Instituto Politécnico Nacional

Escuela Superior de Cómputo

Materia: Sistemas Distribuidos

Profesor: Carreto Arellano Chadwick

Grupo: 7CM1

Práctica 8 - PWA

Valdés Castillo Diego - 2021630756

Fecha de Entrega: 07/05/2025

Índice

1. Antecedente (Marco teórico)

En el desarrollo moderno de aplicaciones web, las Progressive Web Apps (PWA) han emergido como una solución híbrida que combina lo mejor de las aplicaciones web y las aplicaciones móviles nativas. Una PWA es una aplicación construida con tecnologías web estándar como HTML, CSS y JavaScript, que gracias al uso de componentes clave como Service Workers y Web App Manifest, puede funcionar sin conexión a internet, enviar notificaciones, y ofrecer una experiencia similar a la de una app instalada.

El Service Worker es un script que el navegador ejecuta en segundo plano, separado de la página web, permitiendo gestionar eventos como el cacheo de recursos, la sincronización en segundo plano, y la interceptación de peticiones de red para mejorar el rendimiento y la experiencia offline. Por su parte, el manifest.json define aspectos esenciales de la aplicación como el nombre, el ícono, los colores de la interfaz y el modo de visualización, permitiendo que la PWA pueda ser instalada en dispositivos móviles o de escritorio como una app nativa.

A diferencia de las aplicaciones web tradicionales, que dependen completamente de una conexión a internet, las PWA buscan ser resilientes, es decir, que sigan funcionando incluso ante fallos de red, mediante estrategias como el cacheo previo

de archivos. Este enfoque proporciona beneficios clave como una carga más rápida, una menor dependencia de la conectividad, y una experiencia de usuario enriquecida.

En el contexto de esta práctica, se desarrolló una aplicación de simulación de carrera de autos donde los usuarios pueden registrar entre cuatro y seis autos, iniciar una carrera, y visualizar el podio final. La aplicación integra las características de una PWA: tiene un Service Worker que permite usarla sin internet una vez cacheada, un manifest.json que define los íconos y apariencia, y se ejecuta en modo standalone, es decir, sin la interfaz del navegador. Esto no solo mejora la experiencia del usuario, sino que también explora conceptos fundamentales de las PWAs como la persistencia de recursos, la instalación local y el rendimiento optimizado.

La arquitectura PWA representa un modelo eficiente y moderno para construir aplicaciones web robustas y accesibles, especialmente en entornos donde la conectividad es limitada o intermitente.

2. Planteamiento del Problema

Esta práctica tiene como objetivo fortalecer el conocimiento en el desarrollo de aplicaciones web modernas mediante la implementación de una Progressive Web App (PWA). En este caso, se propone desarrollar una aplicación que simule una carrera de autos con un mínimo de cuatro y un máximo de seis participantes, incorporando funcionalidades clave de las PWAs como el funcionamiento offline, la instalación en dispositivos y la carga rápida de recursos.

El desafío principal radica en garantizar que la aplicación ofrezca una experiencia fluida, atractiva y funcional, incluso en condiciones de conectividad limitada o inexistente. La PWA debe ser capaz de manejar correctamente los diferentes estados de la aplicación: el registro de autos, el inicio de la carrera, el control del avance de los autos en tiempo real y la presentación del podio final.

Además, es fundamental que el sistema implemente un cacheo eficiente de los recursos críticos como HTML, CSS, JavaScript, imágenes e íconos mediante el uso

de Service Workers. Esto permitirá que, una vez cargada por primera vez, la aplicación pueda seguir siendo utilizada sin necesidad de conexión a internet, garantizando así la resiliencia del sistema y mejorando la experiencia del usuario.

Otro desafío relevante es la validación de las reglas de la carrera: no permitir iniciar la competencia si no hay al menos cuatro autos registrados y controlar que ningún auto avance una vez finalizada la simulación. Estas validaciones son esenciales para asegurar la integridad y coherencia de la lógica de negocio implementada.

Finalmente, la correcta configuración del manifest.json permitirá que la aplicación pueda instalarse en dispositivos móviles o escritorios, funcionando en modo standalone y proporcionando una apariencia profesional similar a la de una app nativa. Esto incluye la definición de íconos, colores de fondo, y el comportamiento inicial al abrirse.

3. Propuesta de Solución

Para abordar el problema planteado, se propone desarrollar una Progressive Web App (PWA) que integre todas las funcionalidades de la simulación de una carrera de autos dentro de una única aplicación web, utilizando tecnologías como HTML, CSS y JavaScript, junto con componentes específicos de PWA como el Service Worker y el manifest.json.

La solución está conformada por los siguientes componentes principales:

- **Módulo de Registro de Autos:**
  Este módulo permite a los usuarios registrar entre cuatro y seis autos antes de iniciar la carrera. Se gestiona mediante un formulario HTML controlado por JavaScript, que valida el número máximo y mínimo de autos requeridos para iniciar la simulación.
- **Módulo de Simulación de Carrera:**
  Administra el avance dinámico de los autos en la pista. Utiliza funciones de JavaScript con temporizadores (setInterval) para generar avances aleatorios en cada auto hasta que lleguen a la meta. Se asegura de que los autos registrados sean los únicos en competir y detiene la simulación al finalizar.

