Módulo 6
Desarrollo de aplicaciones Web Progresivas (PWA)

Almacenamiento en una PWA

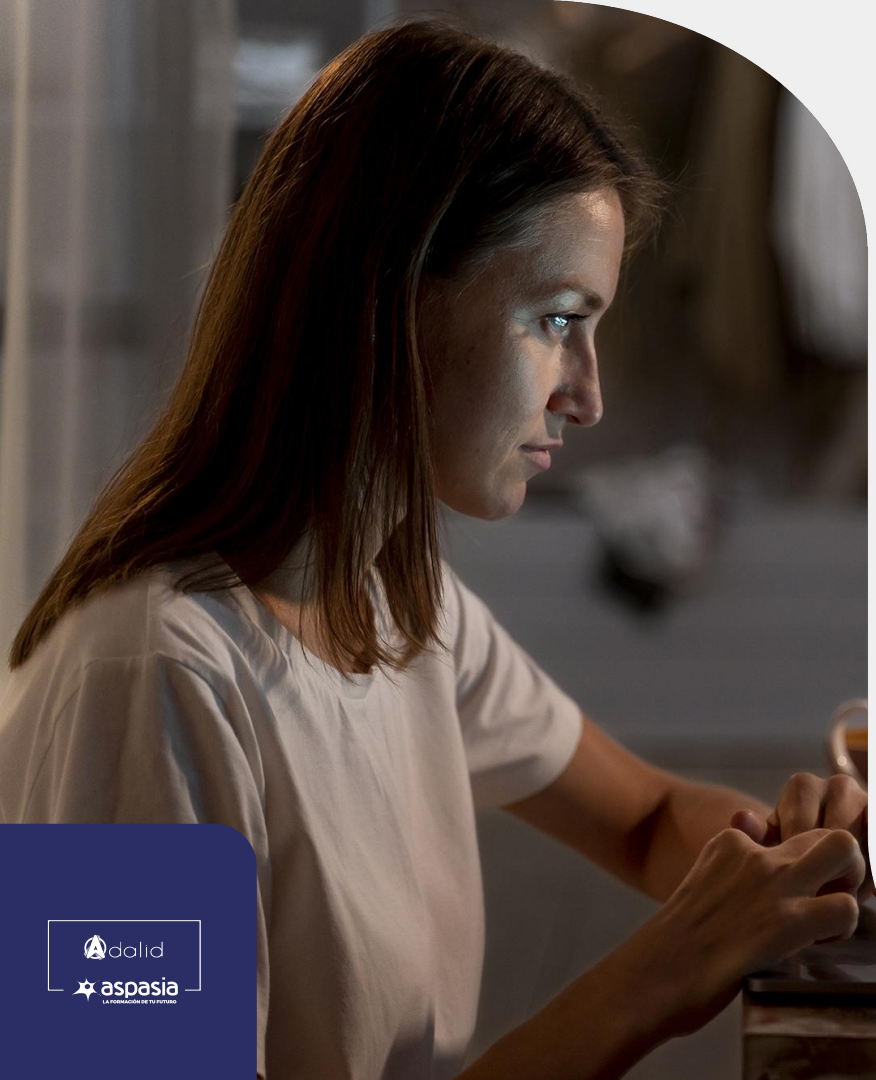
Módulo 6

AE 2.1

OBJETIVOS

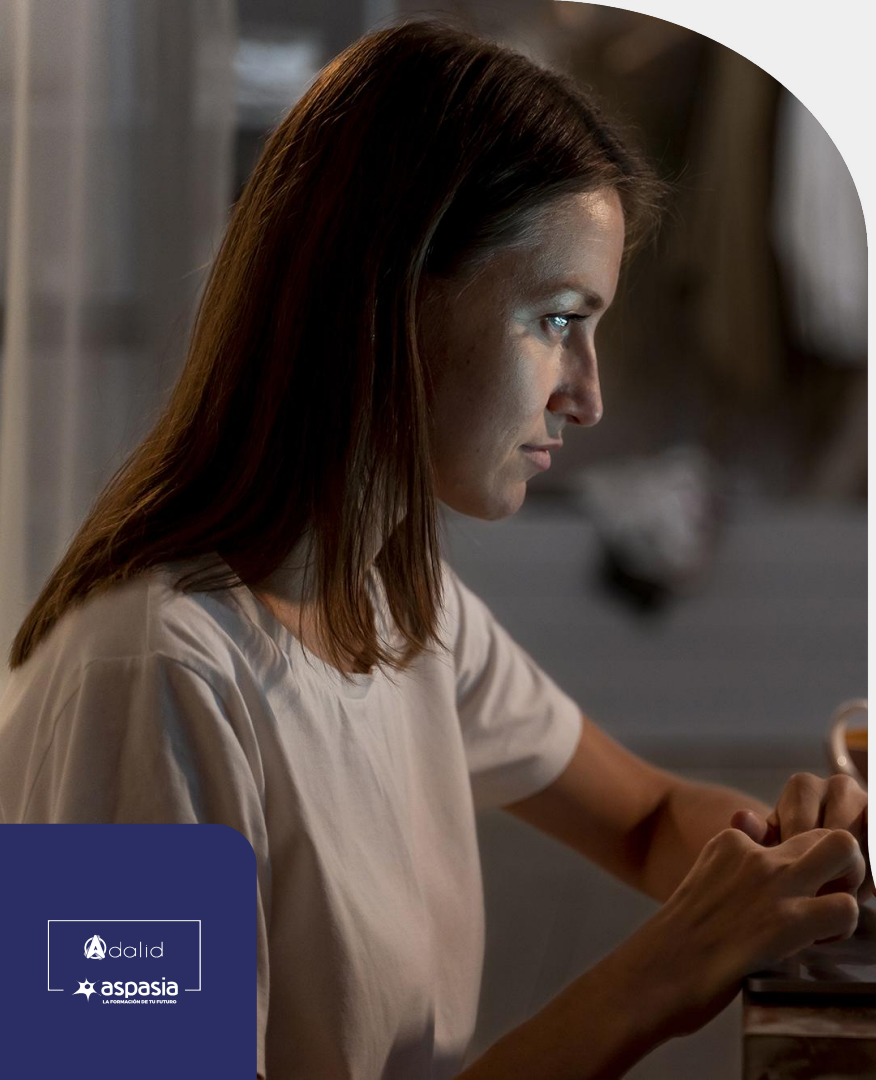
Entender como las PWA usan LocalStorage, IndexedDB y Service Workers para almacenar