** con **WebSockets** se refiere a la capacidad de enviar y recibir datos simultáneamente entre el cliente y el servidor en una única conexión persistente, permitiendo una comunicación bidireccional en tiempo real sin necesidad de establecer nuevas conexiones para cada mensaje.

La diferencia clave entre **WebSockets** y **HTTP** es que **WebSockets** permite una comunicación bidireccional y persistente, lo que significa que el cliente y el servidor pueden enviarse datos en tiempo real sin necesidad de hacer nuevas solicitudes. En cambio, **HTTP** es un protocolo unidireccional y sin estado, donde el cliente debe enviar una solicitud para recibir una respuesta del servidor, estableciendo una nueva conexión para cada intercambio.

La **comunicación en tiempo real** con **Node.js**, **Express** y **WebSockets** permite crear aplicaciones que interactúan instantáneamente entre el servidor y el cliente. **Node.js** maneja la lógica del servidor, **Express** gestiona las rutas y el servidor web, y **WebSockets** facilita la comunicación bidireccional en tiempo real mediante una conexión persistente, permitiendo a los clientes recibir datos instantáneamente sin necesidad de recargar la página o hacer solicitudes adicionales. Esto es ideal para aplicaciones como chats, notificaciones en vivo o juegos en línea.

6. Autentificación

La **autenticación** es el proceso de verificar la identidad de un usuario, mientras que la **autenticación biométrica** utiliza características físicas (como huellas dactilares, reconocimiento facial o iris) para validar al usuario de manera única. La **autenticación en dos pasos (2FA)** agrega una capa extra de seguridad, requiriendo no solo una contraseña, sino también un segundo factor (como un código enviado por mensaje o una aplicación de autenticación) para confirmar la identidad del usuario.

- **Basic Authentication:** Es un método de autenticación en el que el cliente envía un nombre de usuario y una contraseña codificados en base64 a través de los encabezados HTTP. Es sencillo, pero poco seguro sin HTTPS.

- **Cookies:** Son pequeños archivos de datos almacenados en el navegador del usuario que se utilizan para mantener la sesión activa, permitiendo que el servidor identifique al usuario en solicitudes subsecuentes sin requerir que inicie sesión nuevamente.

- **Tokens (como JWT):** Son cadenas de texto que se generan y envían al cliente tras una autenticación exitosa. El cliente incluye el token en las solicitudes subsiguientes, permitiendo la autenticación sin necesidad de almacenar información sensible en el cliente.

OAuth 2.0 es un protocolo de autorización que permite a los usuarios conceder acceso limitado a sus recursos en un servidor sin compartir sus credenciales, utilizando tokens de acceso.

Passport.js es un middleware para Node.js que facilita la implementación de autenticación en aplicaciones web, soportando diversos métodos de autenticación como **OAuth**, **JWT**, y más, a través de estrategias predefinidas.

Cross-Origin Resource Sharing (CORS): Es un mecanismo de seguridad que permite o restringe las solicitudes HTTP de un dominio a otro, controlando qué orígenes pueden acceder a los recursos de un servidor, protegiendo contra posibles ataques.

Cross-site Request Forgery (CSRF): Es un ataque en el que un usuario autenticado en una aplicación web es engañado para realizar acciones no deseadas en esa aplicación, sin su consentimiento, aprovechando la confianza que la aplicación tiene en el navegador del usuario.

Seguridad en Express + csurf: Express se puede asegurar mediante middleware como **csurf**, que protege contra ataques CSRF generando tokens de seguridad en cada solicitud y validándolos para asegurarse de que las peticiones provienen de fuentes legítimas.

7. Frameworks CSS y Tailwind

- **SGML (Standard Generalized Markup Language):** Es un lenguaje de marcas estándar que define la estructura de documentos y permite la creación de lenguajes de marcado personalizados, como HTML y XML.
- **HTML (HyperText Markup Language):** Es el lenguaje de marcado utilizado para estructurar y presentar contenido en la web, definiendo elementos como texto, enlaces, imágenes y tablas.
- **XML (Extensible Markup Language):** Es un lenguaje de marcado flexible que permite definir y almacenar datos de manera jerárquica y legible, sin predefinir etiquetas, permitiendo su uso en diferentes aplicaciones y plataformas.
- **XHTML (Extensible HyperText Markup Language):** Es una versión más estricta de HTML que sigue las reglas de sintaxis de XML, asegurando una mejor compatibilidad y consistencia en la interpretación de documentos web.

