Módulo 6
Desarrollo de aplicaciones Web Progresivas (PWA)

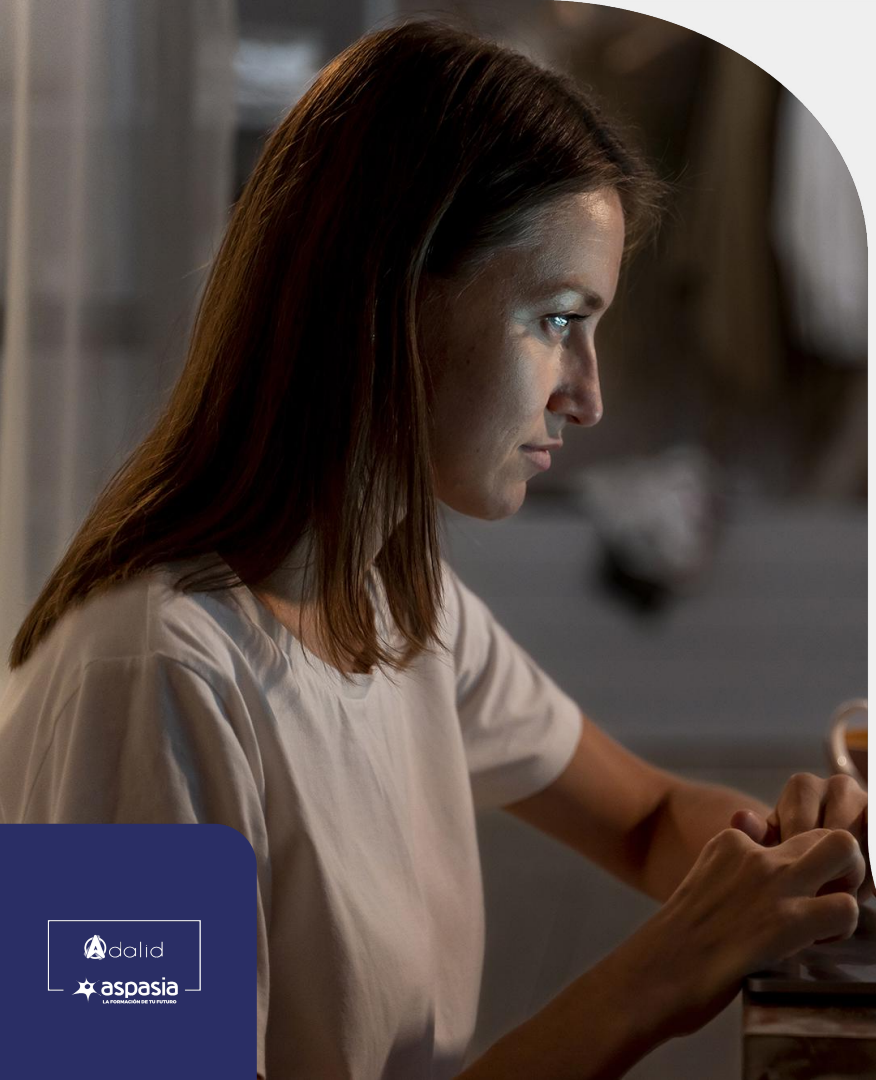
Introducción a las Aplicaciones PWA

Módulo 6

AE 1.1

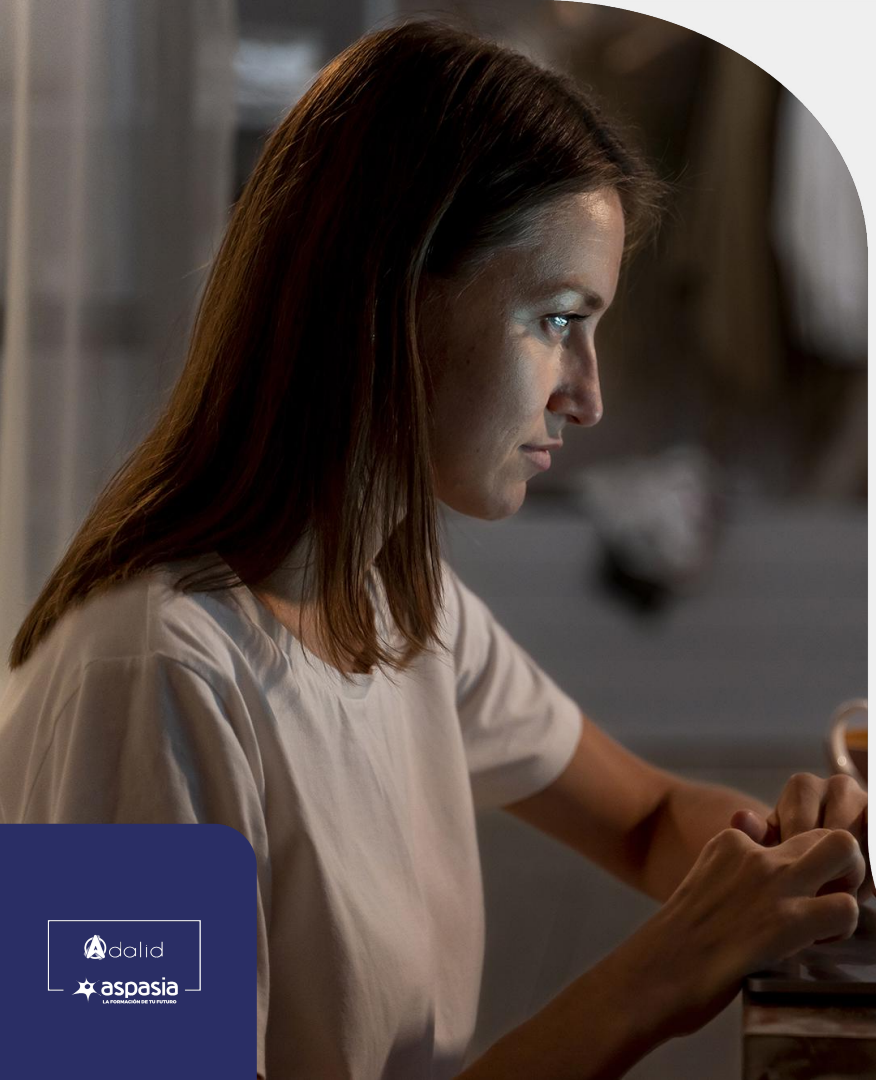
OBJETIVOS

Entender qué es una PWA, sus características, beneficios y limitaciones, aprender a configurar el Manifiesto y el Service Worker, gestionar la caché y la red con diferentes estrategias, y optimizar una aplicación React para funcionar como una PWA segura y eficiente.