datos y mejorar la experiencia offline. Como con Lighthouse, se evalúa su rendimiento, instalación y optimización, logrando apps rápidas, confiables e intuitivas.

Chile
Adalid
aspasia
LA FORMACIÓN DE TU FUTURO



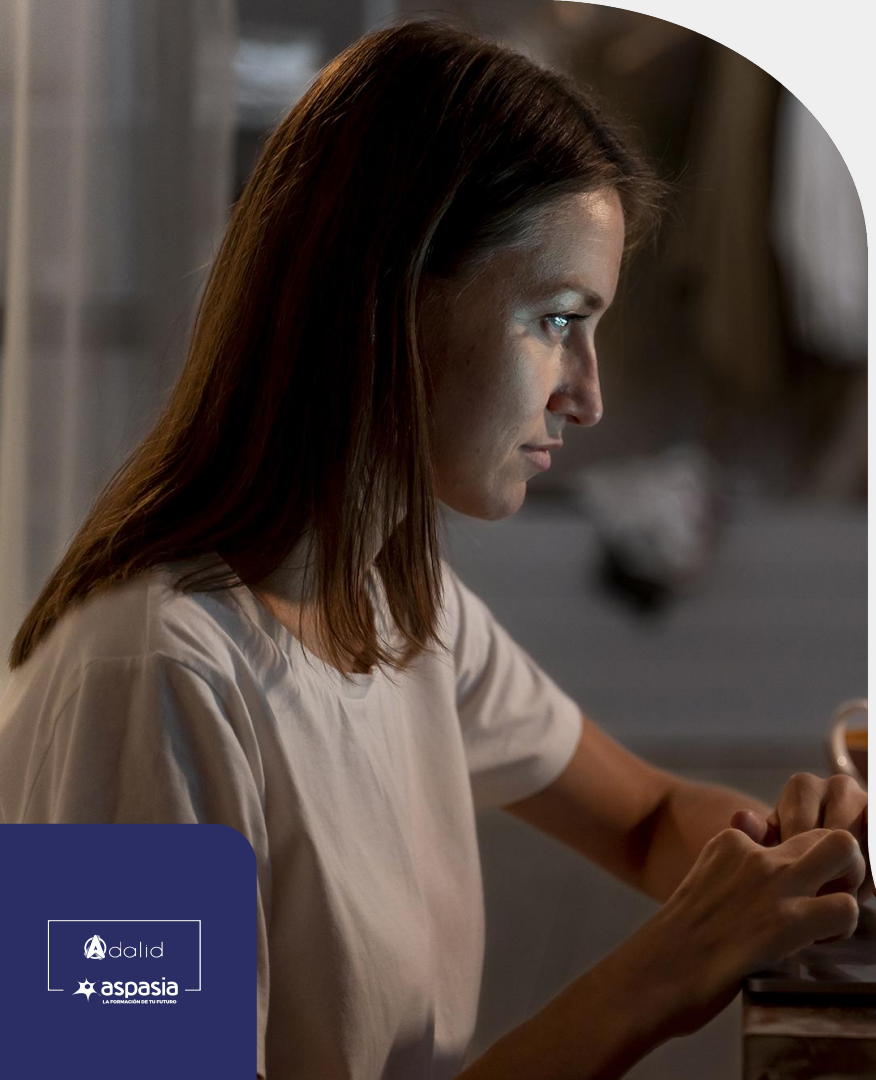
¿QUÉ VAMOS A VER?