- **Módulo de Visualización del Estado y Podio Final:**
  Permite al usuario observar en tiempo real el progreso de la carrera a través de una representación gráfica en la interfaz web. Una vez finalizada la carrera, se muestra el podio con los tres primeros lugares, resaltado visualmente para mejorar la experiencia de usuario.
- **Service Worker:**
  Encargado de cachear los recursos esenciales de la aplicación (HTML, CSS, JS, imágenes, íconos), lo que permite que la PWA funcione incluso sin conexión a internet. Intercepta las peticiones de red y responde desde la caché cuando no hay conexión disponible, garantizando así la resiliencia de la aplicación.
- **Manifest.json:**
  Define los metadatos principales de la aplicación, como nombre, íconos, colores y comportamiento de inicio. Esto permite que la aplicación pueda instalarse en dispositivos móviles o escritorios, ejecutándose en modo standalone y ofreciendo una apariencia similar a la de una app nativa.
- **Interfaz web interactiva:**
  Los usuarios interactúan a través de una página web visualmente atractiva, donde pueden registrar autos, iniciar la carrera, ver el progreso en la pista y conocer los resultados finales. Los elementos gráficos, como la pista, los autos y el podio, se actualizan dinámicamente para ofrecer una experiencia inmersiva.

A diferencia de una arquitectura distribuida basada en microservicios, servicios web, esquema cliente/servidor, esta solución integra todos los componentes en una sola aplicación PWA, pero mantiene la separación de responsabilidades a nivel lógico dentro del código. Además, aprovecha las ventajas de las PWAs, como el cacheo inteligente, la instalación local y la experiencia offline, para asegurar que los usuarios puedan disfrutar de la aplicación incluso en contextos con conectividad limitada o intermitente.

4. Materiales y Métodos empleados

- **Hardware:** Laptop ASUS y Teléfono iPhone 16.
- **Software:** Google Chrome, Word y Acrobat Reader.
- **Lenguaje de programación:** HTML, CSS, JavaScript y JSON.
- **Entorno de desarrollo:** Se utilizó Visual Studio Code como entorno de desarrollo (IDE), con extensiones para soporte de HTML, CSS y JavaScript. Además, se empleó un servidor local mediante extensiones como Live Server para facilitar la visualización y pruebas de la aplicación en el navegador. No fue necesario utilizar comandos por terminal ni herramientas extras, ya que la interacción del sistema se maneja completamente en el cliente mediante JavaScript.
- **Bibliotecas y tecnologías utilizadas:**
    - **HTML5 y CSS3:** para la estructura y diseño visual de la aplicación, incluyendo la pista de carrera, autos y podio.
    - **JavaScript:** para la lógica de registro, avance de autos, control del flujo de la carrera y actualización de la interfaz en tiempo real.
    - **Service Worker:** para gestionar el cacheo de recursos esenciales (HTML, CSS, JS, imágenes), permitiendo el funcionamiento offline.
    - **Manifest.json:** para definir el nombre, íconos, colores y modo de instalación de la aplicación.
    - **Google Fonts (Orbitron):** para personalizar la tipografía y dar un aspecto visual atractivo a la app.
- **Métodos empleados:**
    - **Registro de autos:**
        - Mediante un formulario HTML, el usuario ingresa el nombre de cada auto.
        - JavaScript valida que haya entre 4 y 6 autos registrados.
        - Una vez registrados, se renderiza dinámicamente la pista con cada auto en su carril.
    - **Simulación de carrera:**

        - Se utiliza setInterval en JavaScript para avanzar los autos en intervalos regulares con valores aleatorios.
        - La lógica asegura que solo los autos registrados participen y detiene la simulación cuando todos alcanzan la meta.
        - Las posiciones de los autos se actualizan dinámicamente en la interfaz.
    - **Visualización del podio:**
        - Al finalizar la carrera, se muestra un podio con los tres primeros lugares.
        - Se oculta la pista y se despliega la sección de resultados finales.
    - **Service Worker:**
        - Se implementó para interceptar y cachear archivos estáticos como HTML, CSS, JS, imágenes e íconos.
        - Permite que la aplicación funcione offline después de la primera carga.
    - **Manifest.json:**
        - Define los metadatos que permiten que la aplicación pueda instalarse en el dispositivo del usuario como si fuera una app nativa.
        - Incluye configuraciones como íconos, nombre corto, esquema de colores y modo de inicio.
- **Estrategia de programación:**
    - El registro, la simulación y la visualización de resultados están claramente diferenciados en el código.
    - El Service Worker garantiza la resiliencia ante fallos de red, mejorando la experiencia del usuario.
    - El uso de recursos locales cacheados asegura tiempos de carga rápidos y reduce la dependencia de la conectividad.
    - La programación con setInterval y el manejo del DOM permiten que la carrera se actualice en tiempo real, simulando un sistema dinámico.

5. Desarrollo de la Solución

Para implementar la solución de la simulación de una carrera de autos como Progressive Web App (PWA), se desarrolló una aplicación web completa que integra todas las funcionalidades dentro de un único proyecto, utilizando HTML, CSS y JavaScript, junto con tecnologías específicas de PWA como Service Workers y manifest.json.

El proceso inicia cuando un usuario accede a la interfaz web desde un navegador. La interfaz permite registrar autos, iniciar la carrera, visualizar el progreso en tiempo real y mostrar el podio final al concluir la competencia. La aplicación está diseñada para permitir entre 4 y 6 autos participantes, garantizando que no se pueda iniciar la carrera hasta que se cumpla el mínimo requerido.

La estructura general del sistema incluye los siguientes módulos:

- **Registro de Autos:**
  El usuario introduce el nombre de cada auto mediante un formulario HTML. El sistema valida que no se supere el límite máximo de seis autos y que se cumpla el mínimo de cuatro para habilitar el inicio de la carrera. Una vez registrados, los autos son renderizados dinámicamente en la pista de carrera.
- **Simulación de Carrera:**
  Al presionar el botón de iniciar carrera, la lógica en JavaScript utiliza setInterval para generar avances aleatorios en cada auto. Las posiciones se actualizan en tiempo real en la interfaz, simulando una competencia dinámica y entretenida. El sistema se asegura de que los autos no sigan avanzando una vez alcanzada la meta.
- **Visualización del Podio:**
  Al finalizar la carrera, el sistema ordena a los autos según su llegada y muestra un podio final. Esta información se presenta de forma visual en la interfaz, destacando a los tres primeros lugares con medallas de oro, plata y bronce.
- **Service Worker:**

El Service Worker intercepta las solicitudes de red y almacena en caché los recursos críticos de la aplicación, como HTML, CSS, JavaScript, imágenes e íconos. Gracias a esto, la aplicación sigue siendo funcional aunque el usuario pierda la conexión a internet después de la primera carga.

