PWA Experiment -9

Jai Navani

D15A 30

Aim: To implement Service worker events like fetch, sync and push for E-commerce PWA

Theory:

Service Worker

Service Worker is a script that works on browser background without user interaction independently. Also, It resembles a proxy that works on the user side. With this script, you can track network traffic of the page, manage push notifications and develop “offline first” web applications with Cache API.

Things to note about Service Worker:

- A service worker is a programmable network proxy that lets you control how network requests from your page are handled.
- Service workers only run over HTTPS. Because service workers can intercept network requests and modify responses, "man-in-the-middle" attacks could be very bad.
- The service worker becomes idle when not in use and restarts when it's next needed. You cannot rely on a global state persisting between events. If there is information that you need to persist and reuse across restarts, you can use IndexedDB databases.
- Service workers make extensive use of promises, so if you're new to promises, then you should stop reading this and check out Promises, an introduction.

Fetch Event

You can track and manage page network traffic with this event. You can check

existing cache, manage “cache first” and “network first” requests and return a response that you want.

Of course, you can use many different methods but you can find in the following example a “cache first” and “network first” approach. In this example, if the request’s and current location’s origin are the same (Static content is requested.), this is called “cacheFirst” but if you request a targeted external URL, this is called “networkFirst”.

- **CacheFirst** - In this function, if the received request has cached before, the cached response is returned to the page. But if not, a new response requested from the network.
- **NetworkFirst** - In this function, firstly we can try getting an updated response from the network, if this process completed successfully, the new response will be cached and returned. But if this process fails, we check whether the request has been cached before or not. If a cache exists, it is returned to the page, but if not, this is up to you. You can return dummy content or information messages to the page.

Sync Event

Background Sync is a Web API that is used to delay a process until the Internet connection is stable. We can adapt this definition to the real world; there is an e-mail client application that works on the browser and we want to send an email with this tool. Internet connection is broken while we are writing e-mail content and we didn’t realize it. When completing the writing, we click the send button.

Here is a job for the Background Sync.
The following view shows the classical process of sending email to us. If the Internet Connection is broken, we can’t send any content to Mail Server.

Push Event

This is the event that handles push notifications that are received from the server.

You can apply any method with received data.

We can check in the following example.
“Notification.requestPermission();” is the necessary line to show notification to the user. If you don’t want to show any notification, you don’t need this line.

In the following code block is in sw.js file. You can handle push notifications with this event. In this example, I kept it simple. We send an object that has “method” and “message” properties. If the method value is “pushMessage”, we open the information notification with the “message” property.

CODE:-

Changes Made in Index.html

```
<body>

    <script>
        if ("serviceWorker" in navigator) {
          navigator.serviceWorker.register("sw.js")
            .then((reg) => {
              console.log("Service Worker registered!", reg);

              // Background Sync
              if ("SyncManager" in window) {
                navigator.serviceWorker.ready.then((swReg) => {
                  return swReg.sync.register("sync-data")
                    .then(() => console.log("Sync event registered"))
                    .catch((err) => console.log("Sync registration
failed", err));
                });
              }

              // Push Notifications
              if ("PushManager" in window) {
                Notification.requestPermission().then((permission) => {
                  if (permission === "granted") {
```

```
                    console.log("Push notifications allowed!");
                  } else {
                    console.log("Push notifications denied!");
                  }
                });
              }
            })
            .catch((err) => console.log("Service Worker registration
failed!", err));
        }

        // Function to send a message to the Service Worker
        function sendMessageToServiceWorker(action) {
          if (navigator.serviceWorker.controller) {
            navigator.serviceWorker.controller.postMessage({ action:
action });
          } else {
            console.log("Service Worker is not active yet.");
          }
        }

        // Detect clicks on "Contact Us" navigation link
        document.getElementById("contactUs")?.addEventListener("click",
() => {
          sendMessageToServiceWorker("contact_us_nav_clicked");
        });

        // Detect clicks on "Contact Us" button

document.getElementById("contactUsBtn")?.addEventListener("click", () =>
{
          sendMessageToServiceWorker("contact_us_btn_clicked");
        });
      </script>


    <!-- **** Header Section **** -->

    <header>
        <nav class="navbar navbar-expand-lg navigation-wrap">
```

```
            <div class="container">
                <a class="navbar-brand" href="#">
                    <img src="assets/images/logo.png" alt=""
class="logo">
                </a>
                <button class="navbar-toggler" type="button"
data-bs-toggle="collapse" data-bs-target="#navbarNav"
                    aria-controls="navbarNav" aria-expanded="false"
aria-label="Toggle navigation">
                    <span class="navbar-toggler-icon"></span>
                </button>
                <div class="collapse navbar-collapse" id="navbarNav">
                    <ul class="navbar-nav ms-auto">
                        <li class="nav-item">
                            <a class="nav-link" href="#home">Home</a>
                        </li>
                        <li class="nav-item">
                            <a class="nav-link" href="#about">About
Us</a>
                        </li>
                        <li class="nav-item">
                            <a class="nav-link"
href="#explore-food">Explore Food</a>
                        </li>
                        <li class="nav-item">
                            <a class="nav-link"
href="#testinomial">Reviews</a>
                        </li>

                        <!-- Contact Us Nav Link -->
                        <li class="nav-item">
                            <a class="nav-link" href="#contact"
id="contactUs">Contact</a>
                        </li>

                        <!-- Contact Us Button -->
                        <li>
                            <button class="main-btn"
id="contactUsBtn">1200 345 123</button>
                        </li>
```

```
                    </ul>
                </div>
            </div>
        </nav>
    </header>
```

