Instituto Politécnico Nacional

Escuela Superior de Computo

Ingeniería en Sistemas Computacionales.

Unidad de Aprendizaje:
Sistemas Distribuidos.

Nombre del Profesor:
Carreto Arellano Chadwick.

Nombre del Alumno:

Cruz Cubas Ricardo

Grupo:
7CM1.

Practica 8:

PWA.

Contenido

Introducción

El desarrollo de software y las tecnologías de la información, la evolución constante de los dispositivos móviles y las necesidades cambiantes de los usuarios han generado una creciente demanda por aplicaciones que sean rápidas, accesibles, multiplataforma y funcionales incluso en condiciones de conectividad limitada. Tradicionalmente, esta demanda ha sido abordada mediante el desarrollo de aplicaciones nativas, diseñadas específicamente para sistemas operativos como Android o iOS. Sin embargo, este enfoque presenta desafíos significativos, como altos costos de desarrollo, mantenimiento paralelo para distintas plataformas, y la necesidad de distribuir las aplicaciones a través de tiendas oficiales.

Como respuesta a estas limitaciones, han emergido las Progressive Web Apps (PWA), un enfoque híbrido que aprovecha las capacidades avanzadas del navegador para proporcionar experiencias de usuario comparables a las de las aplicaciones móviles nativas, pero utilizando tecnologías web estándar. El término "Progressive Web App" fue acuñado por Google en 2015, y desde entonces ha ganado una amplia aceptación en la industria y la academia por su flexibilidad, eficiencia y potencial de escalabilidad.

Las PWA se caracterizan por ser aplicaciones web que implementan una serie de principios y tecnologías diseñadas para mejorar progresivamente la experiencia del usuario. Entre sus componentes esenciales se encuentran:

- Service Workers: scripts que se ejecutan en segundo plano, permitiendo funcionalidades como el almacenamiento en caché, la sincronización en segundo plano y el funcionamiento offline.
- Web App Manifest: archivo JSON que proporciona metadatos sobre la aplicación (nombre, iconos, orientación, pantalla de inicio, etc.), facilitando su instalación en el dispositivo del usuario.
- Responsive Design: diseño adaptable a distintos tamaños de pantalla, desde dispositivos móviles hasta escritorios.
- HTTPS: protocolo seguro obligatorio para garantizar la integridad de la comunicación entre el navegador y el servidor.
- Actualizaciones automáticas y experiencia fluida, sin necesidad de descargar o instalar paquetes desde tiendas de aplicaciones.

Estas características convierten a las PWA en una solución ideal para entornos donde la conectividad es intermitente o limitada, así como para proyectos que buscan un menor costo de desarrollo y un mantenimiento unificado. Además, su capacidad de ser indexadas por motores de búsqueda

les otorga ventajas en términos de accesibilidad y posicionamiento frente a las aplicaciones nativas.

Desde una perspectiva académica, las PWA representan una innovación significativa en varias áreas del conocimiento, incluyendo la arquitectura de software, el diseño de interfaces de usuario, la experiencia de usuario (UX), el desarrollo multiplataforma, y la ingeniería de requisitos. Además, constituyen un objeto de estudio importante para la evaluación del impacto de nuevas tecnologías en la accesibilidad digital y la inclusión tecnológica, especialmente en regiones con acceso limitado a infraestructura tecnológica avanzada.

Las Progressive Web Apps se consolidan como una alternativa tecnológica eficaz y sostenible frente a las aplicaciones tradicionales. Su adopción creciente por parte de empresas, instituciones educativas y desarrolladores independientes confirma su relevancia en el panorama actual del desarrollo de software. A medida que los navegadores continúan incorporando nuevas capacidades y estándares, se espera que las PWA amplíen aún más su funcionalidad y adopción, consolidándose como una herramienta clave en la construcción de experiencias digitales modernas y accesibles.

Planteamiento del Problema

En la era digital, la visualización, organización y acceso a contenidos multimedia, especialmente imágenes, se ha vuelto una necesidad común tanto para usuarios individuales como para profesionales de áreas como el diseño gráfico, la fotografía, la publicidad o la educación. Sin embargo, muchas de las soluciones actuales dependen de aplicaciones móviles nativas o plataformas en línea que requieren conexión constante a internet, instalación a través de tiendas oficiales, o incluso la creación de cuentas para su uso, lo que limita su accesibilidad y disponibilidad en ciertos contextos.