- **Manifest.json:**
  Define los metadatos de la aplicación, incluyendo el nombre, los íconos, el color de fondo y el comportamiento de visualización, permitiendo que la aplicación pueda instalarse en dispositivos móviles o escritorios como si fuera una app nativa.

Durante la ejecución de la aplicación, no es necesario realizar consultas al servidor mediante HTTP (GET o POST), ya que todo el procesamiento ocurre del lado del cliente en el navegador. La lógica de avance, la validación de reglas, la actualización de posiciones y la presentación de resultados son manejadas internamente con JavaScript, garantizando una experiencia fluida y responsiva.

Gracias al uso de las tecnologías PWA, la aplicación ofrece tiempos de carga rápidos, funciona offline, y puede instalarse en dispositivos, proporcionando una experiencia moderna, accesible y eficiente.

A continuación se puede ver la solución propuesta en código documentado en la figura 1, 2, 3, 4, 5, 6, 7, 8, 9 y 10:

Figura 1. Código en JSON – manifest.json

index.html > html > body > h1.titulo

D:\Escuela8voSem\Distribuidos\Practica8\index.html

```html
<html lang="es">
<head>
  <meta charset="UTF-8" />
  <link rel="icon" href="icons/icon-192.png" type="image/png">
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <link rel="manifest" href="manifest.json" />
  <link rel="stylesheet" href="style.css" />
  <title>Carrera de Autos PWA</title>
</head>
<body>
  <h1 class="titulo">🏁 Carrera de Autos</h1>
  <section id="registro">
    <input type="text" id="nombreAuto" placeholder="Nombre del auto" />
    <button onclick="registrarAuto()">Registrar Auto</button>
    <p id="mensaje"></p>
  </section>

  <section id="pista" class="hide">
    <div id="autos"></div>
    <button id="iniciarBtn" onclick="iniciarCarrera()">Iniciar Carrera</button>
  </section>

  <section id="podio" class="hide">
    <h2>Resultado Final 🏆</h2>
    <ol id="resultado"></ol>
    <button onclick="reiniciarCarrera()">Reiniciar</button>
  </section>

  <script src="app.js"></script>
  <script>
    if ('serviceWorker' in navigator) {
      navigator.serviceWorker.register('service-worker.js');
    }
  </script>
</body>
</html>
```

Figura 2. Código en HTML – index.html

service-worker.js > ...

```javascript
const CACHE_NAME = 'carrera-autos-cache-v3';
const urlsToCache = [
  '/',
  'https://fonts.googleapis.com/css2?family=Orbitron:wght@600&display=swap',
  '/index.html',
  '/style.css',
  '/app.js',
  '/manifest.json',
  '/service-worker.js',
  '/icons/icon-192.png',
  '/icons/icon-512.png',
  '/img/auto1.png',
  '/img/auto2.png',
  '/img/auto3.png',
  '/img/auto4.png',
  '/img/auto5.png',
  '/img/auto6.png'
];

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

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

Figura 3. Código en JS – service-worker.js

```
JS app.js > ...
let autos = [];
let posiciones = [];
let intervalo = null;
function registrarAuto() {
  const input = document.getElementById('nombreAuto');
  const nombre = input.value.trim();
  const mensaje = document.getElementById('mensaje');

  if (!nombre) {
    mensaje.innerText = 'Ponle nombre al auto ';
    return;
  }

  if (autos.length >= 6) {
    mensaje.innerText = 'Máximo 6 autos permitidos';
    return;
  }

  autos.push({ nombre, id: autos.length });
  mensaje.innerText = '';

  input.value = '';
  renderizarAutos();
}
function renderizarAutos() {
  const pista = document.getElementById('pista');
  const contenedor = document.getElementById('autos');
  contenedor.innerHTML = '';

  autos.forEach((auto, index) => {
    const carril = document.createElement('div');
    carril.className = 'carril';

    const nombre = document.createElement('div');
    nombre.className = 'nombre-auto';
    nombre.innerText = auto.nombre;
```

Figura 4. Código en JS – app.js – 1

```
    const imagen = document.createElement('div');
    imagen.className = 'carro';
    imagen.id = `auto-${index}`;
    imagen.innerHTML = `<img src="img/auto${(index % 6) + 1}.png" alt="auto" />`;

    const meta = document.createElement('div');
    meta.className = 'meta';

    carril.appendChild(nombre);
    carril.appendChild(imagen);
    carril.appendChild(meta);
    contenedor.appendChild(carril);
  });

  pista.classList.remove('hide');
}
function iniciarCarrera() {
  if (autos.length < 4) {
    alert('Se necesitan al menos 4 autos para iniciar la carrera.');
    return;
  }

  posiciones = [];
  const meta = document.querySelector('.carril').offsetWidth - 60;

  intervalo = setInterval(() => {
    autos.forEach((auto, i) => {
      if (posiciones.includes(auto)) return;

      const img = document.getElementById(`auto-${i}`);
      let pos = parseInt(img.style.left) || 0;
      let avance = Math.floor(Math.random() * 10) + 5;
      pos += avance;

```

Figura 5. Código en JS – app.js – 2