- Almacenamiento en una PWA.
  - Cómo se administra el Almacenamiento Web en una PWA.
  - El uso de LocalStorage y SessionStorage.
  - Acerca de WebAssembly y el código precompilado.
  - Bibliotecas de bases de datos con soporte para PWA(IndexedDB, PouchDB, RxDB, GunDB).



¿QUÉ VAMOS A VER?

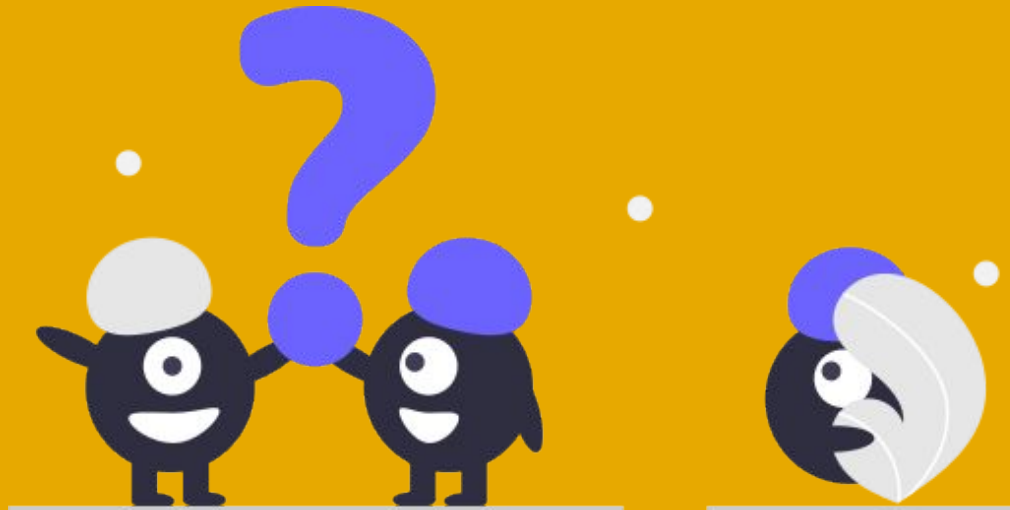
- Creando una PWA.
  - Creando un proyecto PWA en ReactJs.
  - Registrando un Service Worker.
  - Personalizando un Service Worker.
  - Navegando a través de una PWA.
  - Implementando las distintas estrategias de almacenamiento en caché de Service Worker.
  - Accediendo a periféricos del Sistema Operativo.
  - Despliegue de una PWA.

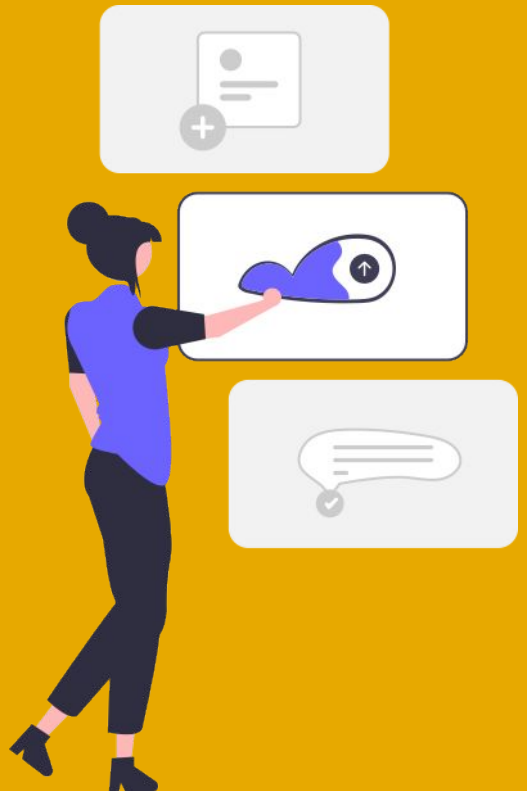


¿QUÉ VAMOS A VER?

