

Name:Komal Singh
Div:D15B
Roll No:56

PWA Practical 8

Aim:

To convert a Vite React application into a Progressive Web App (PWA) by registering a service worker and verifying its installation and offline capabilities.

Theory:

A **Progressive Web App (PWA)** is a type of web application that delivers a native app-like experience using modern web capabilities. PWAs are installable, work offline, and can send push notifications, providing improved performance and user engagement.

To create a PWA using **Vite** and **React**, the **vite-plugin-pwa** is typically used. This plugin allows automatic service worker registration and caching strategies.

Key Concepts:

- **Service Worker:** A script that runs in the background and handles caching, push notifications, and offline capabilities.
- **Manifest File:** Describes the app's metadata such as name, icons, theme color, and display mode, allowing the app to be installed.
- **registerSW():** A function used to register the service worker provided by the **vite-plugin-pwa**.

Steps to Make Your Vite React App a PWA

1 Register the Service Worker

1. Open **main.jsx** (or wherever React renders the app).
2. Import the **registerSW** function from **virtual:pwa-register**.
3. Call **registerSW()** to activate the service worker.
4. This ensures that the app works offline and updates correctly.

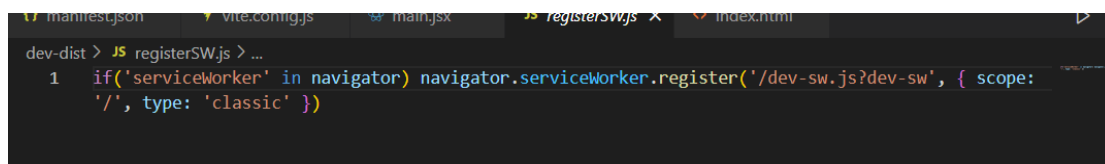
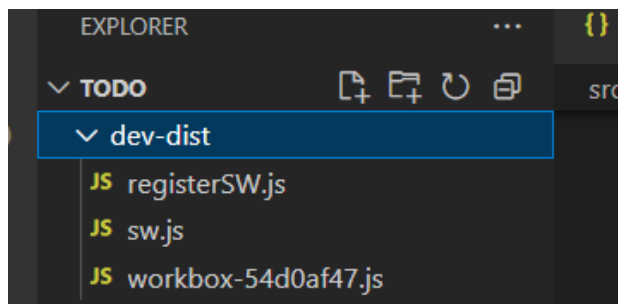
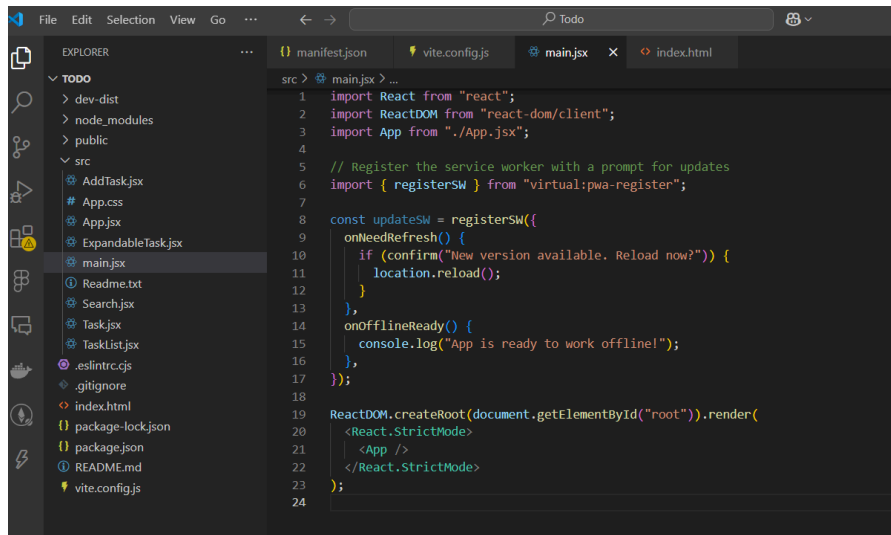
2 Verify PWA Installation