```
      if (pos >= meta) {
        pos = meta;
        posiciones.push(auto);
        if (posiciones.length === autos.length) {
          clearInterval(intervalo);
          mostrarPodio();
        }
      }

      img.style.left = `${pos}px`;
    });
  }, 100);
}
function mostrarPodio() {
  document.getElementById('pista').classList.add('hide');
  const podio = document.getElementById('podio');
  const lista = document.getElementById('resultado');
  lista.innerHTML = '';
  posiciones.forEach(auto => {
    const li = document.createElement('li');
    li.innerText = auto.nombre;
    lista.appendChild(li);
  });
  podio.classList.remove('hide');
}
function reiniciarCarrera() {
  autos = [];
  posiciones = [];
  document.getElementById('autos').innerHTML = '';
  document.getElementById('registro').classList.remove('hide');
  document.getElementById('pista').classList.add('hide');
  document.getElementById('podio').classList.add('hide');
}
```

Figura 6. Código en JS – app.js – 3

style.css > .titulo

```
@import url('https://fonts.googleapis.com/css2?family=Orbitron:wght@600&display=swap');

body {
  font-family: 'Orbitron', sans-serif;
  margin: 0;
  padding: 0;
  background: linear-gradient(to right, #141e30, #243b55);
  color: #fff;
  text-align: center;
}

.titulo {
  margin: 1.5rem 0;
  font-size: 2.8rem;
  color: #fcd307;
  text-shadow: 2px 2px 5px black;
}

#registro,
#pista,
#podio {
  margin: 1.5rem auto;
  max-width: 800px;
  background: rgba(0, 0, 0, 0.7);
  padding: 1.5rem;
  border-radius: 15px;
  box-shadow: 0 0 12px rgba(0, 0, 0, 0.5);
}

input[type="text"] {
  padding: 0.7rem;
  width: 60%;
  border-radius: 6px;
  border: none;
  background: #f8f9fa;
  font-size: 1rem;
  margin-bottom: 10px;
```

Figura 7. Código en CSS – style.css – 1

```css
}

button {
  margin-top: 1rem;
  padding: 0.7rem 1.5rem;
  border: none;
  border-radius: 8px;
  background: #28a745;
  color: white;
  font-size: 1rem;
  font-weight: bold;
  cursor: pointer;
  transition: background 0.3s;
}

button:hover {
  background: #218838;
}

.carril {
  position: relative;
  height: 70px;
  margin: 14px 0;
  background: #495057;
  border-radius: 10px;
  overflow: hidden;
  border: 2px dashed #fcd307;
}

.nombre-auto {
  position: absolute;
  left: 10px;
  top: 8px;
  font-weight: bold;
  background: rgba(255, 255, 255, 0.9);
  color: #000;
  padding: 4px 10px;
```

Figura 8. Código en CSS – style.css – 2

```css
  border-radius: 6px;
  z-index: 2;
  font-size: 0.9rem;
}

.carro {
  position: absolute;
  top: 10px;
  left: 0;
  transition: left 0.3s linear;
  z-index: 1;
}

.carro img {
  height: 50px;
  width: auto;
  filter: drop-shadow(2px 2px 3px #000);
}

.meta {
  position: absolute;
  right: 0;
  top: 0;
  width: 12px;
  height: 100%;
  background: repeating-linear-gradient(45deg, red, red 5px, white 5px, white 10px);
  z-index: 0;
}

#resultado {
  list-style-type: none;
  padding: 0;
}

#resultado li {
  background: #343a40;
```

Figura 9. Código en CSS – style.css – 3