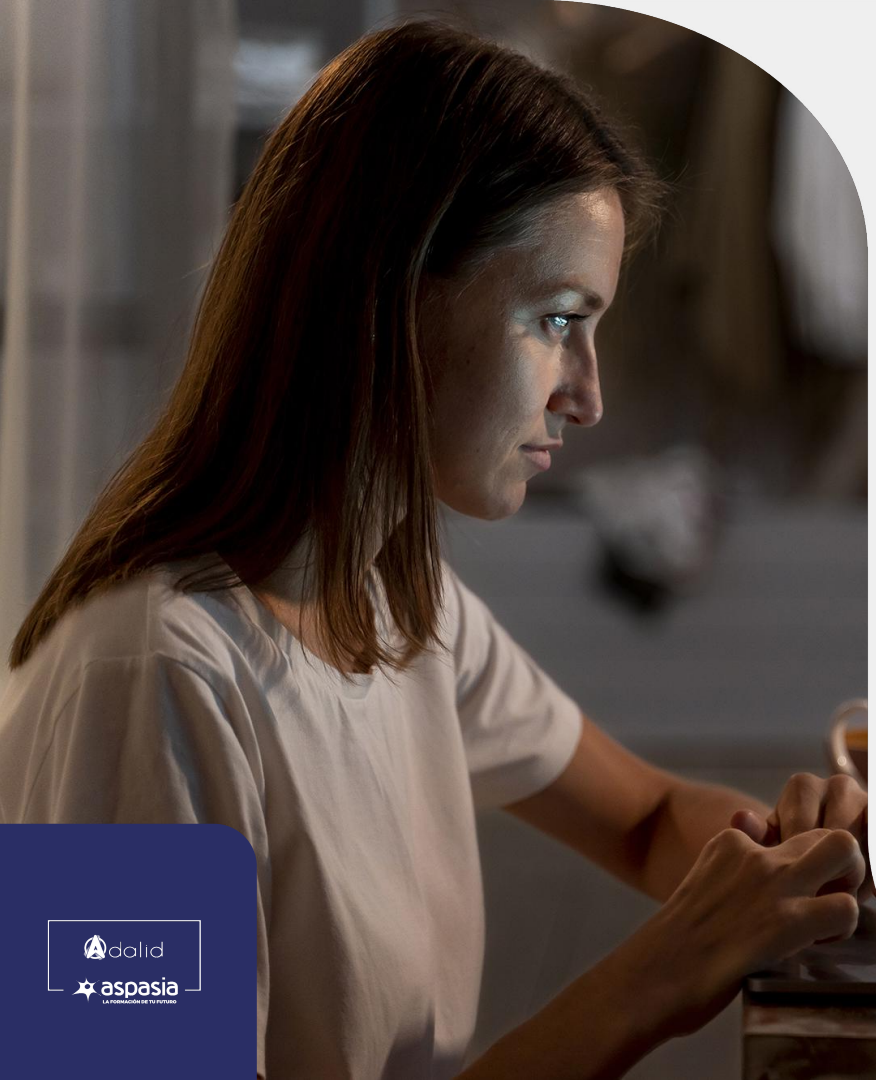
¿QUÉ VAMOS A VER?

- Introducción a las Aplicaciones PWA.
- Qué es unaPWA(Progressive Web Application).
- Características de una PWA (Progresiva,responsiva, adaptable, segura, independiente).
- Beneficios de una PWA.
- Limitaciones de una PWA.
- Diferencias entre una PWA, una aplicación Web tradicional y una aplicación Nativa.
- Arquitectura y componentes de una PWA (Service Workers, Manifiesto, Shell de la aplicación).



¿QUÉ VAMOS A VER?

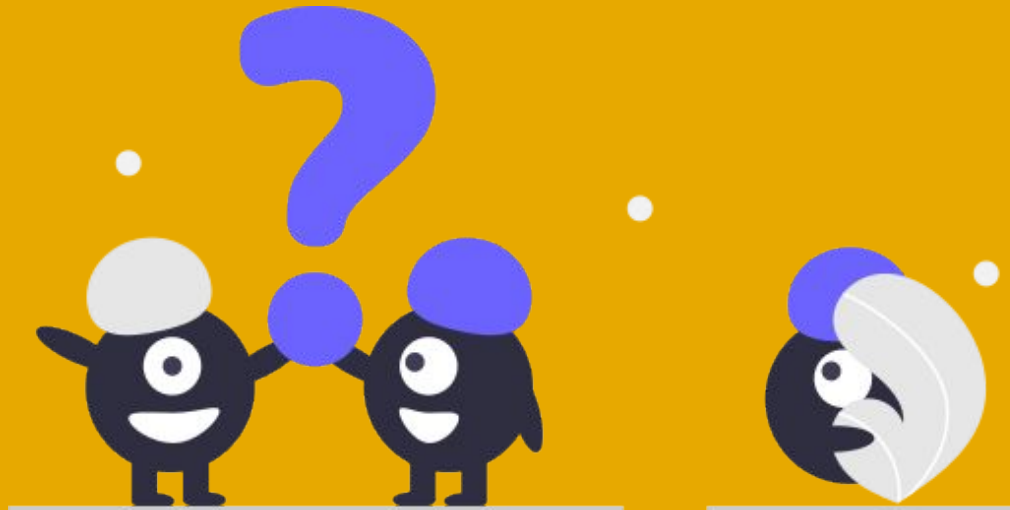
- Frameworks que soportan el desarrollo de PWA's.
- El Manifiesto.
- Qué es el Manifiesto.
- Para qué se usa el Manifiesto.
- Estructura de un archivo de Manifiesto.
- El Service Worker.
- Qué es el ServiceWorker.
- Para qué se usa el Service Worker.
- Ventajas de usar un Service Worker.
- Descripción general de un Service Worker (API asíncrona, API basada en eventos, precaching, aislamiento del hilo principal).