- Explorando el Estado de una PWA con Lighthouse.
  - Instalando la extensión lighthouse en el navegador.
  - Algunos aspectos relevantes de lighthouse (Rápido, instalable, optimizado para PWA).
  - Ejecutando lighthouse (Análisis de carga de página)
  - Revisando el informe de lighthouse.

¿Cómo se almacenan los datos en un dispositivo?





Almacenamiento en una PWA.

Cómo se Administra el Almacenamiento Web en una PWA

Las PWA necesitan almacenar datos de forma eficiente para mejorar la experiencia del usuario, permitiendo **acceso sin conexión y mejor rendimiento**.

Tipos de almacenamiento disponibles:

- Almacenamiento basado en clave-valor (LocalStorage, SessionStorage).
- Bases de datos estructuradas (IndexedDB, PouchDB, RxDB, GunDB).
- Almacenamiento en caché a través de Service Workers.

Cómo se Administra el Almacenamiento Web en una PWA

Tipo de Almacenamiento	Persistencia	Capacidad	Uso Ideal
LocalStorage	Persistente	~5MB	Preferencias del usuario, configuraciones.
SessionStorage	Se elimina al cerrar la pestaña	~5MB	Datos temporales de sesión.
IndexedDB	Persistente	GBs	Datos complejos, apps offline.
Cache Storage	Persistente	Ilimitado	Recursos estáticos (CSS, imágenes, APIs).

El Uso de LocalStorage y SessionStorage

LocalStorage

- Permite almacenar datos clave-valor de manera persistente en el navegador.
- No tiene fecha de expiración.

Ejemplo: Guardar y recuperar datos con localStorage

```
// Guardar datos
localStorage.setItem("usuario", "Juan Pérez");

// Recuperar datos
const usuario = localStorage.getItem("usuario");
console.log(usuario); // Juan Pérez
```

El Uso de LocalStorage y SessionStorage

SessionStorage

- Similar a localStorage, pero los datos se eliminan al cerrar la pestaña.

Ejemplo: Guardar y recuperar datos con sessionStorage

```
// Guardar datos
sessionStorage.setItem("tema", "Oscuro");

// Recuperar datos
const tema = localStorage.getItem("tema");
console.log(tema); // oscuro
```

Acerca de WebAssembly y el Código Precompilado

Es un formato de código binario que permite ejecutar código de alto rendimiento en el navegador.

Beneficios de WebAssembly en PWA:

- Permite ejecutar aplicaciones más rápidas en comparación con JavaScript.
- Soporta múltiples lenguajes como C, C++, Rust.
- Ideal para juegos, edición de imágenes y procesamiento de datos.

Acerca de WebAssembly y el Código Precompilado

Ejemplo de un Módulo WebAssembly Simple

1. Compila un archivo **C** a WebAssembly:

```
// archivo.c
int multiplicar(int a, int b) {
    return a * b;
}
```

2. Compilar con **Emscripten**:

```
emcc archivo.c -o archivo.wasm -s SIDE_MODULE=1
```

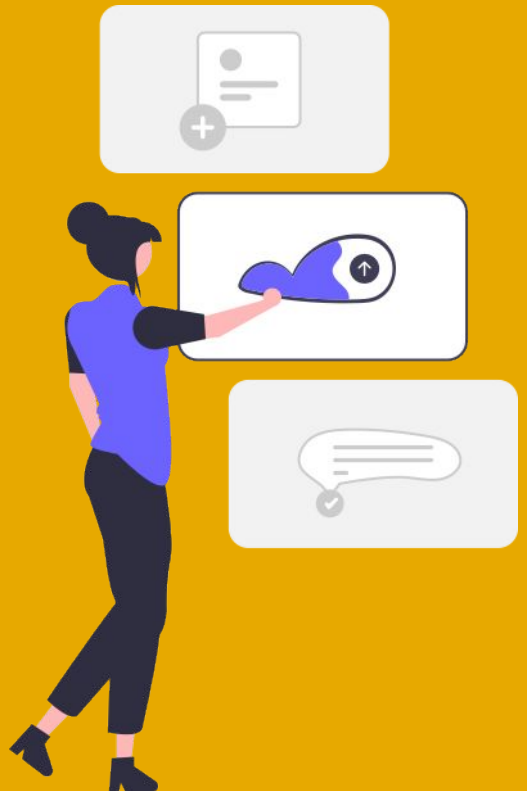
3. Importar en **JavaScript**:

```
fetch('archivo.wasm')
  .then(response => response.arrayBuffer())
  .then(bytes => WebAssembly.instantiate(bytes))
  .then(result => {
    console.log(result.instance.exports.multiplicar(3, 4)); // 12
  });
```

Acerca de WebAssembly y el Código Precompilado

¿Cuándo usar WebAssembly en una PWA?