```
111   padding: 0.7rem;
112   border-radius: 10px;
113   margin: 0.6rem 0;
114   font-weight: bold;
115   font-size: 1.1rem;
116   color: #fff;
117   box-shadow: 0 0 8px #000;
118 }
119
120 #resultado li:nth-child(1)::before {
121   content: "🥇 ";
122   color: gold;
123 }
124
125 #resultado li:nth-child(2)::before {
126   content: "🥈 ";
127   color: silver;
128 }
129
130 #resultado li:nth-child(3)::before {
131   content: "🥉 ";
132   color: #cd7f32;
133 }
134
135 .hide {
136   display: none;
137 }
138
```

Figura 10. Código en CSS – style.css – 4

6. Resultados

Los resultados obtenidos evidencian que la implementación de la simulación de carrera de autos como Progressive Web App (PWA) permite ofrecer una experiencia fluida, interactiva y resiliente, incluso en escenarios con conectividad limitada o nula. La aplicación respondió correctamente durante las pruebas, mostrando que la lógica de registro, simulación y presentación de resultados funciona de manera consistente y sin errores.

Durante las ejecuciones de prueba, se verificó que el registro de autos se realiza correctamente, validando el número mínimo y máximo permitido, y generando mensajes de advertencia adecuados cuando se intenta iniciar la carrera con menos de cuatro autos. Una vez iniciada la simulación, el avance aleatorio de los autos produjo resultados variados en cada ejecución, logrando una experiencia dinámica y realista para el usuario.

El uso de JavaScript para el control de movimiento y actualización de posiciones permitió que la visualización en la interfaz web se mantuviera sincronizada y fluida en todo momento. La representación gráfica de la pista, los autos y la meta se actualizó dinámicamente, reflejando el progreso de la carrera en tiempo real.

Al finalizar la carrera, el sistema generó automáticamente el podio final, ordenando a los autos según su llegada y mostrando los resultados de forma clara y atractiva en la interfaz. Los usuarios pudieron reiniciar la carrera sin necesidad de recargar la página, demostrando la solidez de la lógica de reinicio implementada.