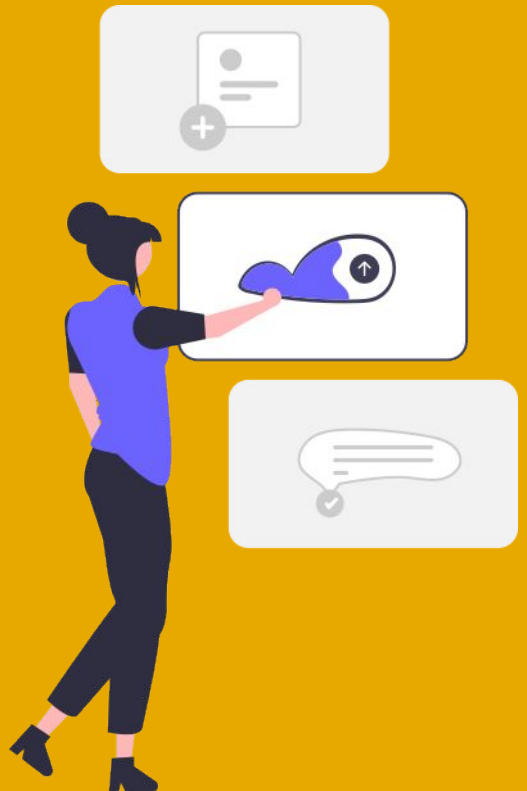


¿QUÉ VAMOS A VER?

- Ciclo de vida de un Service Worker.
- Cómo interactúa el Service Worker con el caché y el acceso a la red.
- Cómo configurar Service Worker para ReactJs.
- Funcionamiento de una PWA con HTTPS.
- Estrategias de almacenamiento en caché de Service Worker.
    - Stale-While-Revalidate.
    - Cache-first.
    - NetworkFirst.
    - CacheOnly.
    - NetworkOnly.
    - CacheAndNetwork.

¿Qué es una PWA?





Introducción a las Aplicaciones PWA

¿Qué es una PWA (Progressive Web Application)?

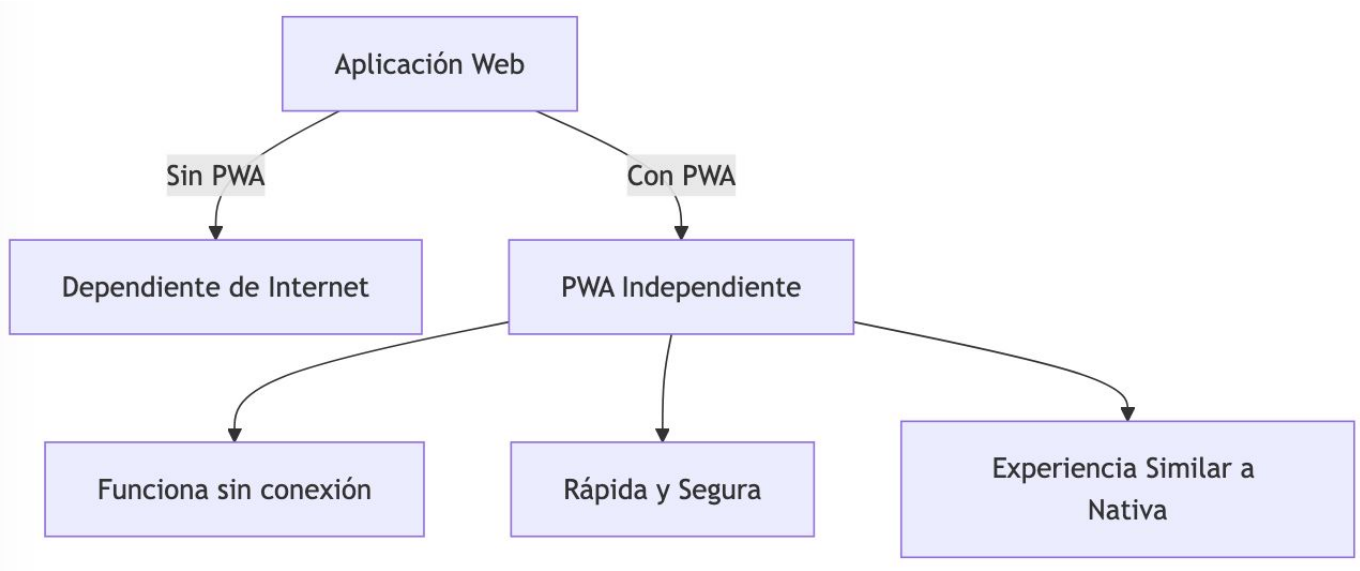
Una Progressive Web Application (PWA) es una aplicación web que combina las mejores características de las **aplicaciones web tradicionales** y **las aplicaciones nativas**, ofreciendo una experiencia rápida, confiable y accesible en cualquier dispositivo.

- Funciona sin conexión o con conectividad limitada.
- Se instala en dispositivos móviles y de escritorio.
- Mejora el rendimiento con almacenamiento en caché.
- Garantiza seguridad y rapidez en la navegación.

