



Instituto Politécnico Nacional

Escuela Superior de Cómputo



## DESARROLLO DE SISTEMAS DISTRIBUIDOS

### Práctica 8

**Profesor:** Chadwick Carreto Arellano

**Grupo:** 4CM11

**Alumnos:**

González Hipólito Miguel Ángel

Ponce López José Angel

## Contents

1	Antecedentes .....	3
2	Planteamiento del problema.....	3
3	Materiales y Métodos empleados.....	3
3.1	Materiales Utilizados.....	3
4	Desarrollo .....	4
4.1	🔧 Configuración técnica (PWA con Vite).....	4
4.2	Tablero.....	6
4.3	Cuadrados .....	8
4.4	Turno .....	10
5	Instalación .....	10
6	Conclusiones .....	12

## **1 Antecedentes**

Las aplicaciones web progresivas (PWA) son aplicaciones que se compilan mediante tecnologías web y que se pueden instalar y ejecutar en todos los dispositivos, desde un código base.

Las PWA proporcionan experiencias nativas a los usuarios en dispositivos auxiliares. Se adaptan a las funcionalidades admitidas por cada dispositivo y también se pueden ejecutar en exploradores web, como sitios web.

Las ventajas de las pwa:

- Las PWA tienen sus propios iconos de aplicación que se pueden agregar a la pantalla principal o barra de tareas de un dispositivo.
- Las PWA se pueden iniciar automáticamente cuando se abre un tipo de archivo asociado.
- Las PWA se pueden ejecutar cuando el usuario inicia sesión.
- Las PWA se pueden enviar a almacenes de aplicaciones, como Microsoft Store.

## **2 Planteamiento del problema**

En el contexto actual, donde los usuarios exigen aplicaciones rápidas, accesibles y funcionales incluso sin conexión a internet, surge la necesidad de transformar aplicaciones web tradicionales en soluciones más robustas y modernas. Muchos juegos simples como el Tic Tac Toe (gato) están limitados a ejecutarse únicamente en línea o requieren de una instalación nativa desde tiendas de aplicaciones, lo cual dificulta su disponibilidad en entornos con conectividad limitada.

Problema:

No contar con una versión de Tic Tac Toe que sea instalable, funcione sin conexión y brinde una experiencia de usuario similar a una aplicación móvil o de escritorio, representa una limitación en cuanto a accesibilidad, portabilidad y experiencia de uso.

## **3 Materiales y Métodos empleados**

### **3.1 Materiales Utilizados**

- **Lenguaje de programación:** Javascript

- **Herramienta:** vite
- **Entorno de desarrollo:** VS Code
- **Sistema operativo:** Windows

## 4 Desarrollo

Como solución, se desarrolló una **Progressive Web App (PWA)** que implementa el clásico juego de Tic Tac Toe utilizando **React** y **Vite**. Este enfoque permite que la aplicación sea:

- Instalable en el dispositivo del usuario.
- Ejecutable sin conexión a internet.
- Visualmente adaptable a diferentes dispositivos (responsive).
- Rápida en carga y fluida en su interacción.

### 4.1 Configuración técnica (PWA con Vite)

Se utilizó el plugin vite-plugin-pwa para convertir la aplicación en una PWA. Dentro de la configuración se implementó el manifest.json de forma declarativa:

```
1  export default defineConfig({
2    plugins: [
3      react(),
4      VitePWA(
5        {
6          registerType:'autoUpdate',
7          includeAssets:['favicon.svg','robots.txt'],
8          manifest:{
9            name:"Practica 8",
10           short_name:"P8",
11           start_url '/',
12           display:"standalone",
13           background_color:'#ffffff',
14           theme_color:'#ffffff',
15           lang:"es",
16           screenshots:[
17             {
18               src:'/screenshots/screen-728-410.png',
19               sizes:"728x410",
20               type:"image/png",
21               form_factor:"wide"
22             },
23             {
24               src:'/screenshots/screen-736-1309.png',
25               sizes:"736x1309",
26               type:"image/png",
27               form_factor:'narrow'
28             }
29           ],
30           icons:[
31             {
32               src:'/icons/icon-192.png',
33               sizes:"192x192",
34               type:"image/png"
35             },
36             {
37               src:'/icons/icon-512.png',
38               sizes:"512x512",
39               type:"image/png"
40             }
41           ]
42         },
43         workbox:{
44           globPatterns: ["**/*.{js,css,html,ico,png,svg,webmanifest}"],
45         },
46       }
47     )
48   ],
49 })
50 }
```

## 4.2 Tablero

Este componente representa el **tablero de juego** completo. Recibe como propiedades el estado actual de las casillas (squares) y una función para manejar los clics (onGame).