Por otra parte, las aplicaciones web tradicionales presentan limitaciones significativas cuando se desea utilizarlas en dispositivos móviles: no siempre ofrecen una experiencia fluida, carecen de funcionalidades offline, y no permiten su instalación directa como si se tratara de una app nativa. Esto reduce su utilidad en entornos con conectividad limitada o para usuarios que requieren acceso rápido y frecuente a sus contenidos sin depender del navegador.

Ante esta situación, surge la necesidad de desarrollar una aplicación de galería de imágenes que combine las ventajas de las aplicaciones web con la funcionalidad de las aplicaciones nativas, permitiendo su uso en distintos dispositivos, su instalación sin intermediarios y su funcionamiento offline. En este contexto, las Progressive Web Apps (PWA) ofrecen una solución viable, ya que permiten que una aplicación web se comporte como una app instalable, rápida, segura y disponible incluso sin conexión a internet.

El problema radica en que aún existen pocas soluciones de galerías de imágenes que estén desarrolladas bajo este enfoque tecnológico, lo cual limita la experiencia de usuarios que desean acceder a sus imágenes en diferentes plataformas, sin importar el estado de la red o la disponibilidad de una tienda de aplicaciones. Es por ello que se plantea el desarrollo de una aplicación tipo galería, accesible desde cualquier navegador moderno, pero que también pueda instalarse como una PWA, permitiendo al usuario visualizar, organizar y acceder a sus imágenes de forma eficiente y sin conexión, garantizando así una experiencia multiplataforma, optimizada y accesible.

Propuesta de Solución

Con el fin de abordar las limitaciones identificadas en el acceso y visualización de galerías de imágenes desde múltiples dispositivos y en condiciones de conectividad variable, se propone el desarrollo de una aplicación web progresiva (Progressive Web App, PWA) que funcione como una galería de imágenes multiplataforma, accesible desde navegadores modernos y con la capacidad de instalarse como una aplicación nativa en teléfonos móviles, tabletas o computadoras de escritorio.

La solución consistirá en una aplicación desarrollada utilizando tecnologías web estándar como HTML5, CSS3 y JavaScript, en conjunto con frameworks y bibliotecas modernas como React.js o Vue.js, que faciliten una experiencia de usuario dinámica y responsiva. Para habilitar las características propias de una PWA, se integrará un Service Worker, encargado de gestionar el almacenamiento en caché de los recursos y permitir el funcionamiento de la aplicación sin conexión a internet. Además, se implementará un Web App Manifest, que contendrá los metadatos necesarios para permitir la instalación de la aplicación en el dispositivo del usuario, incluyendo íconos, nombre de la app, colores y orientación preferida.

La arquitectura de la aplicación estará diseñada bajo un enfoque modular y escalable, permitiendo su futura integración con servicios externos como almacenamiento en la nube (Firebase, por ejemplo), autenticación de usuarios o sincronización entre dispositivos.

Esta solución busca no solo mejorar la experiencia del usuario al interactuar con sus contenidos visuales, sino también promover el uso de tecnologías web modernas que aumentan la accesibilidad, reducen la dependencia de tiendas de aplicaciones y disminuyen los costos de desarrollo y mantenimiento. En última instancia, la implementación de esta PWA de galería servirá como un ejemplo práctico de cómo las aplicaciones progresivas pueden ofrecer una experiencia rica, fluida y universal desde la web.

Materiales y Métodos Empleados

Materiales

Para el desarrollo e implementación de la aplicación tipo galería como Progressive Web App (PWA), se utilizaron los siguientes recursos:

a) Software

- Visual Studio Code: editor de código fuente utilizado para el desarrollo del proyecto.
- Google Chrome y Mozilla Firefox: navegadores web para pruebas y verificación de compatibilidad.
- Node.js y npm: entorno de ejecución y gestor de paquetes necesario para el manejo de dependencias y la ejecución del entorno de desarrollo.
- Framework React.js (o alternativamente Vue.js): biblioteca JavaScript empleada para construir una interfaz de usuario dinámica y modular.
- Webpack / Vite: herramienta de empaquetado y optimización de recursos.
- Git: sistema de control de versiones utilizado para el seguimiento del desarrollo.
- GitHub: plataforma de alojamiento de código para la gestión del repositorio del proyecto.
- Lighthouse (Google): herramienta de evaluación de rendimiento, accesibilidad y capacidades PWA.