Características de una PWA

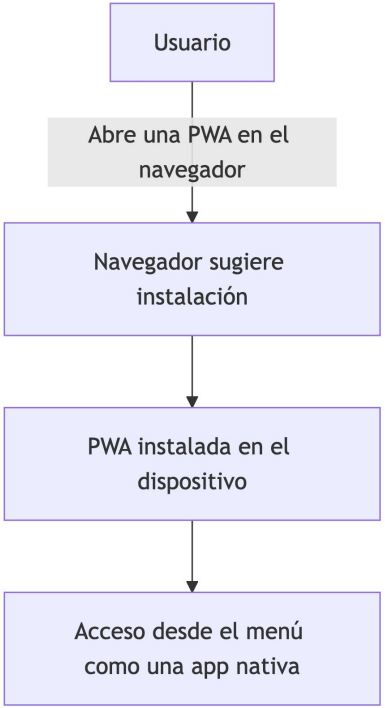
- Funciona en cualquier navegador moderno, sin importar la plataforma.
- Se adapta a cualquier tamaño de pantalla o dispositivo.
- Puede instalarse en el escritorio o móvil como una aplicación nativa.
- Siempre usa HTTPS para evitar ataques de seguridad.
- Gracias a Service Workers, puede funcionar sin conexión.

Características de una PWA



Beneficios de una PWA

- No requiere instalación desde una tienda de apps.
- Carga rápida, incluso en redes lentas.
- Reduce el consumo de datos móviles gracias al caché.
- Aumenta la conversión y retención de usuarios en sitios web.
- Compatibilidad con múltiples dispositivos.



Limitaciones de una PWA

- **No tiene acceso completo a todas las funcionalidades del sistema**, como Bluetooth avanzado o ciertos sensores.
- **Dependencia del navegador**, ya que las PWA requieren compatibilidad con Service Workers.
- **Algunas plataformas restringen funcionalidades** (Ejemplo: Apple limita Service Workers en iOS).
- **No aparecen en tiendas de aplicaciones** (aunque Google Play permite algunas PWA).

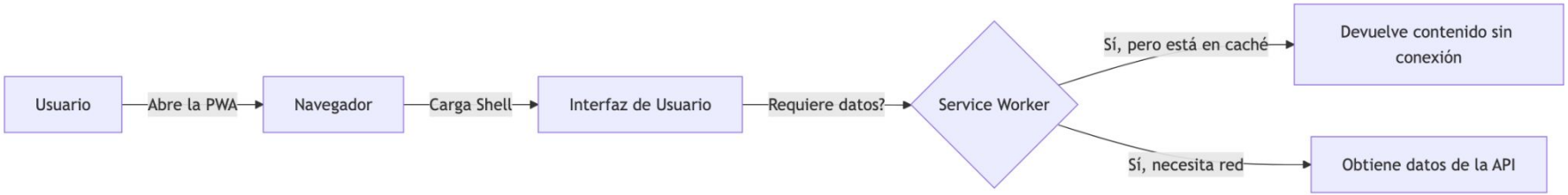
Diferencias entre una PWA, una Web Tradicional y una App Nativa

Característica	Web Tradicional	PWA	Aplicación Nativa
Acceso desde navegador	✔️ Sí	✔️ Sí	❌ No
Funciona sin conexión	❌ No	✔️ Sí	✔️ Sí
Instalable en dispositivos	❌ No	✔️ Sí	✔️ Sí
Acceso a APIs avanzadas	❌ Limitado	🔸 Parcial	✔️ Total
Publicación en tiendas	❌ No	🔸 Algunas	✔️ Sí
Actualizaciones automáticas	✔️ Sí	✔️ Sí	❌ No

Arquitectura y Componentes de una PWA

- **Service Workers**
  - Script que actúa como proxy entre la red y la aplicación.
  - Permite cargar contenido sin conexión y mejorar el rendimiento.
  - Funciona en segundo plano sin afectar la interfaz.
- **Manifiesto**
  - Archivo JSON que define la apariencia y el comportamiento de la PWA.
  - Permite personalizar el icono, nombre y colores de la app.
- **Shell de la Aplicación**
  - Estructura mínima que permite cargar la interfaz rápidamente.
  - Se almacena en caché para que la aplicación se abra al instante.

Arquitectura y Componentes de una PWA



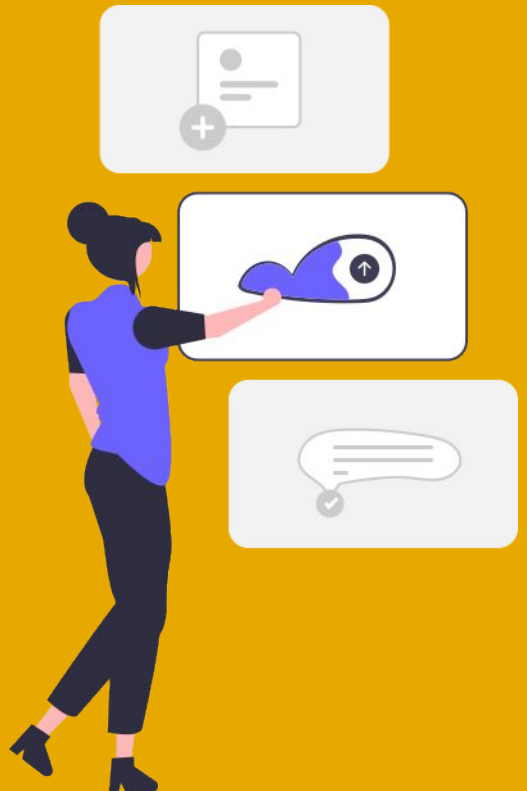
Frameworks que Soportan el Desarrollo de PWA’s

- **React.js**
  - Permite crear PWA’s con herramientas como create-react-app y Vite.
- **Angular**
  - Tiene soporte nativo para PWA’s con @angular/pwa.
- **Vue.js**
  - Puede convertir una aplicación en PWA con vue-pwa.
- **Next.js**
  - Ofrece optimización de PWA con el paquete next-pwa.