### Funciones clave:

- validateWinner: Recorre las combinaciones posibles para determinar si hay un ganador.
- handleClick: Llama a onGame solo si la celda está vacía y aún no hay ganador.
- Dibuja las 9 celdas del tablero utilizando el componente Square.

### Mejoras incluidas:

- Detección de la línea ganadora.
- Pasar celdas ganadoras a Square para resaltarlas visualmente.

```
 1  interface BoardProps {
 2    squares: Array<string>;
 3    onGame: (actualIndex: number) => void;
 4  }
 5
 6  const Board = ({ squares, onGame }: BoardProps) => {
 7    const winnerInfo = getWinnerInfo();
 8
 9    const handleClick = (index: number) => {
10      if (!winnerInfo && !squares[index]) {
11        onGame(index);
12      }
13    };
14
15    function getWinnerInfo(): { winner: string, line: number[] } | null {
16      const lines = [
17        [0, 1, 2],
18        [3, 4, 5],
19        [6, 7, 8],
20        [0, 3, 6],
21        [1, 4, 7],
22        [2, 5, 8],
23        [0, 4, 8],
24        [2, 4, 6],
25      ];
26      for (let line of lines) {
27        const [a, b, c] = line;
28        if (squares[a] && squares[a] === squares[b] && squares[a] === squares[c]) {
29          return { winner: squares[a], line };
30        }
31      }
32      return null;
33    }
34
35    const winner = winnerInfo?.winner;
36    const winnerCells = winnerInfo?.line ?? [];
37
38    return (
39      <>
40        {winner && <h2>🎉 ¡Ganó {winner.toUpperCase()}!</h2>}
41        <div className="board">
42          {squares.map((value, i) => (
43            <Square
44              key={i}
45              value={value}
46              handleClickSquare={() => handleClick(i)}
47              highlight={winnerCells.includes(i)}
48              disabled={!winner}
49            />
50          )))
51        </div>
52      </>
53    );
54  };
55
56  export default Board;
57
```

#### 4.3 Cuadrados

Representa una **celda individual del tablero**. Recibe el valor actual (X, O o vacío), si debe estar deshabilitada y si debe resaltarse por haber sido parte de una jugada ganadora.

##### Props utilizadas:

- value: el contenido de la celda.
- handleClickSquare: función que se ejecuta al hacer clic.
- disabled: evita que una celda ya jugada o bloqueada por finalización se vuelva a seleccionar.
- highlight: permite aplicar estilos especiales si la celda forma parte de la jugada ganadora.

```
 1 import Square from "./square"
 2
 3 interface BoardProps {
 4   squares: Array<string>;
 5   onGame: (actualIndex: number) => void;
 6 }
 7
 8 const Board = ({ squares, onGame }: BoardProps) => {
 9   const winnerInfo = getWinnerInfo();
10
11   const handleClick = (index: number) => {
12     if (!winnerInfo && !squares[index]) {
13       onGame(index);
14     }
15   };
16
17   function getWinnerInfo(): { winner: string, line: number[] } | null {
18     const lines = [
19       [0, 1, 2],
20       [3, 4, 5],
21       [6, 7, 8],
22       [0, 3, 6],
23       [1, 4, 7],
24       [2, 5, 8],
25       [0, 4, 8],
26       [2, 4, 6],
27     ];
28     for (let line of lines) {
29       const [a, b, c] = line;
30       if (squares[a] && squares[a] === squares[b] && squares[a] === squares[c]) {
31         return { winner: squares[a], line };
32       }
33     }
34     return null;
35   }
36
37   const winner = winnerInfo?.winner;
38   const winnerCells = winnerInfo?.line ?? [];
39
40   return (
41     <>
42       {winner && <h2>¡Ganó {winner.toUpperCase()}!</h2>}
43       <div className="board">
44         {squares.map((value, i) => (
45           <Square
46             key={i}
47             value={value}
48             handleClickSquare={() => handleClick(i)}
49             highlight={winnerCells.includes(i)}
50             disabled={!winner}
51           />
52         )));
53       </div>
54     </>
55   );
56 }
57
58 export default Board;
59
```

#### 4.4 Turno

Muestra en pantalla el **turno del jugador actual**. Aunque simple, mejora la experiencia del usuario manteniéndolo informado.