b) Lenguajes y tecnologías

- HTML5: para la estructura básica de la interfaz.
- CSS3 (y/o TailwindCSS): para la presentación y diseño visual responsivo.
- JavaScript (ES6+): para la lógica del cliente y la interacción del usuario.
- Service Worker API: para la implementación de funcionalidad offline mediante almacenamiento en caché.
- Web App Manifest: para definir los parámetros de instalación como PWA.

c) Hardware

- Computadora personal (procesador i5 o superior, 8 GB de RAM).
- Dispositivos de prueba: smartphone Android (versión 9 o superior), tableta y laptop con distintos navegadores.

Métodos

El desarrollo de la aplicación se llevó a cabo siguiendo un enfoque iterativo e incremental, basado en metodologías ágiles, específicamente una adaptación de Scrum con entregas parciales y revisiones frecuentes.

a) Fases del desarrollo

1. Análisis de requisitos
    - Identificación de las funcionalidades mínimas necesarias (visualización de imágenes, carga local, navegación fluida, uso sin conexión).
    - Estudio de características obligatorias de una PWA.
2. Diseño
    - Estructura de la interfaz gráfica de usuario (GUI) mediante wireframes.
    - Definición de rutas, componentes y estructura del proyecto en React.js.
    - Diseño responsivo adaptado a móviles, tabletas y escritorio.
3. Implementación
    - Desarrollo de componentes funcionales en React.js.
    - Implementación del Service Worker para la funcionalidad offline.
    - Configuración del manifest.json para permitir la instalación.
    - Integración de almacenamiento local (IndexedDB o LocalStorage) para guardar imágenes en caché.
    - Optimización de recursos estáticos para una carga rápida.
4. Pruebas
    - Pruebas funcionales en distintos dispositivos y navegadores.

- Evaluación con Lighthouse para verificar cumplimiento como PWA.
- Pruebas de instalación y comportamiento offline.

5. Despliegue
    - Publicación del sitio web en un servidor HTTPS (por ejemplo, GitHub Pages o Firebase Hosting).
    - Validación de la capacidad de instalación como PWA en Android y escritorio.

Desarrollo

Serviceworker

Este archivo es el corazón que le da capacidades offline y caché a tu PWA usando un service worker. El service worker es un script que corre en segundo plano en el navegador y puede interceptar las solicitudes de red, almacenar recursos en caché y devolverlos cuando no hay conexión.

```
import { clientsClaim } from "workbox-core";
```

Esto es para que el service worker tome control de las pestañas abiertas tan pronto se active, sin esperar a que se recargue la página. Así, el SW empieza a funcionar inmediatamente.

```
clientsClaim();
```

Que es la ejecución de lo anterior, para tomar el control rápido.

```
precacheAndRoute(self.__WB_MANIFEST);
```

Esto es importante: cuando haces el build de tu app con herramientas como Create React App, se genera una lista de archivos (JS, CSS, imágenes, etc.) que deben precachearse, para que estén disponibles offline. Ese self.__WB_MANIFEST es esa lista generada automáticamente. La función precacheAndRoute toma esa lista y cachea esos archivos, además de configurar el service worker para responder a solicitudes con esos recursos cacheados.

```
const fileExtensionRegexp = new RegExp("/[^/?]+\\.[^/]+$");
```

Esto sirve para identificar URLs que terminan en un archivo con extensión, como .js, .png, .html, etc. Sirve para saber cuándo una solicitud es para un archivo estático.