Frameworks que Soportan el Desarrollo de PWA’s

Ejemplo de Registro de Service Worker en React:

```
if ('serviceWorker' in navigator) {
  navigator.serviceWorker.register('/sw.js')
    .then(() => console.log('Service Worker registrado'))
    .catch(error => console.log('Error en Service Worker', error));
}
```



El Manifiesto en una PWA

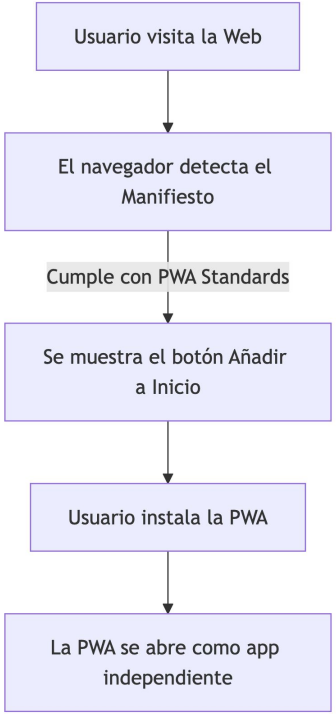
¿Qué es el Manifiesto en una PWA?

El Manifiesto Web App **(manifest.json)** es un archivo JSON que proporciona **metadatos clave** sobre una PWA. Define cómo se comporta la aplicación cuando se instala en un dispositivo, incluyendo:

- Nombre, descripción e iconos
- Modo de visualización (pantalla completa, minimalista, navegador, etc.)
- Colores y temas personalizados
- Página de inicio predeterminada

¿Para qué se usa el Manifiesto?

- Permitir que la PWA se instale en la pantalla de inicio como una app nativa.
- Definir la experiencia del usuario al abrir la aplicación.
- Personalizar la apariencia y comportamiento de la aplicación.



Estructura de un Archivo de Manifiesto

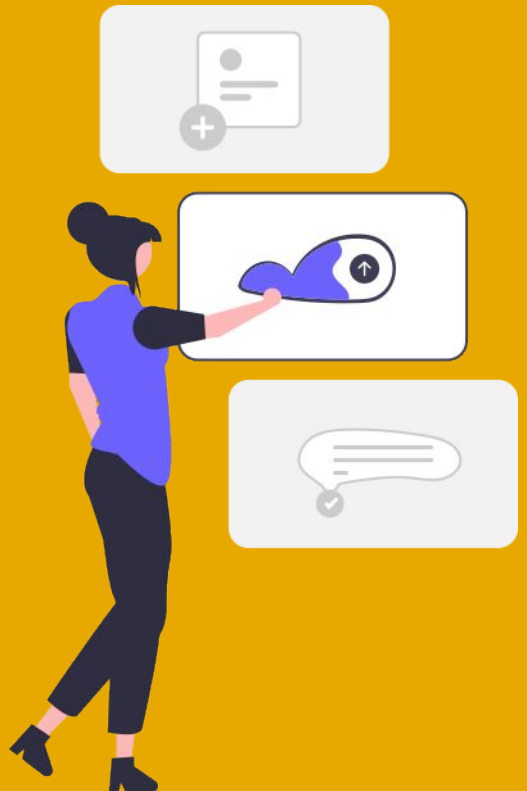
Cómo Agregar el Manifiesto a una PWA

1. Crear el Archivo **manifest.json** en la carpeta **public/**.
2. Enlazar el Manifiesto en **index.html** dentro de la etiqueta **<head>**.
3. Comprobar que el **Navegador** lo Reconoce ingresando al **inspector de elementos.**

```
{
  "name": "Mi PWA",
  "short_name": "PWA",
  "start_url": "/index.html",
  "display": "standalone",
  "background_color": "#ffffff",
  "theme_color": "#2196f3",
  "icons": [
    {
      "src": "/icons/icon-192x192.png",
      "sizes": "192x192",
      "type": "image/png"
    },
    {
      "src": "/icons/icon-512x512.png",
      "sizes": "512x512",
      "type": "image/png"
    }
  ]
}
```

Estructura de un Archivo de Manifiesto

Propiedad	Descripción
name	Nombre completo de la aplicación.
short_name	Nombre corto que aparece en el icono de inicio.
start_url	Página con la que inicia la aplicación.
display	Define cómo se muestra la app (fullscreen, standalone, minimal-ui, browser).
background_color	Color de fondo al abrir la PWA.
theme_color	Color principal de la app en la barra de herramientas.
icons	Lista de iconos en diferentes tamaños.



El Manifiesto en una PWA

¿Qué es un Service Worker?

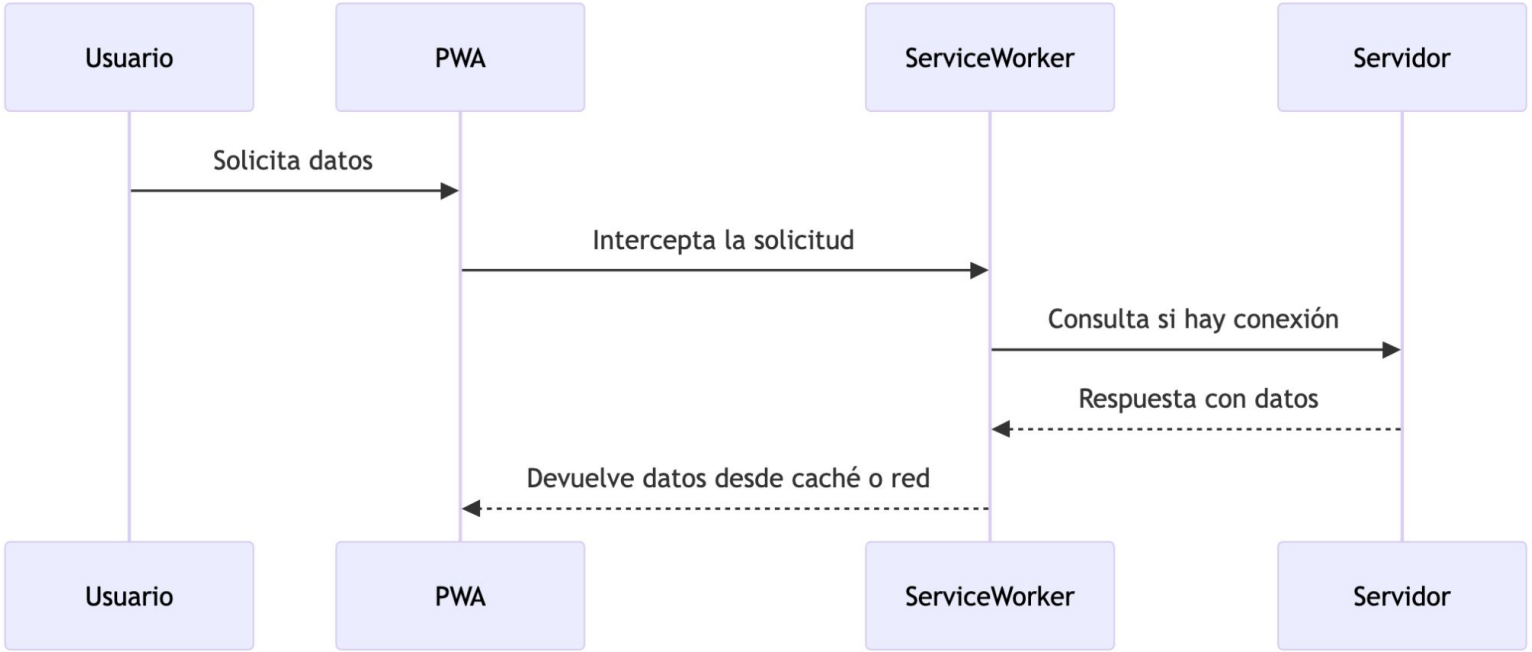
Un Service Worker es un script en segundo plano que actúa como intermediario entre la aplicación y la red. Permite manejar caché, notificaciones push y funcionamiento sin conexión en una PWA.