Finalmente, se validó el correcto funcionamiento del Service Worker, que permitió utilizar la aplicación offline después de la primera carga, garantizando acceso a todas las funcionalidades sin conexión a internet. Esto asegura una experiencia de usuario continua y confiable, cumpliendo con los objetivos fundamentales de una PWA.

A continuación se puede ver una prueba de ejecución de la PWA desde Visual Studio Code con la extensión Live Server, primero en la figura 11 podemos ver el botón “Go live” con el cual habilitamos el Live Server para poder ver nuestra aplicación.

Figura 11. Botón para el despliegue de la PWA con Live Server.

Al oprimir este botón automáticamente nos despliega la aplicación en nuestro navegador web predeterminado, en mi caso la PWA la muestra en Chrome como se puede ver en la figura 12 a continuación:

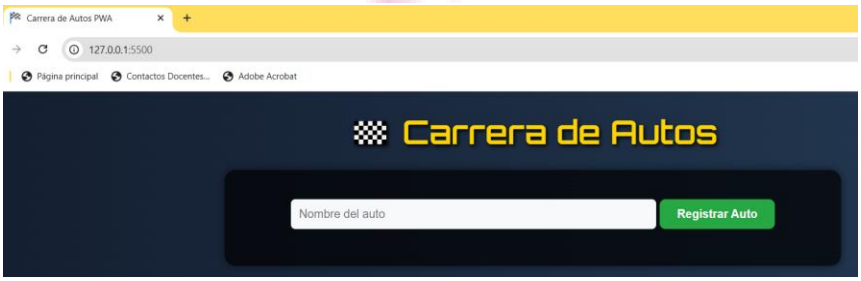

Figura 12. PWA en el navegador web Chrome.

Antes de ver el funcionamiento de la carrera, vamos a comprobar que realmente el esquema corresponda a una PWA, oprimimos “F12” para poder observar y verificar el service worker y el manifest como se puede ver en la figura 13 a continuación:

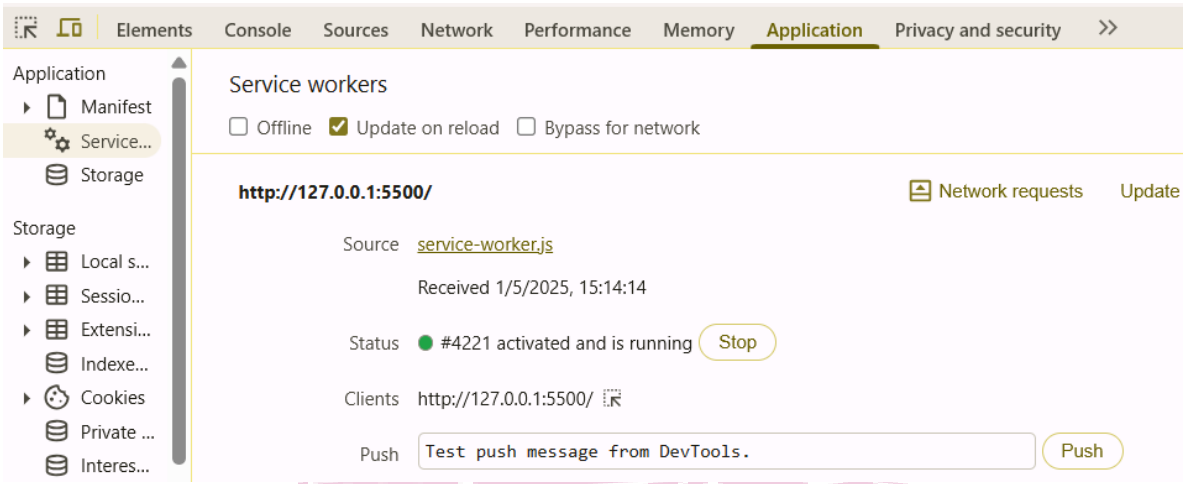

Figura 13. Application para verificar service worker y manifest.

Desde la opción Application nos vamos a manifest y observamos que todo está cargado correctamente para poder instalar nuestra PWA en escritorio o para dispositivo móvil como se puede ver en la figura 14 a continuación:

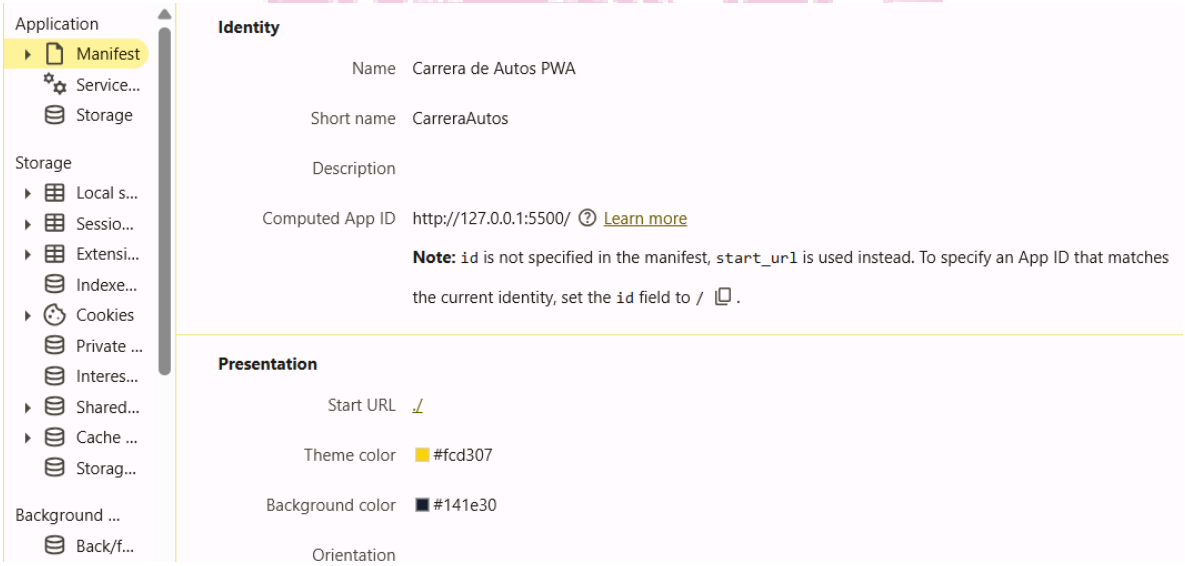

Figura 14. Manifest cargado correctamente para instalación de PWA.

Ahora vamos a verificar la parte del service worker, igual nos vamos a la opción de Application y en la parte de Service workers, si esta activado y corriendo el service worker todo funcionará correctamente para nuestra PWA, pero si no vemos esa opción habilitada, entonces no funcionará como corresponde nuestra PWA. En nuestro caso todo salió perfecto y el service worker esta ejecutando correctamente como se puede ver en la figura 15 a continuación:

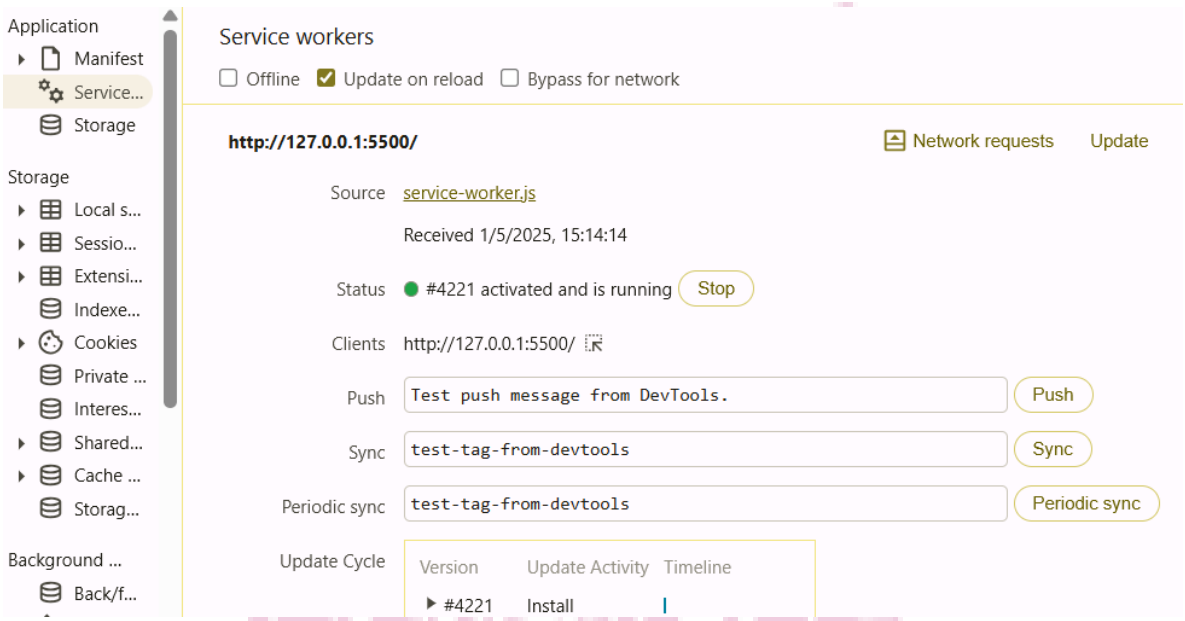

Figura 15. Service worker cargado correctamente para instalación de PWA.