Después registra una ruta con registerRoute que maneja solicitudes de navegación (cuando el usuario cambia de página o escribe una URL):

```
registerRoute(
  ({ request, url }) => {
    if (request.mode !== "navigate") {
      return false;
    }
    if (url.pathname.startsWith("/_")) {
      return false;
    }
    if (url.pathname.match(fileExtensionRegexp)) {
      return false;
    }
    return true;
  },
  createHandlerBoundToURL(process.env.PUBLIC_URL + "/index.html")
);
```

Esto significa que si la solicitud es una navegación (p. ej. usuario pone una URL en el navegador), y no es un archivo estático (no termina en extensión ni empieza con "/_"), entonces el service worker devolverá el archivo index.html. Esto es típico en apps SPA porque siempre debes cargar el archivo base para que React pueda manejar la ruta.

```
registerRoute(
  ({ url }) =>
    url.origin === self.location.origin &&
url.pathname.endsWith(".png"),
  new StaleWhileRevalidate({
    cacheName: "images",
    plugins: [
      new ExpirationPlugin({ maxEntries: 50 }),
    ],
  })
);
```

Esto significa que para todas las imágenes PNG que vienen del mismo servidor (origen igual), usará la estrategia llamada "StaleWhileRevalidate". Es decir, servirá la imagen que está en caché inmediatamente (si existe) para que sea rápido, pero en segundo plano la actualizará con la versión más reciente para que la próxima vez tenga la imagen actualizada. Además, solo guarda hasta 50 imágenes en caché para no saturar el almacenamiento.

```
self.addEventListener("message", (event) => {
  if (event.data && event.data.type === "SKIP_WAITING") {
    self.skipWaiting();
  }
});
```

Esto es para que, cuando recibas un mensaje con tipo "SKIP_WAITING", el service worker activo se salte la fase de espera y tome control inmediatamente. Esto es útil para actualizar la app sin que el usuario tenga que cerrar pestañas.

About.jsx

About.jsx es un componente muy sencillo que regresa un fragmento de JSX con un elemento <main> donde pone un título ESCOM y un párrafo que explica qué es esa escuela. Básicamente, es la sección "Acerca de" de tu app, donde muestras información estática para que el usuario conozca de qué trata ESCOM.

```
export default function About() {
  return (
    <>
      <main>
        <h2>ESCOM</h2>
        <p>La Escuela Superior de Cómputo es una institución educativa
de nivel superior en México, dedicada a la formación de profesionales
en el área de la computación y tecnologías de la información.</p>
      </main>
```

```
    </>
  );
}
```

Gallery.jsx

Regresa un <main> con un título "Galería" y una imagen fija que toma de internet (con una URL en el atributo src). Es para mostrar una sección donde puedes exhibir imágenes o fotos, aunque aquí solo hay una imagen de prueba.

```
export default function Gallery() {
  return (
    <>
      <main>
        <h2>Galería</h2>
        <img src="https://encrypted-
tbn0.gstatic.com/images?q=tbn:ANd9GcQLHgMkYvDmWWq4vgK_t1-
HpPLBG56Q7eDK9w&s" alt="imagen-prueba" />
      </main>
    </>
  );
}
```

Home.jsx

es la pantalla principal de la app, también muy simple. Dentro del <main> muestra varios encabezados con mensajes de bienvenida, el nombre del curso y tu nombre, y un párrafo con un mensaje corto. Esto es para que cuando alguien entre a la página de inicio, vea esa presentación o saludo.

Finalmente, el más complejo es Layout.jsx, que es el componente que envuelve todas las páginas para dar estructura común a la app. Este componente tiene un encabezado con un título “PWA” y un botón que aparece solo si la app detecta que está lista para instalarse (esto ocurre gracias a que usa un estado isReadyForInstall que se activa cuando escucha el evento beforeinstallprompt). Si el usuario presiona ese botón, se lanza el prompt para instalar la app como Progressive Web App.

También tiene una barra de navegación (<nav>) con enlaces que usan React Router para cambiar entre “Inicio”, “Acerca” y “Galería”.

Finalmente, dentro del div principal, muestra el contenido que le pases por props.children, que serán las diferentes páginas (About, Home, Gallery, etc.) dependiendo de la ruta.

```
export default function Home() {
  return (
    <>
      <main>
        <h2>Bienvenido a mi PWA </h2>
        <h2>Sistemas Distribuidos</h2>
```

```
        <h2>Cruz Cubas Ricardo</h2>
        <p>Tarde pero seguro</p>
      </main>
    </>
  );
}
```

ServiceWorkerRegistration.js

Este archivo es el que se encarga de registrar el service worker en el navegador cuando cargas tu aplicación, para que las funcionalidades de PWA, como el cacheo y el trabajo offline, funcionen correctamente.