- Funciona en segundo plano y no bloquea la UI.
- Intercepta y maneja peticiones de red.
- Permite el almacenamiento en caché.
- Requiere HTTPS por seguridad.

¿Para qué se usa el Service Worker?

- Permitir que una aplicación funcione sin conexión.
- Mejorar el rendimiento con almacenamiento en caché.
- Enviar notificaciones push.
- Hacer que la PWA se sienta como una app nativa.

¿Para qué se usa el Service Worker?



Ventajas de Usar un Service Worker

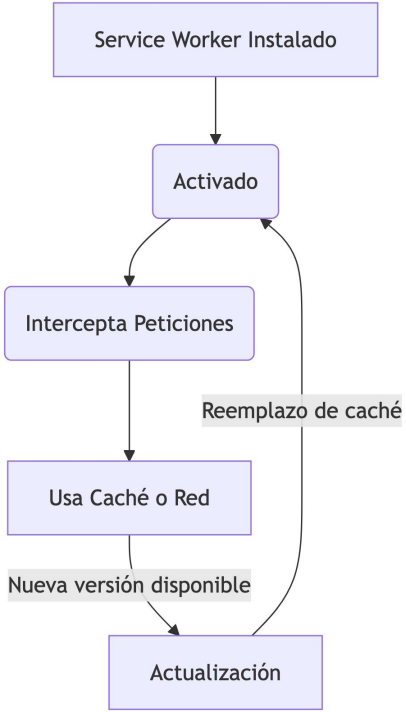
- **Mejor rendimiento:** Carga instantánea desde caché.
- **Funcionalidad sin conexión:** Acceso a contenido sin internet.
- **Menos consumo de datos:** Reduce peticiones al servidor.
- **Experiencia nativa:** Interacciones fluidas y rápidas.

Descripción General de un Service Worker

- **API Asíncrona:**
  - Usa Promesas (fetch(), caches.open()), lo que evita bloqueos en la UI.
- **API Basada en Eventos:**
  - Escucha eventos como install, activate, y fetch.
- **Precaching:**
  - Descarga y almacena recursos al instalarse.
- **Aislamiento del Hilo Principal:**
  - No afecta la ejecución de la aplicación.

Ciclo de Vida de un Service Worker

1. **Instalación (install)**
   a. Se registra el Service Worker.
2. **Activación (activate)**
   a. Se eliminan cachés antiguas.
3. **Intercepción de Peticiones (fetch)**
   a. Se gestionan solicitudes de red y caché.
4. **Actualización**
   a. Se activa un nuevo Service Worker cuando hay cambios.



Configuración de un Service Worker en ReactJS

- **Crear el Archivo sw.js**
  - Ubicación: public/sw.js

```
self.addEventListener('install', event => {
  event.waitUntil(
    caches.open('pwa-cache-v1').then(cache => {
      return cache.addAll([
        '/',
        '/index.html',
        '/styles.css',
        '/app.js',
        '/icon.png'
      ]);
    })
  );
});

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

Configuración de un Service Worker en ReactJS

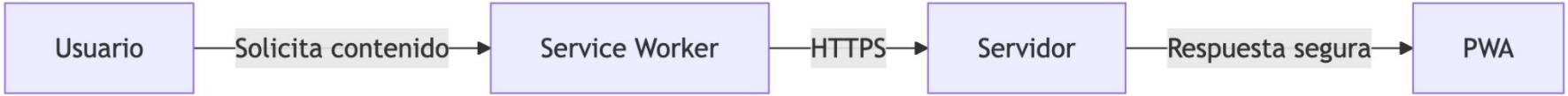
- **Registrar el Service Worker en index.js**