##### Propiedad:

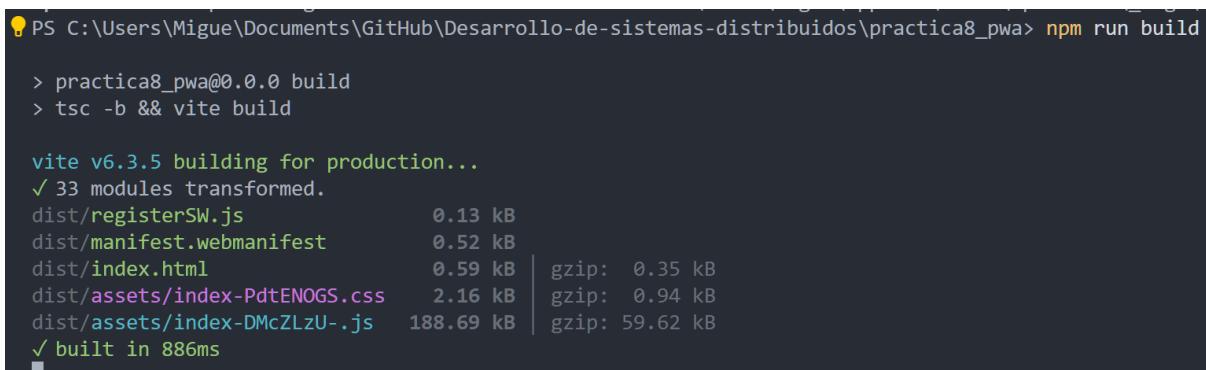
- user: el jugador actual (X u O).



```
1
2
3 interface ScoreBoardProps{
4     user:string
5 }
6
7 const ScoreBoard = ({ user }: ScoreBoardProps) => {
8     return (
9         <div style={{ fontSize: '1.2rem', fontWeight: 'bold' }}>
10             Turno de: <span style={{ color: 'oklch(50% 0.2 256)' }}>{user.toUpperCase()}</span>
11         </div>
12     );
13 };
14
15 export default ScoreBoard
```

#### 5 Instalación

Primero, se compila la aplicación



```
PS C:\Users\Migue\Documents\GitHub\Desarrollo-de-sistemas-distribuidos\practica8_pwa> npm run build

> practica8_pwa@0.0.0 build
> tsc -b && vite build

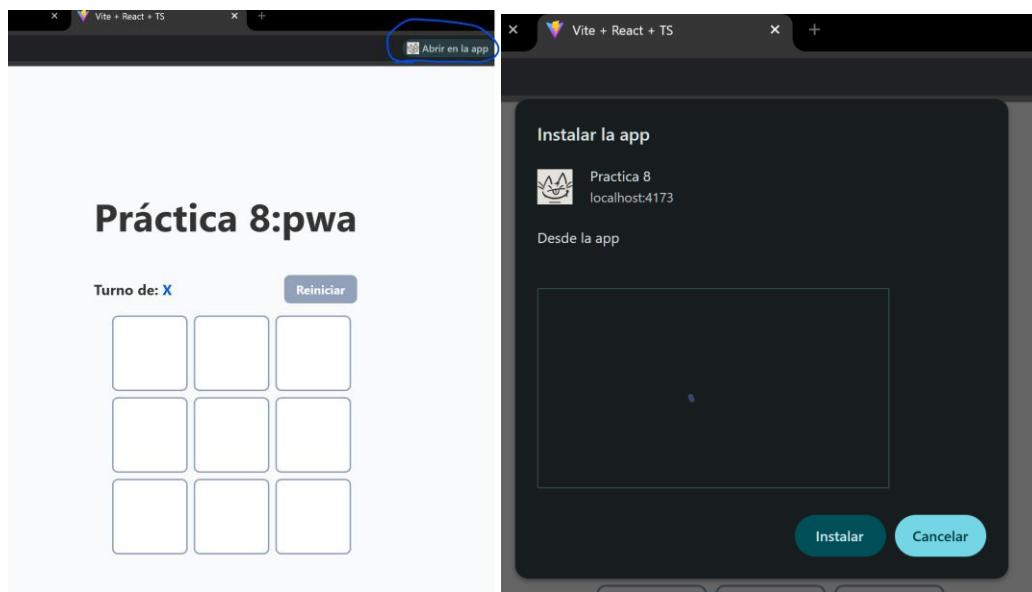
vite v6.3.5 building for production...
✓ 33 modules transformed.
dist/registerSW.js          0.13 kB
dist/manifest.webmanifest      0.52 kB
dist/index.html              0.59 kB  gzip:  0.35 kB
dist/assets/index-PdtENOOGS.css 2.16 kB  gzip:  0.94 kB
dist/assets/index-DMcZLzU-.js   188.69 kB  gzip: 59.62 kB
✓ built in 886ms
```