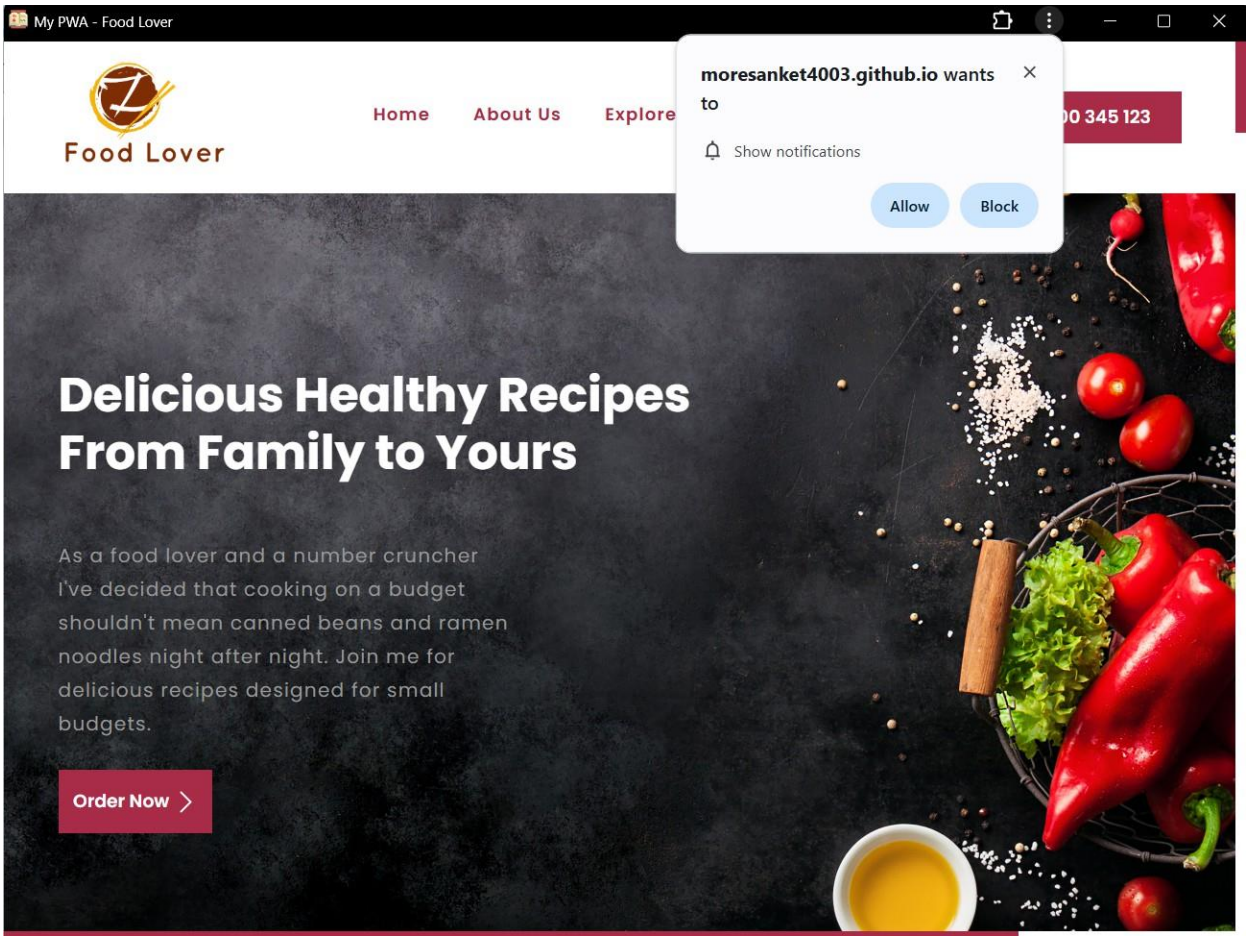
New sw.js

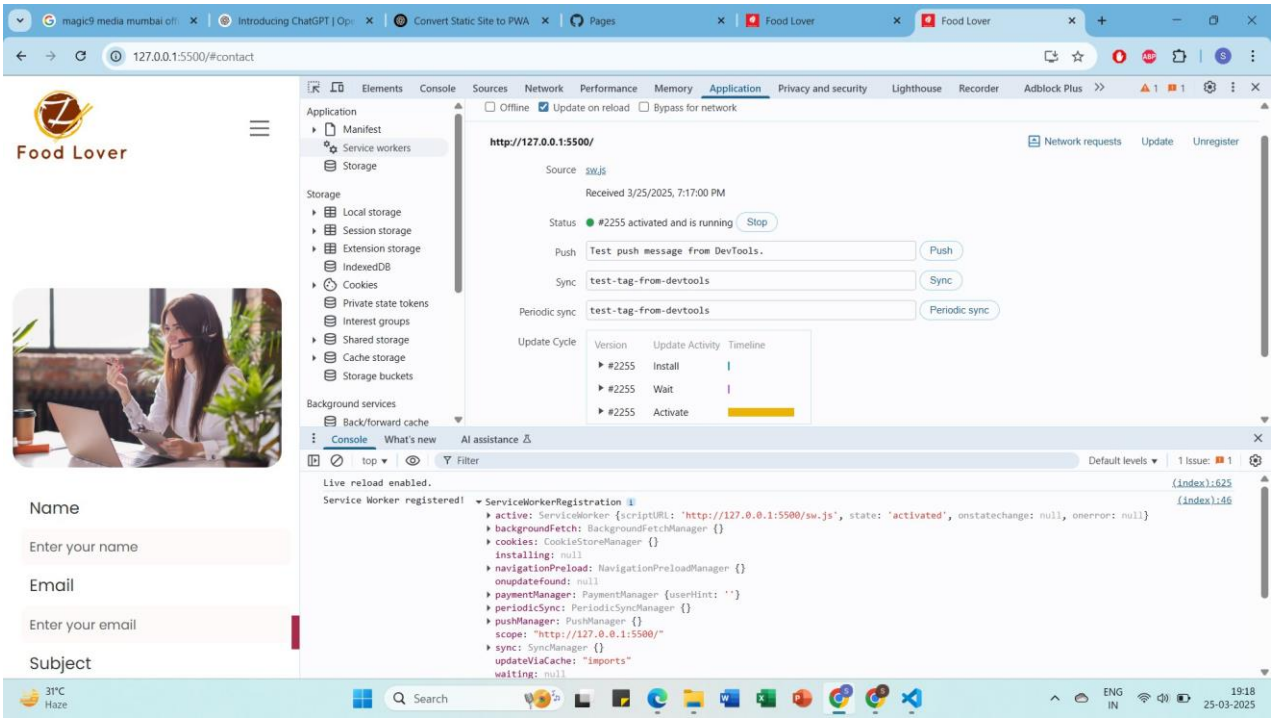
```
const CACHE_NAME = "pwa-cache-v1";
const urlsToCache = [
  "/index.html",
  "/assets/css/style.css",   // Update with actual CSS filename
  "/assets/images/icon-192x192.png",
  "/assets/images/icon-512x512.png"
];

// Install event: Caches assets
self.addEventListener("install", (event) => {
  event.waitUntil(
    caches.open(CACHE_NAME).then((cache) => {
      return cache.addAll(urlsToCache);
    })
  );
});

// Fetch event: Serve cached files when offline
self.addEventListener("fetch", (event) => {
  event.respondWith(
    caches.match(event.request).then((response) => {
      return response || fetch(event.request);
    })
  );
});

// Sync event: Background sync (requires registration in main script)
self.addEventListener("sync", (event) => {
  if (event.tag === "sync-data") {
    event.waitUntil(syncDataFunction());
```

```
  }
});

async function syncDataFunction() {
  console.log("Syncing data in the background...");
  // Example: Send stored requests to server when online
}

// Push event: Handle push notifications
self.addEventListener("push", (event) => {
  const options = {
    body: "New message received!",
    icon: "/assets/images/icon-192x192.png",
    badge: "/assets/images/icon-192x192.png"
  };
  event.waitUntil(