```
if ('serviceWorker' in navigator) {
  navigator.serviceWorker.register('/sw.js')
    .then(() => console.log('Service Worker registrado'))
    .catch(error => console.log('Error en Service Worker', error));
}
```

- **Verificar la Instalación en Chrome DevTools**
  - Abre DevTools (F12 o Ctrl + Shift + I).
  - Ve a la pestaña “Application” > “Service Workers”.
  - Verifica que el Service Worker está activado.

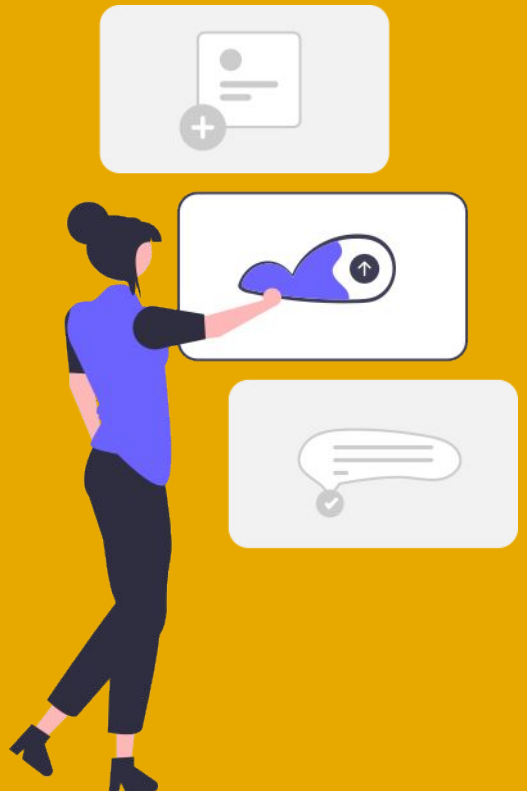
Funcionamiento de una PWA con HTTPS

- **Los Service Workers solo funcionan con HTTPS,** ya que requieren un entorno seguro.
- Para pruebas locales, **localhost es una excepción** y permite ejecutarlos.
- En producción, **se necesita un certificado SSL.**



Estrategias de Almacenamiento en Caché de Service Worker

Estrategia	Descripción
Stale-While-Revalidate	Usa caché primero y actualiza en segundo plano.
Cache-first	Usa caché si está disponible, si no, hace una solicitud de red.
NetworkFirst	Intenta cargar desde la red, si falla, usa caché.
CacheOnly	Solo usa caché, sin red.
NetworkOnly	Solo usa red, sin caché.
CacheAndNetwork	Usa caché y red simultáneamente.



Almacenamiento en Caché

Estrategias de Almacenamiento en Caché con Service Worker

Los Service Workers pueden manejar las peticiones de red de diferentes maneras, dependiendo de la **estrategia de caché implementada**.

Cada estrategia tiene un uso óptimo dependiendo de los requisitos de rendimiento, disponibilidad y actualización de la aplicación.

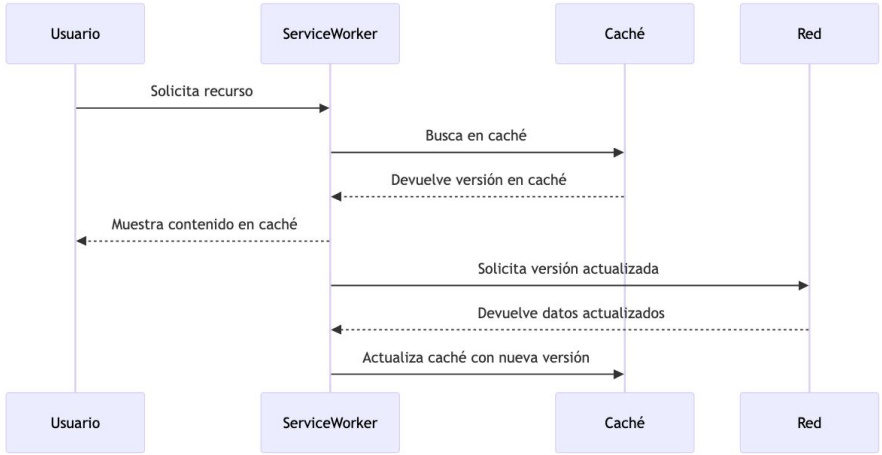
Estrategias de Almacenamiento en Caché con Service Worker

Estrategia	Uso Ideal	Desventaja
Stale-While-Revalidate	Contenido frecuente pero no crítico (blogs).	Puede mostrar contenido desactualizado temporalmente.
Cache-First	Recursos estáticos (CSS, imágenes).	Puede mostrar contenido viejo si no se actualiza.
Network-First	Datos en tiempo real (noticias, chats).	Depende de la conexión, puede fallar si no hay red.
Cache-Only	Aplicaciones offline puras.	No permite contenido nuevo sin actualizar la caché manualmente.
Network-Only	Datos dinámicos (autenticación).	No funciona sin internet.
Cache-And-Network	Mezcla de rendimiento y actualización.	Puede hacer peticiones dobles (consumo de datos).

Estrategias de Almacenamiento en Caché con Service Worker

Estrategia: Stale-While-Revalidate

- Devuelve inmediatamente la versión en caché (si existe).
- Mientras tanto, obtiene una versión más nueva de la red y la actualiza en la caché.
- **Ideal para:** Contenido que cambia, pero no es crítico mostrar siempre la última versión inmediatamente (ejemplo: noticias, blogs).



Estrategias de Almacenamiento en Caché con Service Worker

Estrategia: Stale-While-Revalidate

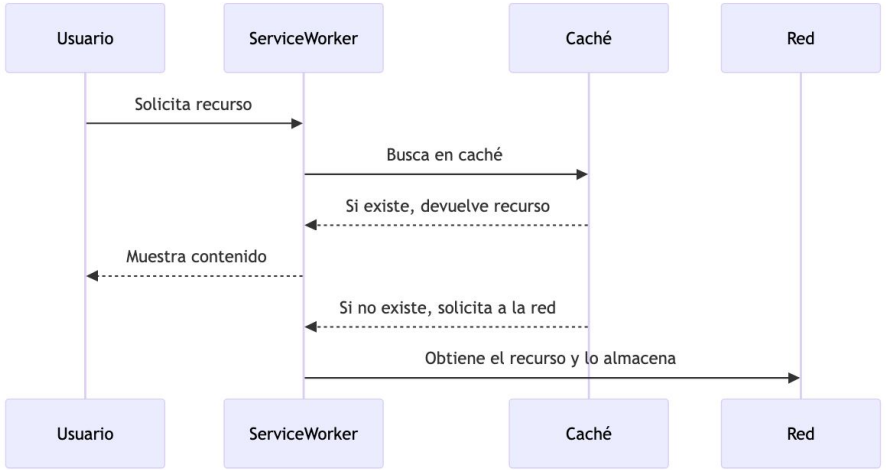
- Implementación en sw.js