Primero, define una variable isLocalhost que detecta si estás corriendo la app en tu máquina local (localhost) o en una red local. Esto es útil porque el comportamiento del service worker es un poco diferente cuando desarrollas en local frente a cuando ya la tienes publicada en producción.

Luego, está la función register(config), que se llama para registrar el service worker. Pero solo hace algo si estás en modo producción y el navegador soporta service workers.

Dentro de esta función, se arma la URL del service worker (por defecto, algo como /service-worker.js), y al cargar la ventana (window.addEventListener("load")), se decide qué hacer:

- Si estás en localhost, primero se llama a checkValidServiceWorker(swUrl, config), que básicamente revisa si el archivo del service worker existe y es válido. Esto ayuda a evitar errores cuando estás desarrollando y puedes cambiar mucho.
- Si no estás en localhost, se registra el service worker directamente con registerValidSW(swUrl, config).

La función registerValidSW(swUrl, config) es la que realmente registra el service worker con navigator.serviceWorker.register(swUrl). También añade un listener para cuando detecta que hay una actualización del service worker (cuando se instala uno nuevo), para manejar el ciclo de vida y avisar que hay nuevo contenido disponible.

En caso de que se encuentre un nuevo service worker instalado, pero que la página aún usa el antiguo, avisa que la nueva versión estará activa cuando cierres todas las pestañas.

La función checkValidServiceWorker(swUrl, config) hace un fetch para verificar que el service worker realmente existe en el servidor y es un archivo JS válido. Si no lo encuentra (por ejemplo, borraste el service worker o

cambiaste la app), fuerza una recarga y elimina el service worker viejo para evitar problemas de cache obsoleta.

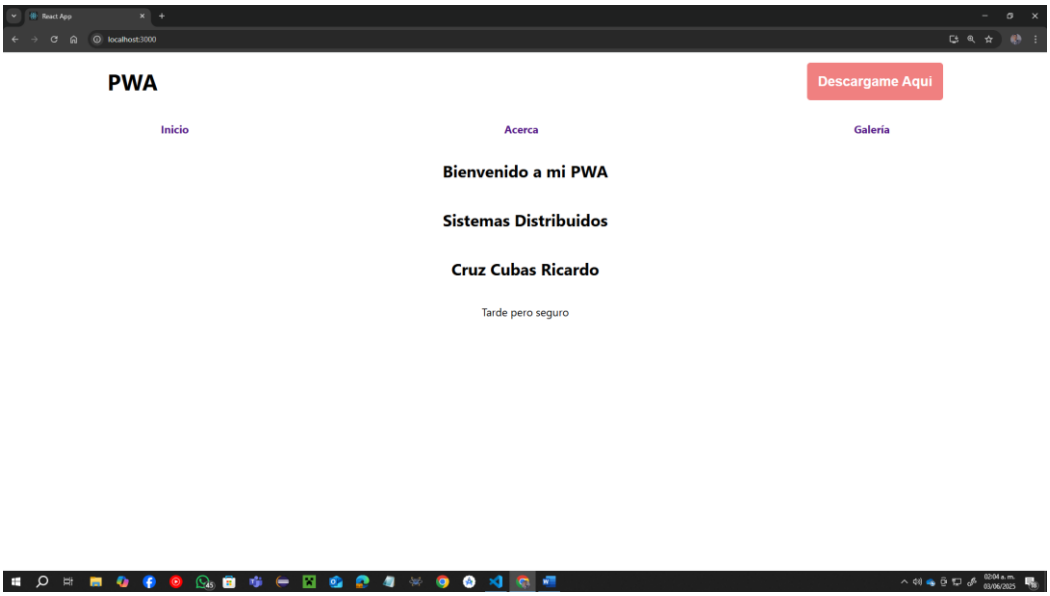
Por último, hay una función unregister() para quitar el service worker, por si quieres desactivar la PWA o hacer pruebas.