Los **elementos de HTML** son bloques básicos que definen la estructura y el contenido de una página web. Cada elemento generalmente consiste en una **etiqueta de apertura** (por ejemplo, `<div>`), un **contenido** (como texto, imágenes o enlaces) y una **etiqueta de cierre** (por ejemplo, `</div>`). Algunos elementos comunes incluyen:

- `<h1>`, `<h2>`, ..., `<h6>`: Encabezados de diferentes niveles.
- `<p>`: Párrafo de texto.
- `<a>`: Enlace o hipervínculo.
- `<img>`: Imagen.
- `<ul>`, `<ol>`, `<li>`: Listas desordenadas, ordenadas y elementos de lista.
- `<div>`: Contenedor genérico para agrupar otros elementos.
- `<span>`: Contenedor en línea para aplicar estilos o agrupar texto.

Responsive Design: El diseño **responsive** ajusta el contenido automáticamente a diferentes tamaños de pantalla utilizando **CSS** y **media queries**, asegurando que la página se vea bien en cualquier dispositivo.

- **Adaptive Design:** El diseño **adaptive** utiliza diferentes plantillas o layouts prediseñados que se cargan según el dispositivo o tamaño de pantalla detectado, adaptándose a varios tamaños específicos.
- **Mobile-First:** El enfoque **mobile-first** significa diseñar y desarrollar primero para dispositivos móviles, y luego adaptar la experiencia para pantallas más grandes, priorizando la funcionalidad en dispositivos pequeños.

Bootstrap: Es un framework CSS que proporciona componentes y plantillas predefinidos para diseñar sitios web responsivos y móviles de manera rápida, utilizando una estructura de rejilla y estilos prediseñados.

Tailwind CSS: Es un framework CSS basado en utilidades, donde se aplican clases específicas para cada estilo (como colores, márgenes, tipografía) directamente en el HTML, ofreciendo mayor flexibilidad y personalización en comparación con otros frameworks.

8. API Restful y Logging

API (Interfaz de Programación de Aplicaciones): Es un conjunto de reglas y protocolos que permiten que diferentes software o aplicaciones se comuniquen entre sí, facilitando el acceso a funciones y datos de un sistema sin necesidad de conocer su implementación interna.

API REST (Representational State Transfer): Es un tipo de API que utiliza los principios de REST, un estilo arquitectónico que se basa en el uso de solicitudes HTTP para interactuar con recursos (datos), y emplea métodos estándar como GET, POST, PUT y DELETE para realizar operaciones. Se enfoca en la simplicidad, escalabilidad y el uso de representaciones de recursos (generalmente en formato JSON o XML). Son sin estado. Contienen tod la información para obtener el recurso.

El **principio de idempotencia** significa que una operación puede repetirse múltiples veces sin cambiar el resultado más allá de la primera ejecución. En las **APIs REST**, esto asegura que si se realiza la misma solicitud varias veces, el efecto será el mismo, sin causar efectos adicionales.

Los **endpoints** son puntos de acceso específicos dentro de una **API** donde se pueden realizar solicitudes para interactuar con los recursos de un sistema. Cada endpoint está asociado a una URL única y define una acción que el servidor debe realizar (como obtener, crear, actualizar o eliminar datos), generalmente usando métodos HTTP como **GET**, **POST**, **PUT** o **DELETE**.

Swagger es una herramienta de documentación para **APIs REST**, que permite describir, consumir y visualizar las operaciones de una API de manera interactiva y comprensible, generando automáticamente documentación a partir de especificaciones en formato **OpenAPI**.

Logging es el proceso de registrar información sobre la ejecución de una aplicación, como errores, eventos o datos importantes, para ayudar en la depuración y monitoreo. En **Node.js**, se

puede implementar el logging utilizando módulos como **console** para mensajes simples o bibliotecas más avanzadas como **Winston** o **Morgan**, que ofrecen funcionalidades para registrar eventos, gestionar niveles de log y almacenar registros de manera más estructurada y persistente.ç

Los **niveles de logging** indican la gravedad de los mensajes registrados:

- `debug`: Información detallada para depuración.
- `info`: Mensajes informativos sobre el estado.
- `warn`: Advertencias de posibles problemas.
- `error`: Errores que indican fallos.
- **`fatal`** o **`critical`**: Errores graves que requieren atención inmediata.