Posteriormente, se realiza un preview y se accede a la dirección ip

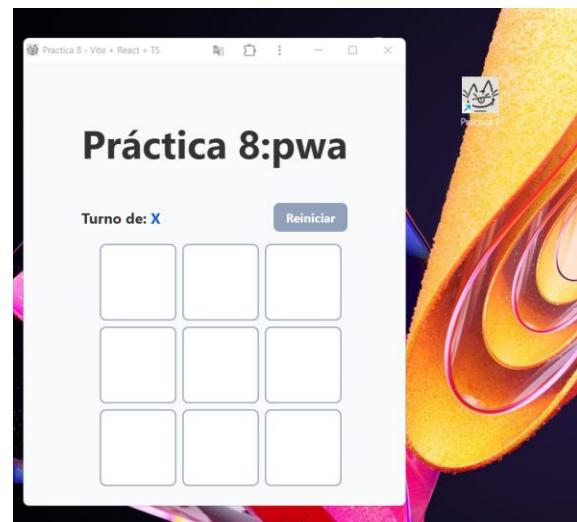
```
PS C:\Users\Migue\Documents\GitHub\Desarrollo-de-sistemas-distribuidos\practica8_pwa> npm run preview
> practica8_pwa@0.0.0 preview
> vite preview

→ Local: http://localhost:4173/
→ Network: use --host to expose
→ press h + enter to show help
```

Dependiendo el dispositivo, podremos instalar la app como si fuera nativa.



Por último, podremos ver el ícono de la aplicación en la pantalla de inicio.



## 6 Conclusiones

La implementación del juego **Tic Tac Toe como una Progressive Web App** demostró ser una solución eficaz para mejorar la experiencia del usuario, al brindar funcionalidades más allá de una aplicación web convencional. A lo largo del desarrollo se cumplieron los objetivos de instalabilidad, funcionamiento offline y experiencia fluida, integrando tecnologías modernas como React, Vite y vite-plugin-pwa.

Entre los principales beneficios alcanzados se encuentran:

- **Accesibilidad mejorada:** los usuarios pueden instalar la aplicación directamente desde su navegador sin necesidad de tiendas de apps.
- **Soporte offline:** mediante el uso de *Service Workers*, el juego permanece funcional incluso sin conexión a internet.
- **Aspecto profesional:** gracias al manifest, la aplicación adopta una apariencia nativa, con íconos personalizados, nombre corto y modo standalone.
- **Desempeño eficiente:** la integración con Vite permitió tiempos de carga rápidos y una experiencia optimizada para dispositivos móviles y de escritorio.

En resumen, convertir este proyecto en una PWA no solo aportó mejoras técnicas, sino también potenció el valor del producto al hacerlo más accesible, confiable y cercano a una aplicación nativa. Esta práctica es aplicable a una amplia variedad de desarrollos web que buscan mejorar su alcance y usabilidad sin depender de plataformas externas.