- Aplicaciones con procesamiento intensivo (juegos, edición de video).
- Mejorar el rendimiento en comparación con JavaScript.



Bibliotecas de Bases de Datos con Soporte para PWA

Adalid
aspasia
LA FORMACIÓN DE TU FUTURO

Bibliotecas de Bases de Datos con Soporte para PWA

IndexedDB

- Base de datos NoSQL integrada en el navegador.
- Permite almacenar grandes cantidades de datos estructurados.

```
const request = indexedDB.open("MiBaseDeDatos", 1);

request.onupgradeneeded = event => {
  const db = event.target.result;
  db.createObjectStore("usuarios", { keyPath: "id" });
};

request.onsuccess = event => {
  const db = event.target.result;
  console.log("IndexedDB lista para usarse");
};
```

Bibliotecas de Bases de Datos con Soporte para PWA

PouchDB

- Biblioteca que permite almacenar datos offline y sincronizarlos con un servidor.

```
import PouchDB from "pouchdb";

const db = new PouchDB("mi_pwa");

db.put({
  _id: "usuario_1",
  nombre: "Juan",
  edad: 30
}).then(() => console.log("Usuario guardado"));
```

Bibliotecas de Bases de Datos con Soporte para PWA

RxDB

- Base de datos reactiva con sincronización en tiempo real.
- Usa IndexedDB como almacenamiento base.

```
import { createRxDatabase, addRxPlugin } from 'rxdb';
import { getRxStorageDexie } from
'rxdb/plugins/storage-dexie';

const db = await createRxDatabase({
  name: 'mi_pwa_db',
  storage: getRxStorageDexie()
});

console.log('Base de datos creada:', db);
```

Bibliotecas de Bases de Datos con Soporte para PWA

GunDB

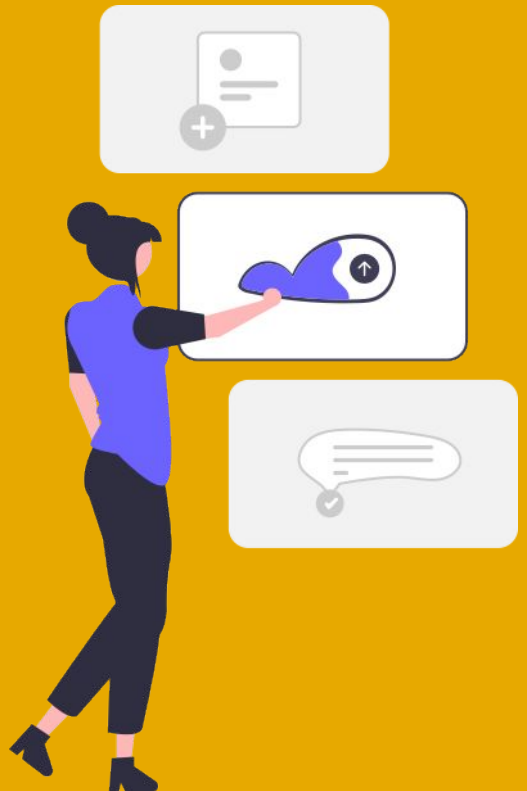
- Base de datos descentralizada y distribuida.
- Permite sincronización de datos en múltiples dispositivos sin servidor.

```
import Gun from 'gun';

const gun = Gun();
const usuarios = gun.get('usuarios');

usuarios.set({ nombre: "Juan", edad: 30 });

usuarios.map().once(data => console.log(data));
```



Creando una PWA.

Creando un Proyecto PWA en ReactJS

Antes hemos explorado cómo se crea y se configura un proyecto con REACT implementando PWA, pero vamos a darle una mirada nuevamente.