Los **manejadores** (handlers) son funciones o módulos que gestionan eventos o solicitudes en una aplicación. En el contexto de **Node.js** y **APIs**, los manejadores procesan eventos, como solicitudes HTTP, y ejecutan la lógica correspondiente, como devolver una respuesta o registrar información.

Winston es una librería de **logging** para **Node.js** que permite registrar mensajes en diferentes niveles de severidad (como `info`, `error`, `warn`, `debug`). Permite configurar varios **transportes**, que son los destinos donde se guardan los registros (por ejemplo, archivos, bases de datos, consola, servicios externos). Winston es flexible y permite personalizar el formato de los logs y gestionar diferentes configuraciones para diferentes entornos.

9. Document Object Model (DOM) y Diseño UI

El **Document Object Model (DOM)** es una representación estructural de un documento HTML o XML en forma de un árbol, donde cada elemento, atributo y texto del documento se convierte en un objeto accesible y manipulable mediante JavaScript. El DOM permite interactuar dinámicamente con la estructura de una página web, como modificar su contenido, estilo o responder a eventos del usuario.

El **JavaScript DOM** es una interfaz que permite a JavaScript interactuar y manipular la estructura de un documento HTML o XML. A través del DOM, se pueden modificar elementos, atributos, contenido y estilo de una página web de manera dinámica.

- **Búsqueda de elementos**: Se usa JavaScript para localizar elementos del DOM mediante métodos como `getElementById`, `getElementsByClassName`, o `querySelector`, entre otros.
- **Recorrido del DOM**: Consiste en navegar a través de los nodos del árbol DOM usando propiedades como `parentNode`, `childNodes`, `nextSibling` o `previousSibling`.
- **Cambios del DOM**: Permite modificar el contenido, estructura y estilo de la página, como añadir, eliminar o cambiar elementos con métodos como `appendChild`, `removeChild` o `setAttribute`.

- **Browser Events**: Son eventos (como clics o teclas presionadas) que se detectan en el navegador, y se manejan con JavaScript para interactuar con el usuario, usando `addEventListener` para asignar funciones a esos eventos.
- **Fetch**: Es una API de JavaScript para hacer solicitudes HTTP asíncronas, permitiendo obtener o enviar datos de manera eficiente entre el cliente y el servidor.

El **diseño responsive** es un enfoque de diseño web que ajusta el contenido y la estructura de una página según el tamaño y la resolución de la pantalla, utilizando **CSS** y **media queries** para ofrecer una experiencia óptima en dispositivos de diferentes tamaños, como móviles, tabletas y escritorios.

Las **unidades relativas** son unidades de medida en CSS que se basan en el tamaño del entorno, como el tamaño de la fuente o el tamaño de la ventana. Ejemplos incluyen:

- `em`: Relativo al tamaño de fuente del elemento actual.
- `rem`: Relativo al tamaño de fuente del elemento raíz (`<html>`).
- `%`: Relativo al tamaño del elemento contenedor.

Flexbox es un modelo de diseño en CSS que permite alinear y distribuir elementos de manera eficiente dentro de un contenedor, incluso cuando su tamaño es desconocido o dinámico. Utiliza propiedades como `display: flex`, `justify-content`, `align-items` y `flex-direction` para organizar los elementos en filas o columnas, ofreciendo una forma más flexible y controlada de gestionar el espacio y el alineamiento en el diseño web. Es responsivo.

Media queries en diseño adaptativo se utilizan para aplicar diferentes estilos en función de características específicas del dispositivo, como el tamaño de pantalla o la orientación. A diferencia del diseño responsivo, donde el contenido se ajusta fluidamente, el diseño adaptativo usa **media queries** para cambiar completamente el diseño de la página con plantillas específicas para cada tipo de dispositivo, como móvil, tablet o escritorio.

UX (User Experience) se refiere a la experiencia general de un usuario al interactuar con un producto o servicio, especialmente en aplicaciones y sitios web. Se enfoca en mejorar la satisfacción del usuario a través de un diseño intuitivo, accesible, eficiente y agradable, considerando aspectos como la facilidad de uso, la velocidad, la accesibilidad y la estética.