Ahora si vamos con la prueba del funcionamiento, para comprobar el esquema de una PWA haremos dos pruebas, la primera descargando la PWA para escritorio y ejecutando la simulación de la carrera y la otra usando el modo offline desde el navegador web y ejecutando igual la simulación de la carrera.

Primero vamos a descargar la PWA oprimiendo el botón que se ve en la figura 16 a continuación:

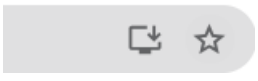

Figura 16. Botón para descargar la PWA.

Una vez descargada la PWA vamos a registrar estos 4 autos: Mazda, Aveo, Tsuru y Ferrari como se puede ver en la figura 17 a continuación:

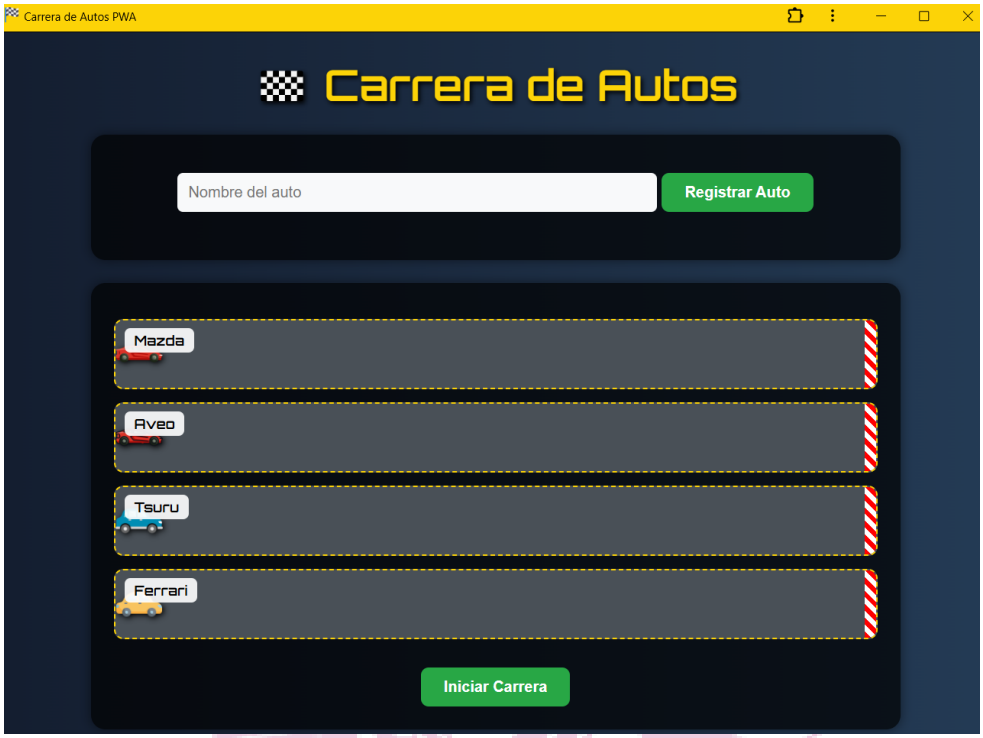

Figura 17. Registro de autos para carrera.

Ahora damos click en Iniciar Carrera y vemos que los autos avanzan de forma aleatoria hasta llegar a la meta como se puede ver en la figura 18 a continuación:

Figura 18. Simulación de la carrera.

Al final llegan los 4 autos a la meta y la PWA muestra el podio final como se puede ver en la figura 19 a continuación:

Figura 19. Podio Final de la carrera desde la PWA.

Además podemos ver en el escritorio que nuestra aplicación se instalo correctamente como se puede ver en la figura 20 a continuación:

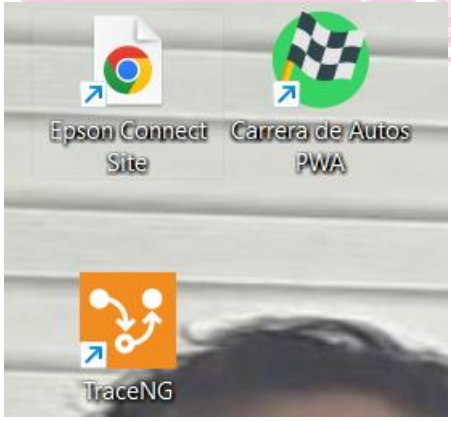

Figura 20. PWA instala en el escritorio correctamente.

Ahora activamos el modo offline desde la opción de Service workers y vamos a registrar estos 4 autos: Mazda, Aveo, Tsuru y Ferrari como se puede ver en la figura 21 a continuación:

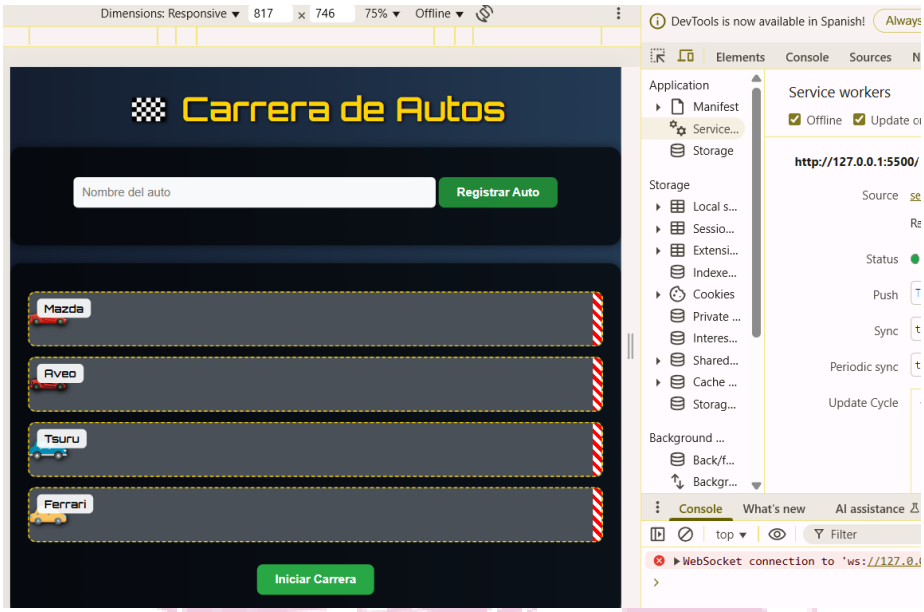

Figura 21. Registro de autos para carrera desde offline.

Ahora damos click en Iniciar Carrera y vemos que los autos avanzan de forma aleatoria hasta llegar a la meta como se puede ver en la figura 22 a continuación:

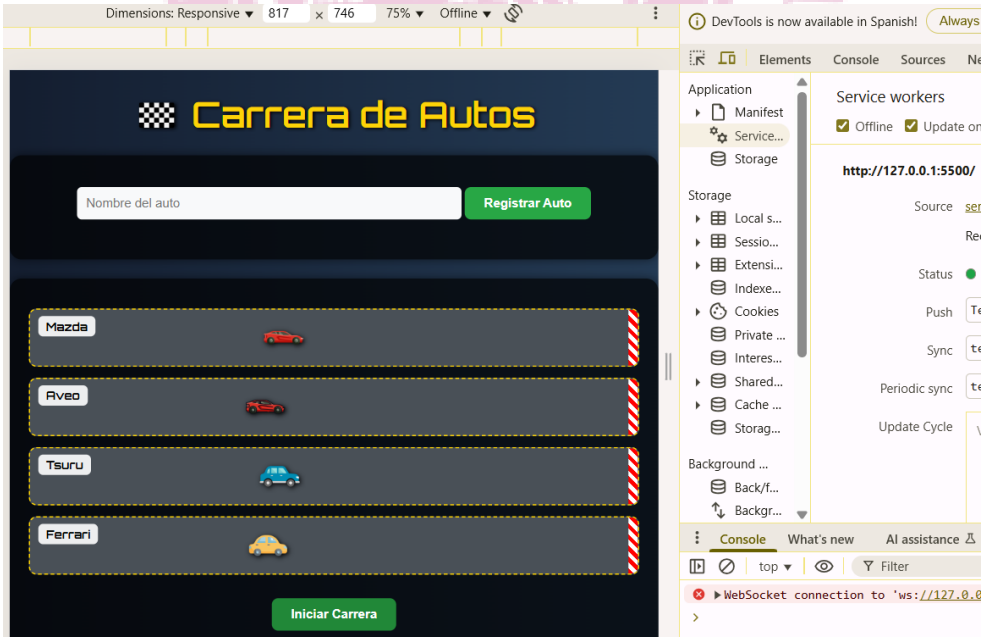

Figura 22. Simulación de la carrera desde offline.

Al final llegan los 4 autos a la meta y la PWA muestra el podio final como se puede ver en la figura 23 a continuación:

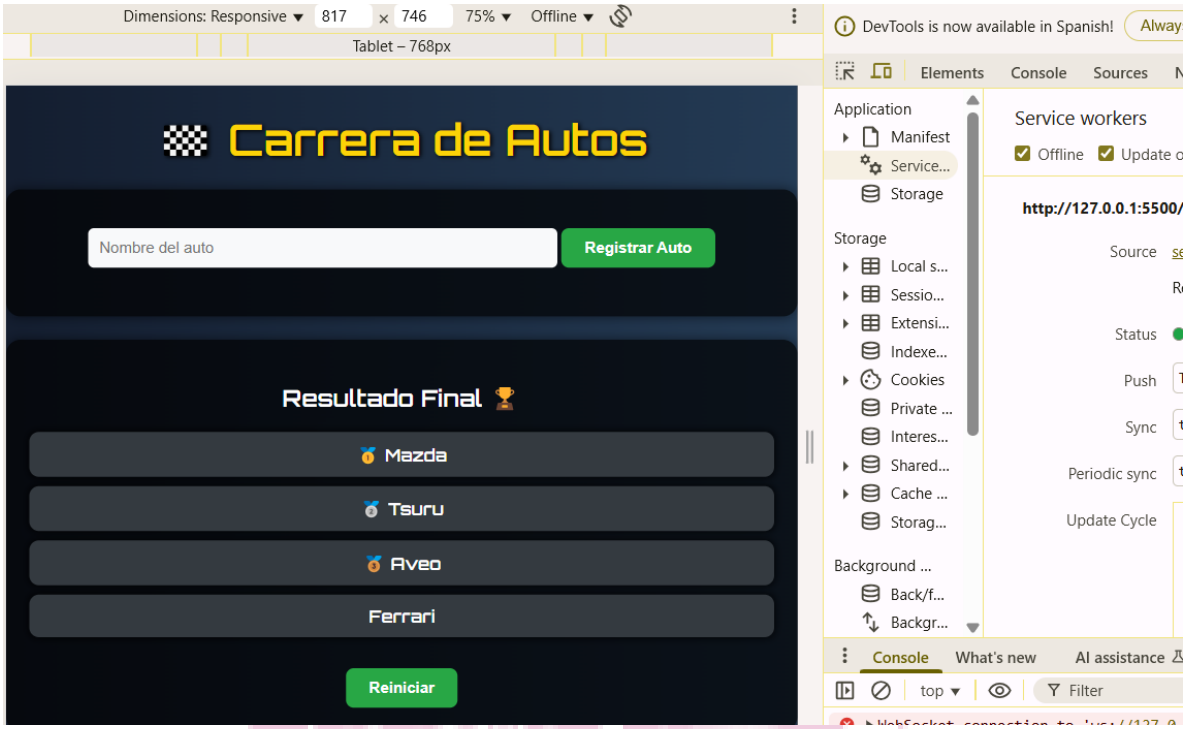

Figura 23. Podio Final de la carrera desde el modo offline.

Además podemos ver que la carrera si es totalmente aleatoria, ya que desde la PWA instalada gano Mazda, luego quedo Aveo, después Ferrari y al final el Tsuru, a diferencia desde el modo offline que volvió a ganar el Mazda, pero segundo quedo Tsuru, tercero Aveo y al último lugar Ferrari.