Creando un Proyecto PWA en ReactJS

Paso 1: Configurar un Proyecto con Vite

```
npm create vite@latest mi-pwa --template react
cd mi-pwa
npm install
```

Paso 2: Instalar Dependencias para PWA

```
npm install vite-plugin-pwa --save-dev
```

Registrando un Service Worker

Paso 1: Registrando un Service Worker en el archivo vite.config.js

```
import { defineConfig } from 'vite';
import react from '@vitejs/plugin-react';
import { VitePWA } from 'vite-plugin-pwa';

export default defineConfig({
  plugins: [
    react(),
    VitePWA({
      registerType: 'autoUpdate',
      manifest: {
        name: 'Mi PWA',
        short_name: 'PWA',
        description: 'Aplicación PWA con React y Vite',
        theme_color: '#ffffff',
        start_url: '/',
        display: 'standalone',
        icons: [
          {
            src: '/icons/icon-192x192.png',
            sizes: '192x192',
            type: 'image/png'
          },
          {
            src: '/icons/icon-512x512.png',
            sizes: '512x512',
            type: 'image/png'
          }
        ]
      }
    })
  ]
});
```

Registrando un Service Worker

Paso 2: Crear el Archivo del Service Worker (public/sw.js)

```
self.addEventListener("install", (event) => {
  console.log("Service Worker instalado");
  event.waitUntil(self.skipWaiting());
});

self.addEventListener("activate", (event) => {
  console.log("Service Worker activado");
});

self.addEventListener("fetch", (event) => {
  console.log("Interceptando solicitud:", event.request.url);
});
```

Paso 3: Registrar el Service Worker en src/main.jsx

```
import ReactDOM from "react-dom";
import App from "./App";

if ("serviceWorker" in navigator) {
  navigator.serviceWorker.register("/sw.js")
    .then(() => console.log("Service Worker registrado"))
    .catch(error => console.log("Error en el registro del Service Worker:", error));
}

ReactDOM.createRoot(document.getElementById("root")).render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
```

Personalizando un Service Worker

Paso 1: Personalizando un Service Worker (public/sw.js)

- Modifica sw.js para incluir almacenamiento en caché
- Permite que los usuarios accedan a los archivos incluso sin conexión.

```
const CACHE_NAME = "pwa-cache-v1";
const urlsToCache = ["/", "/index.html", "/icons/icon-192x192.png"];

self.addEventListener("install", event => {
  event.waitUntil(
    caches.open(CACHE_NAME).then(cache => {
      return cache.addAll(urlsToCache);
    })
  );
});

self.addEventListener("fetch", event => {
  event.respondWith(
    caches.match(event.request).then(response => {
      return response || fetch(event.request);
    })
  );
});
```

Navegando a través de una PWA

Paso 1: Crear un Archivo de Rutas con React Router src/routes.jsx

```jsx
import { BrowserRouter as Router, Routes, Route } from "react-router-dom";
import Home from "./pages/Home";
import About from "./pages/About";

export default function AppRoutes() {
  return (
    <Router>
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/about" element={<About />} />
      </Routes>
    </Router>
  );
}
```

Paso 2: Integrar las Rutas en App.jsx

- Permite navegar entre páginas sin recargar

```jsx
import AppRoutes from "./routes";

function App() {
  return (
    <div>
      <h1>Mi PWA 🚀</h1>
      <AppRoutes />
    </div>
  );
}

export default App;
```

Implementando Estrategias de Caché en Service Worker

Estrategia: Cache-First

```
self.addEventListener("fetch", event => {
  event.respondWith(
    caches.match(event.request).then(response => {
      return response || fetch(event.request);
    })
  );
});
```

Estrategia: Network-First

```
self.addEventListener("fetch", event => {
  event.respondWith(
    fetch(event.request)
      .then(networkResponse => {
        return caches.open(CACHE_NAME).then(cache => {
          cache.put(event.request, networkResponse.clone());
          return networkResponse;
        });
      })
      .catch(() => caches.match(event.request))
  );
});
```

Accediendo a Periféricos del Sistema Operativo

Ejemplo: Acceder a la cámara usando la API de MediaStream

```
navigator.mediaDevices.getUserMedia({ video: true })
  .then(stream => {
    document.getElementById("video").srcObject = stream;
  })
  .catch(error => console.log("Error al acceder a la cámara:", error));
```

Ejemplo: Obtener la ubicación del usuario

```
navigator.geolocation.getCurrentPosition(position => {
  console.log("Latitud:", position.coords.latitude);
  console.log("Longitud:", position.coords.longitude);
});
```