    self.registration.showNotification("PWA Notification", options)
  );
});

// Message event: Handle messages from the main thread
self.addEventListener("message", (event) => {
  if (event.data && event.data.action) {
    console.log(`Service Worker: User clicked on ${event.data.action}`);

    // Example: Show a notification when "Contact Us" is clicked
    if (event.data.action === "contact_us_nav_clicked" ||
event.data.action === "contact_us_btn_clicked") {
      self.registration.showNotification("Contact Us Clicked", {
        body: "User clicked the Contact Us button!",
        icon: "/assets/images/icon-192x192.png"
      });
    }
  }
});

// Activate event: Clean up old caches
self.addEventListener("activate", (event) => {
  event.waitUntil(
    caches.keys().then((cacheNames) => {
```

```
      return Promise.all(
        cacheNames
          .filter((cacheName) => cacheName !== CACHE_NAME)
          .map((cacheName) => caches.delete(cacheName))
      );
    })
  );
});
```

OUTPUT:-






```
waiting: null
▸ [[Prototype]]: ServiceWorkerRegistration
```

```
Sync event registered
```

```
Syncing data in the background...
```

```
Live reload enabled.
```

```
Service Worker registered! ▸ ServiceWorkerRegistration
```

```
Sync event registered
```

```
Syncing data in the background...
```

```
Push notifications allowed!
```