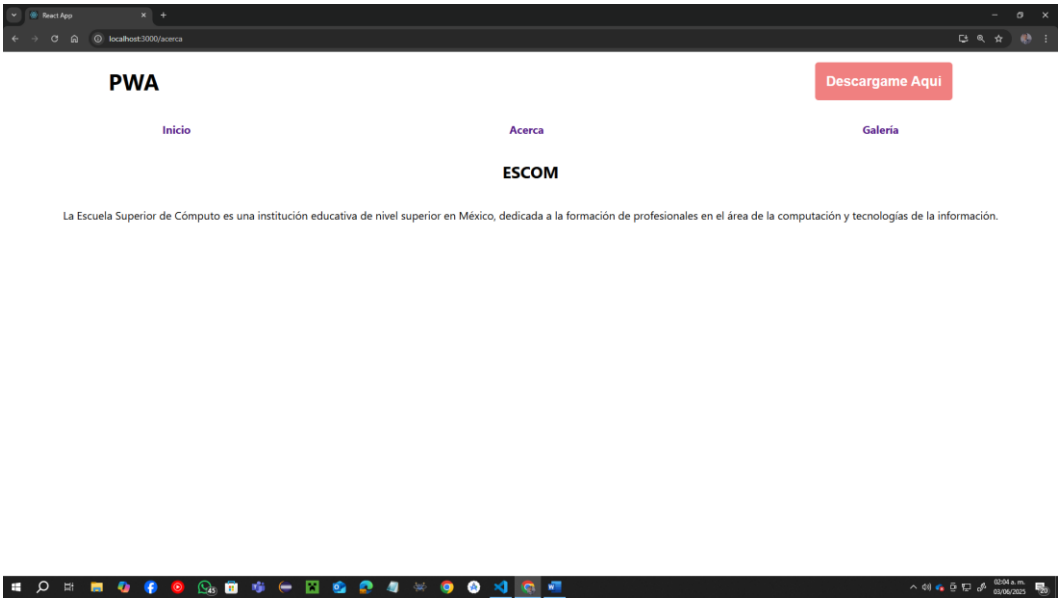
```
const isLocalhost = Boolean(
  window.location.hostname === "localhost" ||
    window.location.hostname === "[::1]" ||
    window.location.hostname.match(
      /^127(?:\.(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)){3}$/
    )
);
export function register(config) {
  if (process.env.NODE_ENV === "production" && "serviceWorker" in
navigator) {
    const publicUrl = new URL(process.env.PUBLIC_URL,
window.location.href);
    if (publicUrl.origin !== window.location.origin) {
      return;
    }
    window.addEventListener("load", () => {
      const swUrl = `${process.env.PUBLIC_URL}/service-worker.js`;

      if (isLocalhost) {
        checkValidServiceWorker(swUrl, config);
        navigator.serviceWorker.ready.then(() => {
          console.log(
            "This web app is being served cache-first by a service " +
              "worker. To learn more, visit https://cra.link/PWA"
          );
        });
      } else {
        registerValidSW(swUrl, config);
      }
    });
  }
}
function registerValidSW(swUrl, config) {
  navigator.serviceWorker
    .register(swUrl)
    .then((registration) => {
      registration.onupdatefound = () => {
        const installingWorker = registration.installing;
        if (installingWorker == null) {
          return;
        }
        installingWorker.onstatechange = () => {
          if (installingWorker.state === "installed") {
            if (navigator.serviceWorker.controller) {
              console.log(
                "New content is available and will be used when all " +
```