Despliegue de una PWA

Paso 1: Generar la Versión para Producción

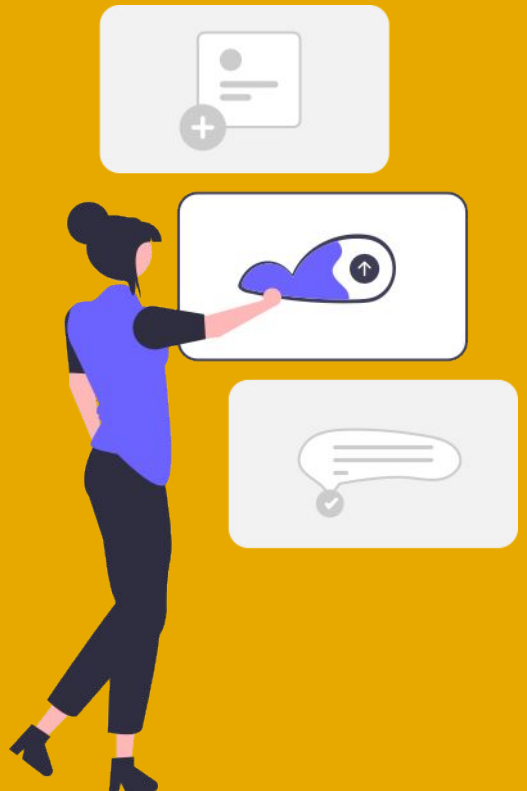
```
npm run build
```

Paso 2: Instalar un Servidor Estático

```
npm install -g serve
serve -s dist
```

Paso 3: Subir a un Servidor con HTTPS (Ejemplo con Vercel)

```
npm install -g vercel
vercel deploy
```



Lighthouse.

Instalando la Extensión Lighthouse en el Navegador

¿Qué es Lighthouse?

Lighthouse es una herramienta de auditoría de Google que permite analizar el rendimiento, accesibilidad y optimización de Progressive Web Apps (PWA).

Aspectos Relevantes de Lighthouse en PWA

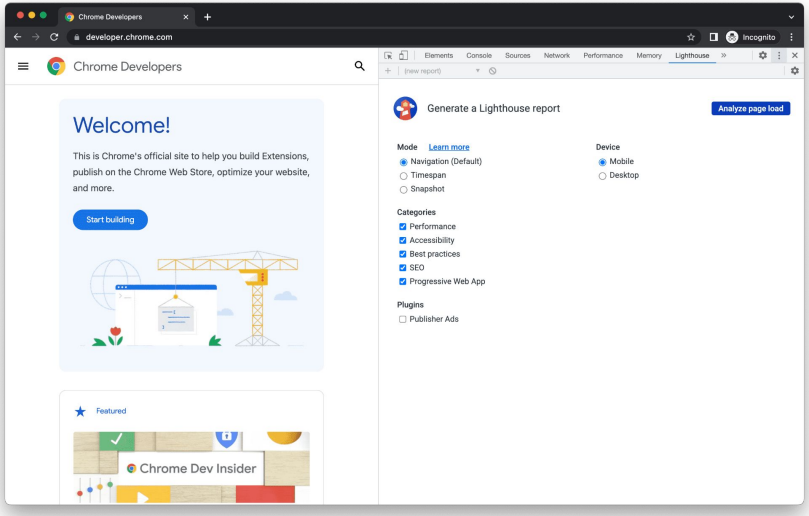
Lighthouse evalúa la calidad general de una PWA asegurando que sea rápida y accesible.

- **Rápido:**
  - Evalúa tiempos de carga, renderizado y uso de caché en la PWA.
- **Instalabilidad:**
  - Confirma si la PWA cumple con los requisitos para ser instalada en dispositivos.
- **Optimización:**
  - Verifica el uso de Service Workers, HTTPS, y Responsive Design.
- **Evaluaciones clave:**
  - **Performance:** Velocidad de carga y respuesta.
  - **PWA:** Cumplimiento de criterios de instalación.
  - **SEO:** Accesibilidad y buenas prácticas.

Ejecutando Lighthouse (Análisis de Carga de Página)

Paso 1: Abrir Lighthouse en Chrome DevTools

1. Abre **Google Chrome** y navega a tu PWA (http://localhost:5173).
2. Pulsa **F12** o **Ctrl + Shift + I** para abrir DevTools.
3. Dirígete a la pestaña **Lighthouse**.



Ejecutando Lighthouse (Análisis de Carga de Página)