7. Conclusiones

La implementación de la simulación de carrera de autos como Progressive Web App (PWA) ha demostrado ser altamente efectiva para ofrecer una experiencia dinámica, interactiva y resiliente, incluso en contextos con conectividad limitada. Al integrar componentes clave como Service Workers y manifest.json, se logró construir una aplicación modular, escalable y mantenible, capaz de funcionar offline y de instalarse en dispositivos móviles o de escritorio como si fuera una app nativa.

La separación lógica de responsabilidades dentro del código permitió gestionar de manera organizada las distintas fases de la aplicación: registro de autos, simulación de la carrera, actualización del estado en tiempo real y presentación del podio final. La lógica implementada en JavaScript garantizó un flujo coherente y controlado de cada etapa, evitando errores como iniciar la carrera sin el mínimo de autos registrados o permitir avances una vez terminada la competencia.

El uso de tecnologías PWA permitió almacenar en caché los recursos críticos y garantizar tiempos de carga rápidos, mejorando significativamente la experiencia del usuario. Además, la aplicación cumplió con su objetivo de ofrecer una interacción atractiva y entretenida, mostrando resultados variados en cada carrera gracias a la lógica de avance aleatorio, lo que aporta realismo e imprevisibilidad al juego.

Los resultados obtenidos confirman que las PWAs son una alternativa sólida para el desarrollo de aplicaciones web modernas, al combinar flexibilidad, rendimiento optimizado y resiliencia frente a fallos de red.

8. Referencias Bibliográficas

- Mozilla. (s. f.). Aplicaciones web progresivas (PWA). MDN Web Docs. Recuperado de https://developer.mozilla.org/es/docs/Web/Progressive_web_apps
- YouTube. (2022). ¿Cómo crear una aplicación web progresiva (PWA) desde cero como un experto? - Guía paso a paso [Video]. YouTube. Recuperado de https://www.youtube.com/watch?v=72F0hcSDXGc
- Maestros Web. (s. f.). Desarrollo PWA desde cero: Guía completa para principiantes. Punta Network. Recuperado de https://maestrosweb.puntanetwork.com/desarrollo-responsive-y-movil/desarrollo-pwa-cero-guia-completa-principiantes/