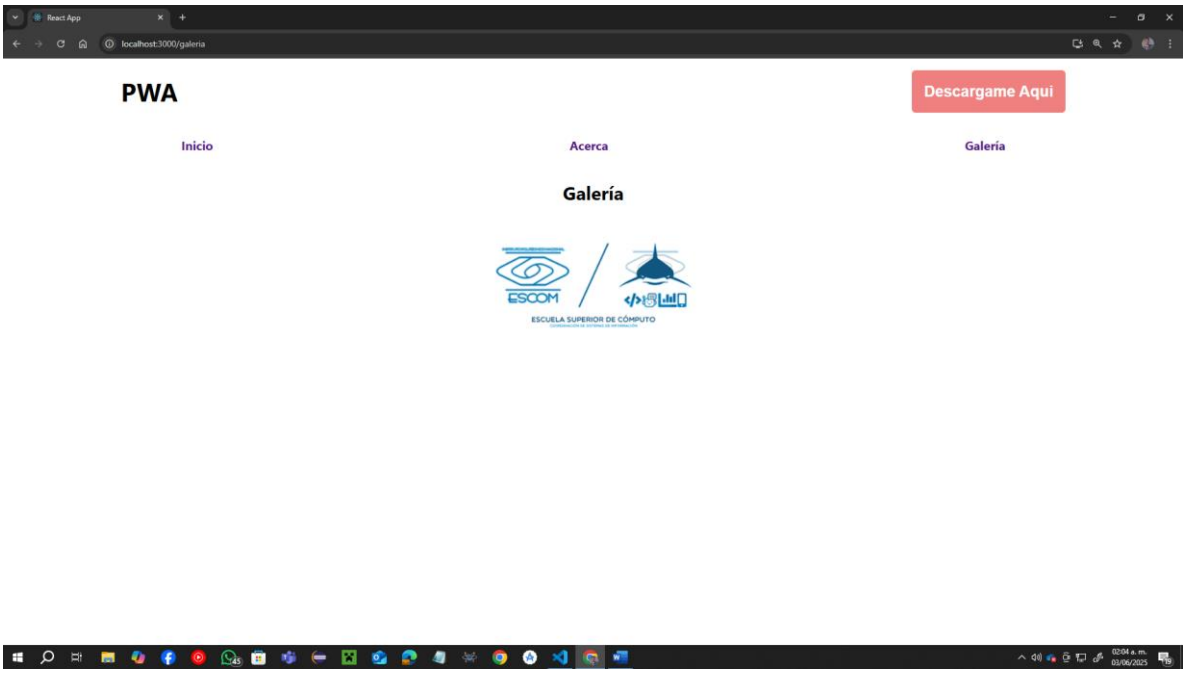
```
                  "tabs for this page are closed. See
https://cra.link/PWA."
                );
                if (config && config.onUpdate) {
                  config.onUpdate(registration);
                }
              } else {
                console.log("Content is cached for offline use.");
                if (config && config.onSuccess) {
                  config.onSuccess(registration);
                }
              }
            }
          };
        };
      })
      .catch((error) => {
        console.error("Error during service worker registration:",
error);
      });
}

function checkValidServiceWorker(swUrl, config) {
  fetch(swUrl, {
    headers: { "Service-Worker": "script" },
  })
    .then((response) => {
      const contentType = response.headers.get("content-type");
      if (
        response.status === 404 ||
        (contentType != null && contentType.indexOf("javascript") === -
1)
      ) {
        navigator.serviceWorker.ready.then((registration) => {
          registration.unregister().then(() => {
            window.location.reload();
          });
        });
      } else {
        registerValidSW(swUrl, config);
      }
    })
    .catch(() => {
      console.log(
        "No internet connection found. App is running in offline mode."
      );
    });
}
export function unregister() {
  if ("serviceWorker" in navigator) {
    navigator.serviceWorker.ready
      .then((registration) => {
        registration.unregister();
```