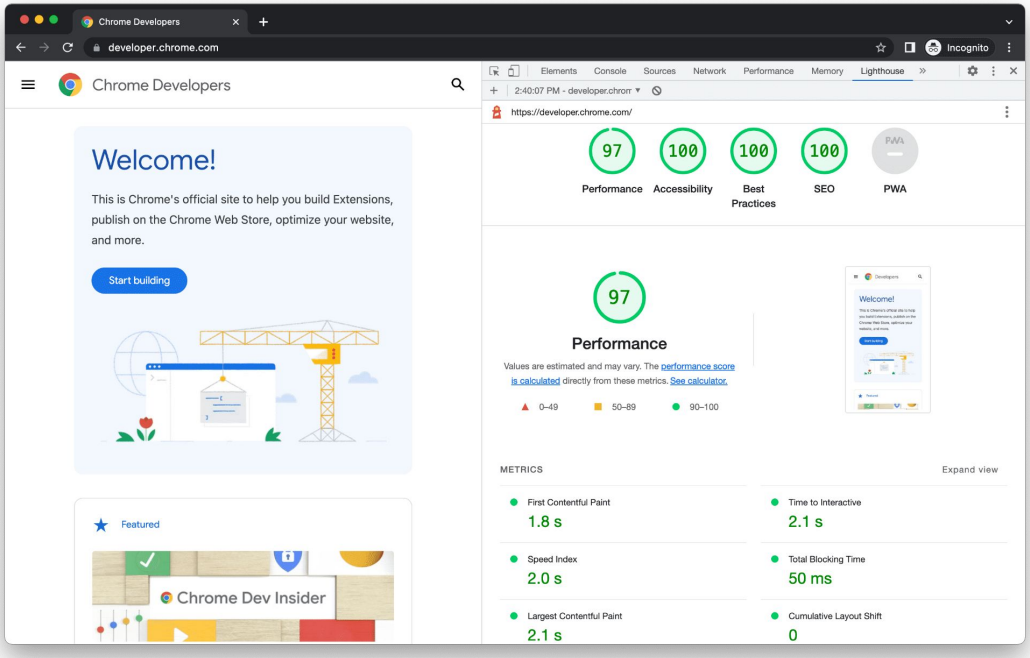
Paso 2: Seleccionar Métricas a Evaluar

- En **Categorías**, marca:
  - Performance
  - Progressive Web App
  - SEO
- En **Dispositivo**, selecciona:
  - Móvil o Escritorio

Paso 3: Ejecutar el Análisis

- Haz clic en **"Analyze Page Load"**.
- Espera unos segundos hasta que Lighthouse genere el informe.

Ejecutando Lighthouse (Análisis de Carga de Página)



Revisando el Informe de Lighthouse

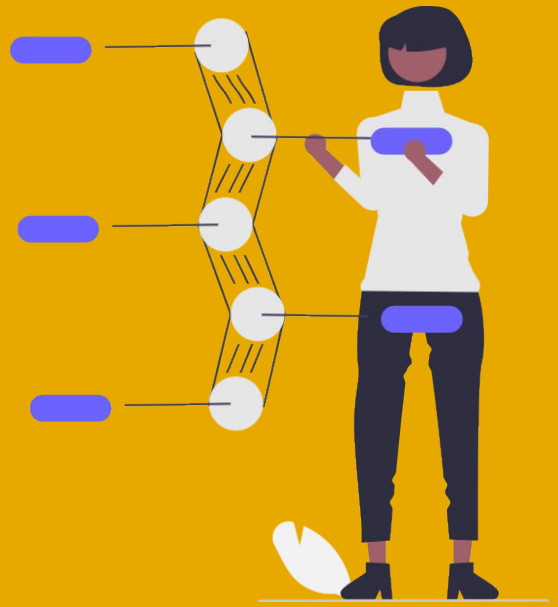
Indicadores Clave del Informe

Métrica	Descripción
Performance	Mide tiempos de carga, interactividad y estabilidad visual.
PWA	Verifica si la app es instalable y usa Service Workers.
SEO	Evalúa si la PWA sigue buenas prácticas para indexación.
Accessibility	Revisa si la app es accesible para usuarios con discapacidad.
Best Practices	Confirma el uso de HTTPS, seguridad y compatibilidad móvil.

Mejorando los Resultados de Lighthouse

- Usa **carga diferida (lazy loading)** para imágenes y scripts.
- Habilita **Gzip** o **Brotli** para comprimir recursos.
- Optimiza el **uso del caché con Service Workers**.
- Asegura que la PWA tenga **HTTPS** habilitado.

Resumen de lo aprendido 🧑‍🎓



Adalid aspasia

Resumen de lo aprendido

- Las PWA pueden almacenar datos con LocalStorage, SessionStorage e IndexedDB, entre otras opciones.
- Service Workers permiten el almacenamiento en caché y mejoran la experiencia offline.
- Lighthouse es una herramienta clave para auditar el rendimiento y optimización de una PWA.
- Una PWA bien construida debe ser rápida, confiable e instalable en múltiples dispositivos.

GRACIAS POR TU ATENCIÓN

Nos vemos en la próxima clase

Chile
Adalid
aspasia
LA FORMACIÓN DE TU FUTURO