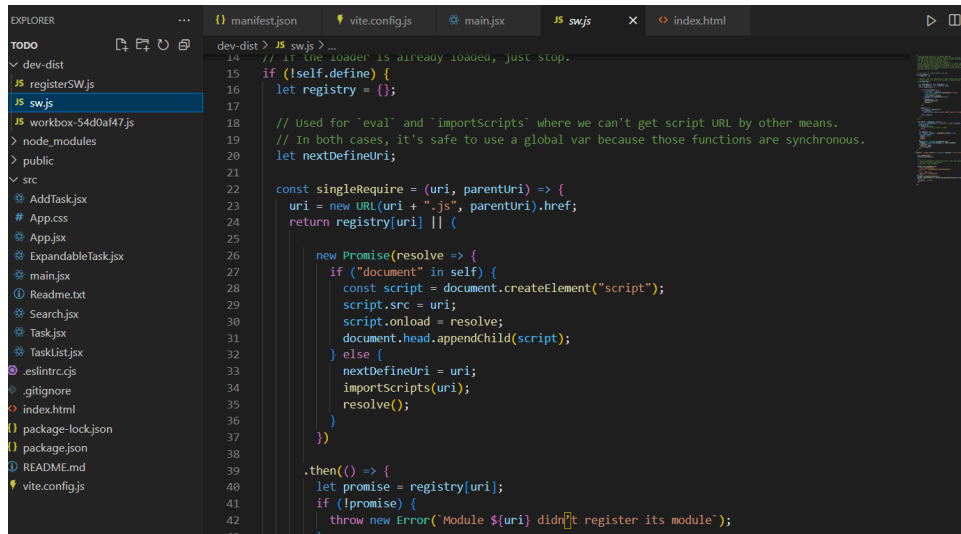
1. Start the Vite development server.
2. Open Chrome DevTools (**F12**) → Go to Application → Manifest.

3. Check if the Install button appears in the address bar.
4. Install the app and check if it works like a standalone application.

3 Test Service Worker Activation

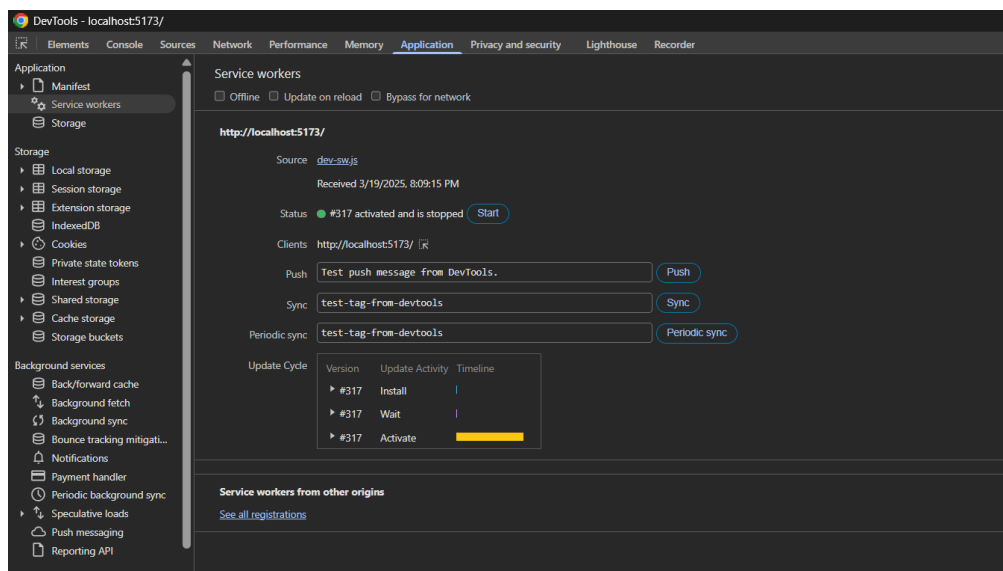
1. Open Chrome DevTools → Application → Service Workers.
2. Check if the service worker is registered.
3. Enable Offline mode and refresh the page to test caching.
4. If the app loads, the service worker is working successfully!





The screenshot shows the VS Code editor with the file explorer on the left and the editor window on the right. The file explorer shows the project structure with files like `manifest.json`, `vite.config.js`, `main.jsx`, `sw.js`, and `index.html`. The editor window displays the content of `sw.js`, which is a service worker script. The script includes comments and code for registering the service worker, handling requests, and caching resources. The code is as follows:

```
dev-dist > JS sw.js > ...
14 // If the loader is already loaded, just stop.
15 if (!self.define) {
16   let registry = {};
17
18   // Used for "eval" and "importScripts" where we can't get script URL by other means.
19   // In both cases, it's safe to use a global var because those functions are synchronous.
20   let nextDefineUri;
21
22   const singleRequire = (uri, parentUri) => {
23     uri = new URL(uri + ".js", parentUri).href;
24     return registry[uri] || (
25       new Promise(resolve => {
26         if ("document" in self) {
27           const script = document.createElement("script");
28           script.src = uri;
29           script.onload = resolve;
30           document.head.appendChild(script);
31         } else {
32           nextDefineUri = uri;
33           importScripts(uri);
34           resolve();
35         }
36       })
37     );
38   };
39
40   .then(() => {
41     let promise = registry[uri];
42     if (!promise) {
43       throw new Error(`Module ${uri} didn't register its module`);
44     }
45   });
46 }
```



Conclusion:

By registering the service worker using `registerSW()` in a Vite React app, we successfully converted the application into a PWA. We verified its installation through Chrome DevTools and confirmed offline functionality by enabling offline mode and observing cached content loading correctly. This demonstrates the app's ability to function as a standalone, installable, and offline-capable web application, enhancing both performance and user experience.