Name:Bhushan Mukund Kor **Academic Year:2024-2025**

Division: D15C **Roll No: 28**

Experiment 9

Aim: To implement Service worker events like fetch, sync and push for ToDo PWA.

Theory:

Service Worker Lifecycle and Architecture

Service Workers act as programmable intermediaries between web applications and the network, running in a separate thread from the main browser context. This separation enables three key capabilities:

- Interception and modification of network requests
- Persistent background operations, such as sync and push
- Event-driven execution, triggered by browser or network events

Event-Driven Architecture

Service Workers follow an observer design pattern, reacting to specific lifecycle and runtime events:

- Installation: Handles initial setup and pre-caching of critical assets
- Activation: Manages cache versioning and removes outdated resources
- Runtime Events: Intercept and respond to network requests, push messages, and background sync triggers

Core Events and Their Mechanics

Fetch Event

Intercepts all outbound network requests within the defined scope, supporting:

- Caching Strategies
    - Cache-first: Prioritizes offline availability
    - Network-first: Ensures up-to-date content
    - Stale-while-revalidate: Combines speed and freshness
- Request/Response Manipulation
    - URL rewriting
    - Header adjustments
    - Custom response injection

Background Sync

Provides reliable task execution when connectivity is restored:

- Task Queuing: Uses tags to register sync events
- Deferred Execution: Waits for stable network connection
- Automatic Retry: Re-attempts failed sync operations

Push Notifications

Enables background messaging with minimal resource usage:

- Subscription Management: Registers endpoints with encryption

- Payload Handling: Parses and decrypts incoming data
- Notification Display: Interfaces with the browser's Notification API

Technical Constraints

Security Requirements

- HTTPS is mandatory
- Restricted to defined scopes
- User permission is required for features like notifications

Resource & Execution Limits

- Bound by memory and storage quotas
- Limited background processing time
- Process priority managed by the browser

Lifecycle Management

- Versioned updates with controlled activation
- Graceful shutdowns and restarts
- Option to persist data across restarts

Code:

1. **Fetch Event (offline caching)**

```
self.addEventListener('fetch', (event) => {
  event.respondWith(
    caches.match(event.request).then((cachedResponse) => {
      if (cachedResponse) {
        console.log(`Service Worker: Serving from cache - ${event.request.url}`);
        return cachedResponse;
      }

      return fetch(event.request)
        .then((networkResponse) => {
          // Only cache valid GET responses
          if (
            networkResponse &&
            networkResponse.status === 200 &&
            event.request.method === 'GET'
          ) {
            const responseClone = networkResponse.clone(); // 👈 Clone before reading
            caches.open(CACHE_NAME).then((cache) => {
              cache.put(event.request, responseClone);
            });
          }

          return networkResponse;
```

```
        })
        .catch((error) => {
          console.error(`Service Worker: Fetch failed - ${event.request.url}`, error);
          return caches.match('/Flutter_PWA/taskverse/offline.html');
        });
    })
  );
});
```

2. **Background Sync**

```
// Background Sync event
self.addEventListener('sync', (event) => {
  if (event.tag === 'sync-taskverse-data') {
    event.waitUntil(
      // Simulated sync task
      (async () => {
        console.log('Service Worker: Background sync triggered!');
        // Here you could sync offline-stored tasks to Firebase/Firestore
      })()
    );
  }
});
```

3. **Push Notifications**

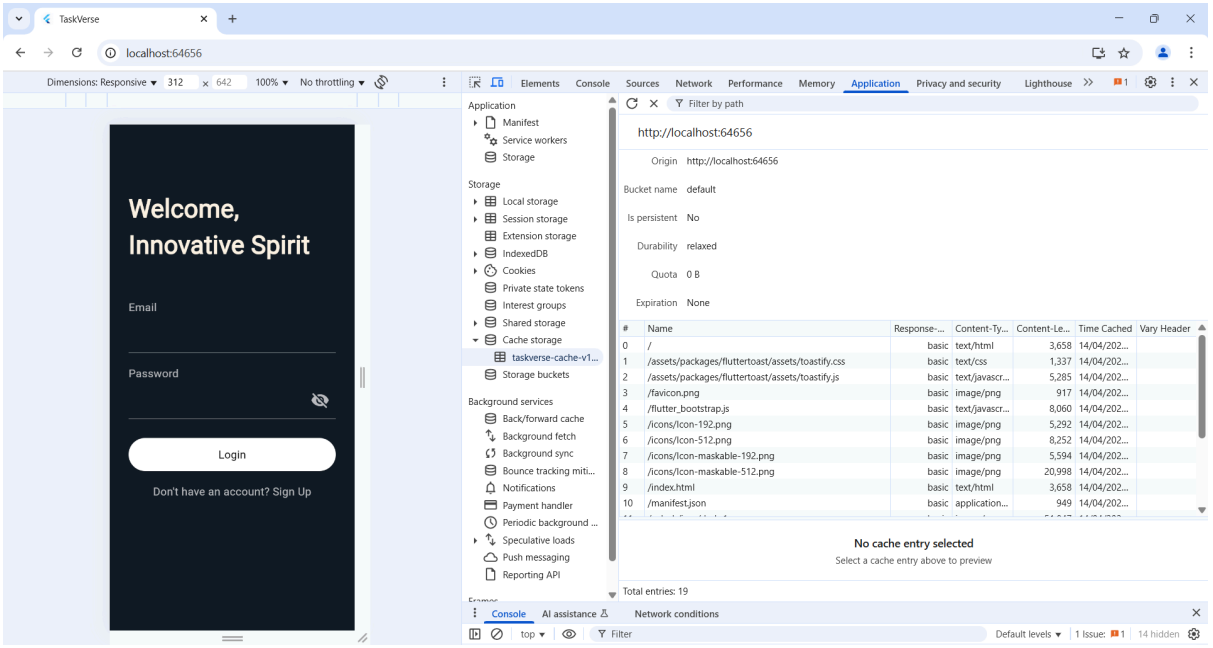
```
// Push Notification event
self.addEventListener('push', (event) => {
  console.log('Service Worker: Push Received.');
  let data = {};
  if (event.data) {
    data = event.data.json();
  }

  const title = data.title || 'Taskverse Notification';
  const options = {
    body: data.body || 'You have a new task update.',
    icon: '/Flutter_PWA/taskverse/icons/Icon-192.png',
    badge: '/Flutter_PWA/taskverse/icons/Icon-192.png',
  };

  event.waitUntil(self.registration.showNotification(title, options));
});
```

Name:Bhushan Mukund Kor **Academic Year:2024-2025**

Division: D15C **Roll No: 28**



GitHub Link: https://github.com/BKCODE2003/Flutter_PWA

Conclusion: By implementing service worker events such as fetch, sync, and push, we have enhanced the reliability, responsiveness, and interactivity of the eCommerce PWA. These features allow the app to serve cached content offline, sync data in the background, and send real-time notifications to users. Together, they contribute to a seamless and engaging user experience, making the PWA behave more like a native application, even under challenging network conditions.