```
self.addEventListener('fetch', event => {
  event.respondWith(
    caches.open('pwa-cache').then(cache => {
      return cache.match(event.request).then(response => {
        const fetchPromise = fetch(event.request).then(networkResponse => {
          cache.put(event.request, networkResponse.clone());
          return networkResponse;
        });
        return response || fetchPromise;
      });
    })
  );
});
```

Estrategias de Almacenamiento en Caché con Service Worker

Estrategia: Cache-First (Caché Primero)

- Busca primero en caché y, si el recurso está disponible, lo devuelve.
- Sii no está en caché, lo solicita a la red.
- **Ideal para:** Recursos estáticos como imágenes, hojas de estilo y scripts.



Estrategias de Almacenamiento en Caché con Service Worker

Estrategia: Cache-First (Caché Primero)

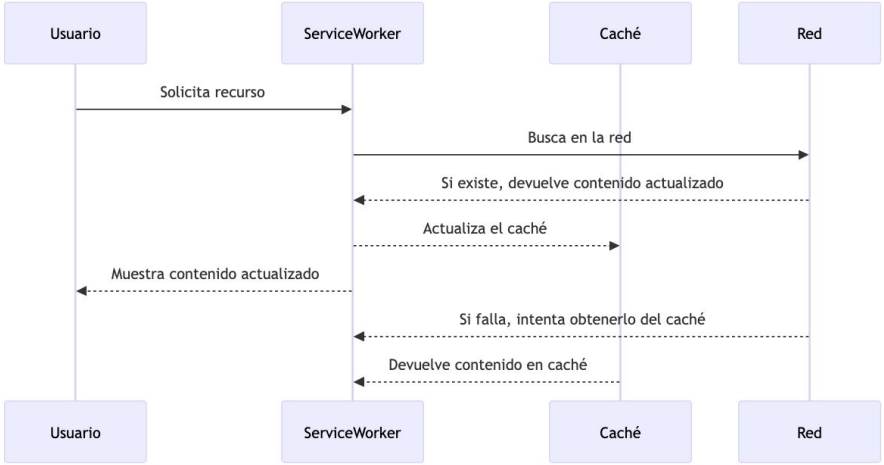
- Implementación en sw.js

```
self.addEventListener('fetch', event => {
  event.respondWith(
    caches.match(event.request).then(response => {
      return response || fetch(event.request).then(networkResponse => {
        return caches.open('pwa-cache').then(cache => {
          cache.put(event.request, networkResponse.clone());
          return networkResponse;
        });
      });
    })
  );
});
```

Estrategias de Almacenamiento en Caché con Service Worker

Estrategia: Network-First (Red Primero)

- Intenta obtener siempre el contenido más reciente desde la red.
- Si la red falla, usa la versión en caché.
- **Ideal para:** Contenido dinámico y crítico como datos de usuario o información en tiempo real.



Estrategias de Almacenamiento en Caché con Service Worker

Estrategia: Network-First (Red Primero)

- Implementación en sw.js

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

Estrategias de Almacenamiento en Caché con Service Worker

Estrategia: Cache-Only (Solo Caché)

- Devuelve siempre el contenido desde la caché.
- **Ideal para:** Aplicaciones completamente offline o donde la red no es necesaria.

    - Implementación en sw.js

```
self.addEventListener('fetch', event => {
  event.respondWith(
    caches.match(event.request)
  );
});
```

Estrategias de Almacenamiento en Caché con Service Worker

Estrategia: Network-Only (Solo Red)

- No almacena nada en caché, todas las peticiones se hacen a la red.
- **Ideal para:** Datos altamente dinámicos como autenticación o pagos.

    - Implementación en sw.js

```
self.addEventListener('fetch', event => {
  event.respondWith(fetch(event.request));
});
```

Estrategias de Almacenamiento en Caché con Service Worker

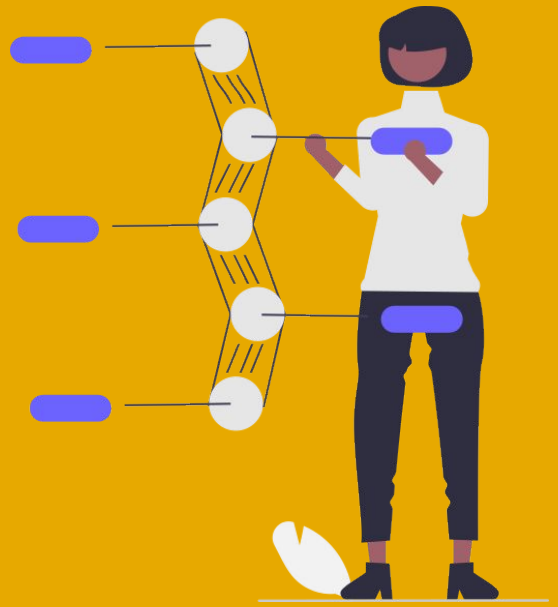
Estrategia: Cache-And-Network (Combinación de Ambas)

- Usa caché y red simultáneamente.
- La primera respuesta viene del caché, pero la red actualiza el contenido en segundo plano.

```
self.addEventListener('fetch', event => {
  event.respondWith(
    caches.match(event.request).then(response => {
      let fetchRequest =
fetch(event.request).then(networkResponse => {
        caches.open('pwa-cache').then(cache => {
          cache.put(event.request,
networkResponse.clone());
        });
        return networkResponse;
      });

      return response || fetchRequest;
    })
  );
});
```

Resumen de lo aprendido 🧑‍🎓



Adalid aspasia

Resumen de lo aprendido

- **Entendiste qué es una PWA**, sus características, beneficios y diferencias con apps web y nativas.
- **Aprendiste a usar el Manifiesto**, configurándolo para personalizar e instalar la PWA.
- **Exploraste los Service Workers**, su ciclo de vida y su rol en la caché y el rendimiento.
- **Implementaste estrategias de almacenamiento en caché**, optimizando la velocidad y funcionalidad offline.

GRACIAS POR TU ATENCIÓN

Nos vemos en la próxima clase

Chile
Adalid
aspasia
LA FORMACIÓN DE TU FUTURO