UI (User Interface) se refiere al diseño visual y la disposición de los elementos interactivos en una aplicación o sitio web, como botones, menús y colores. **UX (User Experience)**, por otro lado, se enfoca en la experiencia global del usuario al interactuar con ese diseño, asegurando que sea fácil, eficiente y satisfactoria. En resumen, UI es el diseño de la interfaz, mientras que UX es la experiencia que el usuario tiene al usarla.

10. Componentes de Frontend y React

En aplicaciones web con **UI complejo**, parte del código se pasa al **cliente** para mejorar el rendimiento. Esto se logra empaquetando el código **ES6** en un solo archivo **`bundle.js`** mediante herramientas como **Webpack**. Webpack agrupa todos los módulos y recursos (como scripts, estilos e imágenes) en un solo archivo optimizado, lo que permite cargar la aplicación de manera más

eficiente y distribuida al cliente, reduciendo tiempos de carga y mejorando la experiencia del usuario.

- **SPA (Single-Page Application)**: Es una aplicación web donde todo el contenido se carga en una sola página, y las interacciones posteriores se manejan mediante JavaScript sin recargar la página. Ofrece una experiencia más fluida y rápida.
- **MPA (Multi-Page Application)**: Son aplicaciones web tradicionales que cargan una nueva página completa cada vez que el usuario navega a una sección diferente. Cada acción genera una solicitud y recarga del servidor.

¿Cuál elegir?

- **SPA** es ideal para aplicaciones interactivas y con contenido dinámico (como redes sociales o aplicaciones de productividad).
- **MPA** es más adecuada para sitios web con muchas páginas estáticas o de contenido más estructurado, como blogs o tiendas online con múltiples categorías.

React es una librería de JavaScript para construir interfaces de usuario (UI) interactivas y dinámicas en aplicaciones web. Permite crear componentes reutilizables que gestionan su propio estado y se actualizan eficientemente cuando los datos cambian.

En el patrón MVC, React se ocupa principalmente de la Vista (V), ya que se enfoca en renderizar la interfaz de usuario y manejar la interacción con el usuario, dejando el Modelo (M) y el Controlador (C) a otras partes de la aplicación, como el servidor o librerías de gestión de estado.

- **Componentes reutilizables**: En **React**, los componentes son bloques de código independientes que gestionan su propio estado y se pueden reutilizar en diferentes partes de la aplicación, lo que facilita la organización y mantenimiento del código.
- **DOM Virtual**: React usa un **DOM Virtual**, que es una representación ligera y optimizada del **DOM real** del navegador. Cuando el estado de un componente cambia, React actualiza primero el DOM Virtual y luego compara (a través del proceso de *reconciliación*) las diferencias con el DOM real, realizando solo las actualizaciones necesarias, lo que mejora el rendimiento.
- **JSX**: **JSX** (JavaScript XML) es una sintaxis de **React** que permite escribir componentes con una mezcla de HTML y JavaScript. Aunque parece HTML, JSX es convertido a código JavaScript que React puede procesar y renderizar en el DOM real.
- **Props**: Son propiedades que se pasan a los componentes desde su componente padre, permitiendo que los componentes sean dinámicos y reutilizables.
- **Eventos**: Son acciones del usuario (como clics o teclas presionadas) que se manejan en React para ejecutar funciones o cambiar el estado.
- **Estado**: Es un objeto que almacena información local en un componente, y su cambio provoca la actualización de la UI.

Next.js: Es un framework de React que facilita la creación de aplicaciones web optimizadas, permitiendo renderizado **del lado del servidor (SSR)** y **generación estática (SSG)**, con características como rutas automáticas y prefetching de datos.

Astro: Es un framework para crear sitios web rápidos y modernos, que permite usar múltiples tecnologías de frontend (React, Vue, Svelte) y optimiza el rendimiento al generar sitios estáticos con la menor cantidad posible de JavaScript.

Vite: Es una herramienta de desarrollo que actúa como un **build tool** y **bundler** ultra-rápido para aplicaciones modernas, utilizando **ESModules** y optimizando el proceso de desarrollo con recarga instantánea y tiempos de construcción reducidos.

11. Taller React/Tailwind

React y **Tailwind CSS** trabajan juntos para crear interfaces dinámicas y estilizadas.