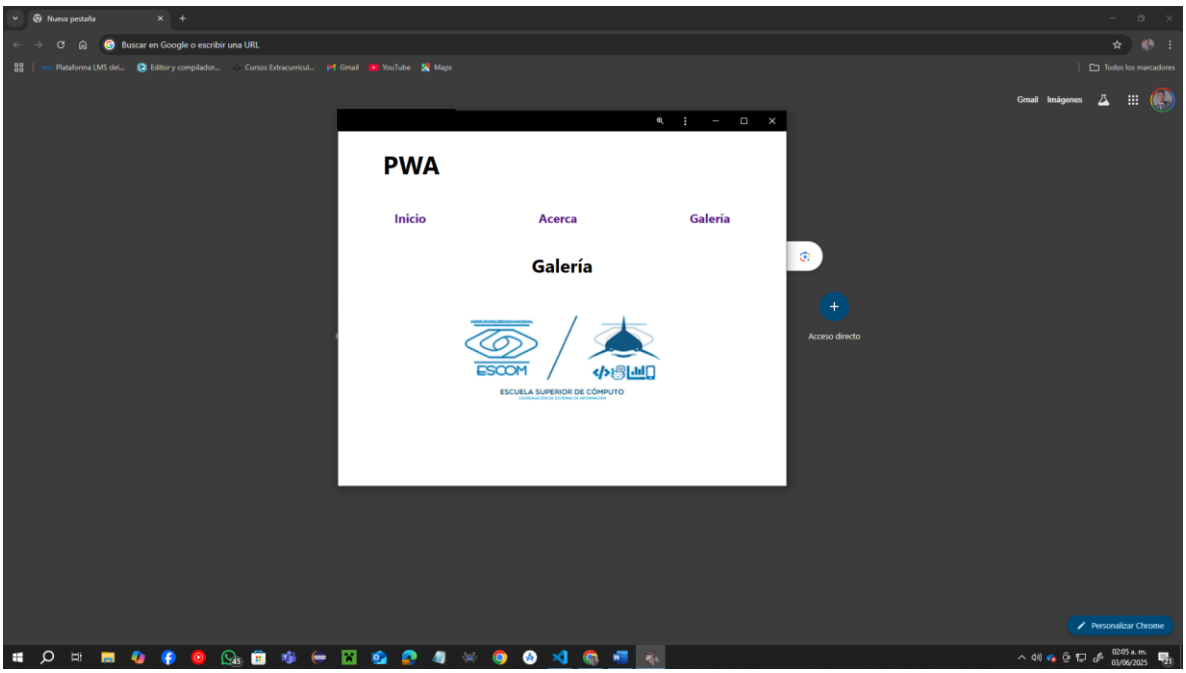
```
        })
        .catch((error) => {
          console.error(error.message);
        });
    }
}
```

Resultados






React App
localhost:3000/galeria
PWA
Descargame Aqui
Inicio
Acerca
Galería
Galería
ESCOM
ESCUELA SUPERIOR DE CÓMPUTO


Nueva pestaña
Buscar en Google o escribir una URL
Gmail
Imágenes
PWA
Inicio
Acerca
Galería
Galería
ESCOM
ESCUELA SUPERIOR DE CÓMPUTO
Acceso directo
Personalizar Chrome

Conclusión

Después de haber trabajado con todos estos archivos y componentes, me quedó mucho más claro qué es una PWA y por qué es tan útil. Básicamente, una Progressive Web App es una aplicación web que se comporta casi como una app nativa. Es decir, se puede abrir desde el navegador como cualquier página, pero también se puede instalar en el escritorio o en la pantalla de inicio del teléfono, funciona sin conexión, y carga mucho más rápido gracias al uso del caché.

Para lograr eso, usé varias piezas clave. Por un lado, el archivo serviceworker.js es el que se encarga de interceptar las peticiones del navegador. Ahí le indico qué archivos quiero que se guarden para que estén disponibles incluso sin internet, y cómo quiero que los maneje. Por ejemplo, le digo que guarde imágenes o la página principal, y que siempre trate de servir la versión más reciente o la que ya tiene guardada.

Luego está serviceworkerregistration.js, que es el archivo que registra ese service worker cuando el usuario abre la app. Me gustó que incluye validaciones para ver si estás trabajando en localhost o ya en producción, y también permite saber si hay una nueva versión del contenido para actualizarla. Además, da la opción de desregistrar el service worker si hace falta.

En cuanto a la estructura visual, usé componentes como Layout, Home, About y Gallery. Layout es el que contiene la navegación y la parte del encabezado, y además ahí coloqué el botón que aparece cuando la app está lista para instalarse. Me pareció muy interesante cómo se puede detectar ese evento (beforeinstallprompt) y luego mostrar el botón para que el usuario decida instalarla, tal como si viniera de una tienda de apps.

Home, About y Gallery son simplemente las diferentes pantallas o rutas de la aplicación. Están hechas con React y muestran contenido como texto e imágenes. Lo bueno es que, al estar dentro de una PWA, ese contenido puede seguir estando disponible aunque el usuario no tenga conexión.

Al final, hacer todo esto me ayudó a comprender cómo una aplicación web sencilla puede evolucionar para ofrecer una experiencia mucho más completa. Aprendí a integrar el service worker, a manejar su registro, a trabajar con rutas y componentes de React, y a mejorar el rendimiento y usabilidad de mi proyecto. Una PWA no solo se ve bien, sino que también funciona bien, incluso en condiciones de poca conectividad. Y eso, para el usuario, hace una gran diferencia.