- **Componentes**: En React, defines componentes que encapsulan tanto la lógica como el estilo. Usas clases de **Tailwind CSS** directamente dentro de los componentes para aplicar estilos.
- **Props**: Puedes pasar **props** a los componentes para personalizar su comportamiento o apariencia, incluyendo clases de Tailwind como props dinámicas (por ejemplo, `className="bg-blue-500"`).
- **Eventos**: React maneja eventos como clics o cambios en formularios, y estos eventos pueden desencadenar acciones que alteran el estado del componente, lo que puede afectar tanto la lógica como la apariencia, usando clases de Tailwind que reaccionan a esos cambios.

12. Generación HTML y Despliegue de aplicaciones en Nube

La **generación de HTML** se refiere al proceso de crear y estructurar el contenido de una página web utilizando etiquetas HTML. Puede hacerse de manera estática, escribiendo directamente el código HTML en un archivo, o dinámicamente, generando HTML a través de lenguajes de programación del lado del servidor (como PHP, Node.js) o del lado del cliente (como JavaScript). En aplicaciones modernas, frameworks y bibliotecas como React o Vue generan HTML de forma dinámica según el estado de la aplicación y las interacciones del usuario.

La diferencia entre **SSR (Server-Side Rendering)** y **SSG (Static Site Generation)** radica en cuándo se genera el HTML de una página:

- **SSR**: El HTML se genera dinámicamente en el servidor en cada solicitud del usuario. Esto permite contenido personalizado y actualizado en tiempo real, pero puede ser más lento debido a la generación del contenido en cada carga.
- **SSG**: El HTML se genera de forma estática durante la construcción del sitio (antes de ser desplegado). Esto permite tiempos de carga rápidos, pero el contenido no cambia hasta que se vuelve a generar el sitio, lo que es ideal para páginas con contenido estático.

El **despliegue** de una aplicación involucra varios entornos, cada uno con un propósito específico:

- **Development Environment (Entorno de Desarrollo)**: Es donde los desarrolladores escriben y prueban el código de la aplicación. Aquí se permite el uso de herramientas de depuración, con configuraciones flexibles para facilitar el desarrollo rápido y sin restricciones.
- **Testing Environment (Entorno de Pruebas)**: Es donde se ejecutan las pruebas para asegurarse de que el código funcione correctamente antes de su lanzamiento. Este entorno simula condiciones de producción pero permite pruebas de integración, rendimiento y calidad del código.
- **Production Environment (Entorno de Producción)**: Es el entorno en el que la aplicación se ejecuta en vivo y es accesible para los usuarios finales. Está optimizado para el rendimiento, la seguridad y la estabilidad, y cualquier cambio debe ser cuidadosamente controlado para evitar interrupciones del servicio.

Code Splitting: Es una técnica en React que divide el código en "fragmentos" más pequeños (bundles) que se cargan solo cuando son necesarios. Esto mejora el tiempo de carga inicial de la aplicación, ya que solo se carga el código relevante para la vista actual del usuario.

Lazy Loading Images: Es una técnica que retrasa la carga de imágenes hasta que el usuario se desplaza hacia ellas en la página. Esto optimiza el rendimiento de la página, ya que solo se cargan las imágenes visibles en la pantalla, reduciendo el tiempo de carga inicial.

Un **proxy inverso** como **Nginx** actúa como intermediario entre los usuarios y los servidores backend, redirigiendo las solicitudes de los usuarios a los servidores adecuados. Se utiliza para mejorar el rendimiento, gestionar el tráfico, equilibrar la carga entre múltiples servidores, y proporcionar seguridad, como ocultar la infraestructura interna o gestionar certificados SSL. Nginx también puede servir contenido estático y manejar solicitudes HTTPS.

- **IaaS (Infrastructure as a Service)**: Proporciona infraestructura de TI virtualizada a través de la nube, como servidores, almacenamiento y redes, sin necesidad de gestionar hardware físico.
- **PaaS (Platform as a Service)**: Ofrece una plataforma completa para desarrollar, ejecutar y gestionar aplicaciones sin preocuparse por la infraestructura subyacente, como servidores o bases de datos.
- **SaaS (Software as a Service)**: Proporciona aplicaciones listas para usar a través de la nube, accesibles desde cualquier dispositivo sin necesidad de instalar ni mantener software, como Gmail o Dropbox.