Name: Mohit Kerkar **Div: D15C** **Roll No: 23**

MAD and PWA Lab
EXPERIMENT NO. 9
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.

to generate flutter_service_worker.js, run flutter build web

Name: Mohit Kerkar **Div: D15C** **Roll No: 23**

```
C:\Users\Dell\Desktop\society_profiles>flutter build web

Warning: In index.html:43: Manual service worker registration deprecated. Use flutter.js service worker bootstrapping
instead. See https://docs.flutter.dev/platform-integration/web/initialization for more details.
Font asset "CupertinoIcons.ttf" was tree-shaken, reducing it from 257628 to 1172 bytes (99.5% reduction). Tree-shaking
can be disabled by providing the --no-tree-shake-icons flag when building your app.
Font asset "MaterialIcons-Regular.otf" was tree-shaken, reducing it from 1645184 to 8944 bytes (99.5% reduction).
Tree-shaking can be disabled by providing the --no-tree-shake-icons flag when building your app.
Compiling lib\main.dart for the Web...                             84.7s
√ Built build\web

C:\Users\Dell\Desktop\society_profiles>
```

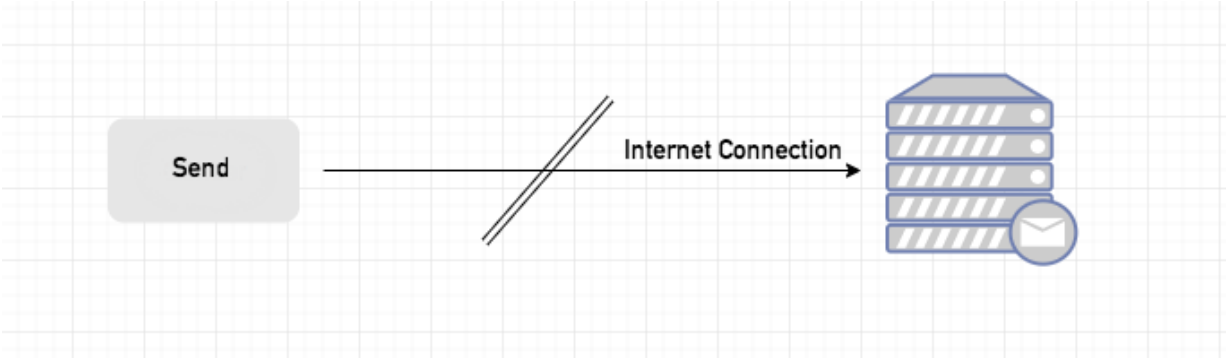
fetch

```
self.addEventListener("fetch", (event) => {
  if (event.request.method !== 'GET') {
    return;
  }
  var origin = self.location.origin;
  var key = event.request.url.substring(origin.length + 1);
  // Redirect URLs to the index.html
  if (key.indexOf('?v=') != -1) {
    key = key.split('?v=')[0];
  }
  if (event.request.url == origin || event.request.url.startsWith(origin + '/#') || key == '') {
    key = '/';
  }
  // If the URL is not the RESOURCE list then return to signal that the
  // browser should take over.
  if (!RESOURCES[key]) {
    return;
  }
  // If the URL is the index.html, perform an online-first request.
  if (key == '/') {
    return onlineFirst(event);
  }
  event.respondWith(
    fetch(event.request).then((response) => {
      return caches.open(CACHE_NAME).then((cache) => {
        cache.put(event.request, response.clone());
        return response;
      });
    }).catch(() => {
      return caches.open(CACHE_NAME).then((cache) => cache.match(event.request));
    })
  );

});
```

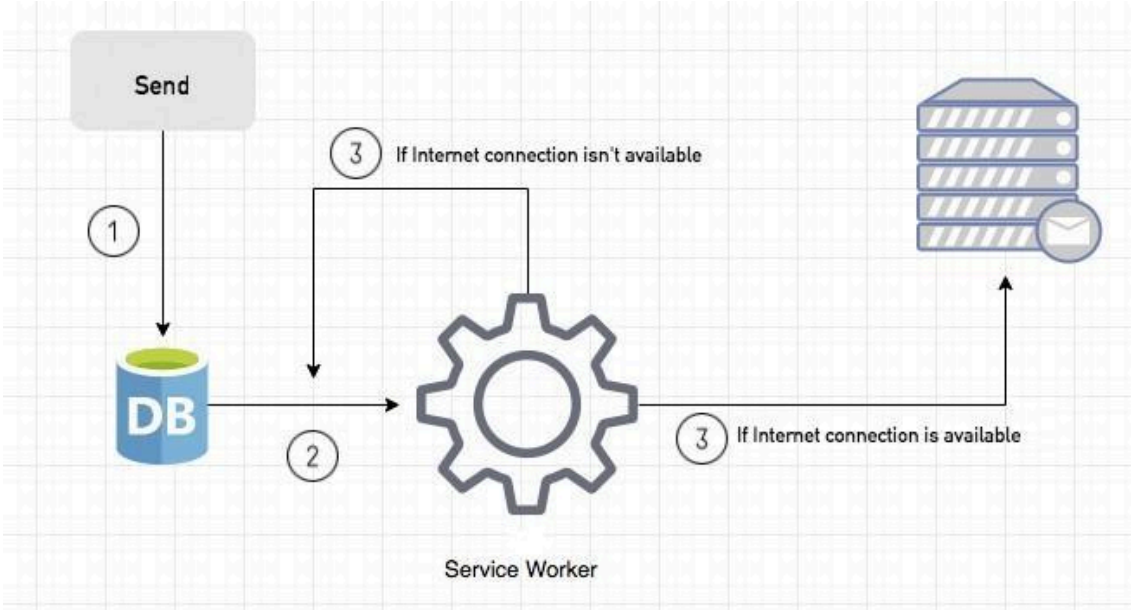
Sync Event

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.



Here, you can create any scenario for yourself. A sample is in the following for this case.



1. When we click the "send" button, email content will be saved to IndexedDB.
2. Background Sync registration.
3. If the Internet connection is available, all email content will be read and sent to Mail Server.

If the Internet connection is unavailable, the service worker waits until the connection is available even though the window is closed. When it is available, email content will be sent to Mail Server.
You can see the working process within the following code block.

Event Listener for Background Sync Registration
added in main.dart

```
// Register the background sync event using service workers
void registerSync() {
  if (kIsWeb) {
    window.navigator.serviceWorker?.ready.then((registration) {
      registration.sync?.register('sync-data').then((_) {
        print("Sync registered successfully");
      }).catchError((error) {
        print("Sync registration failed: $error");
      });
    }).catchError((error) {
      print("Service worker not ready: $error");
    });
  }
}
```

inside flutter_service_worker.js

```
self.addEventListener('sync', (event) => {
  if (event.tag === 'sync-data') {
    event.waitUntil(syncPendingData());
  }
});

async function syncPendingData() {
  await fetch('/api/sync', { method: 'POST' });
}
```

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.

Name: Mohit Kerkar **Div: D15C** **Roll No: 23**

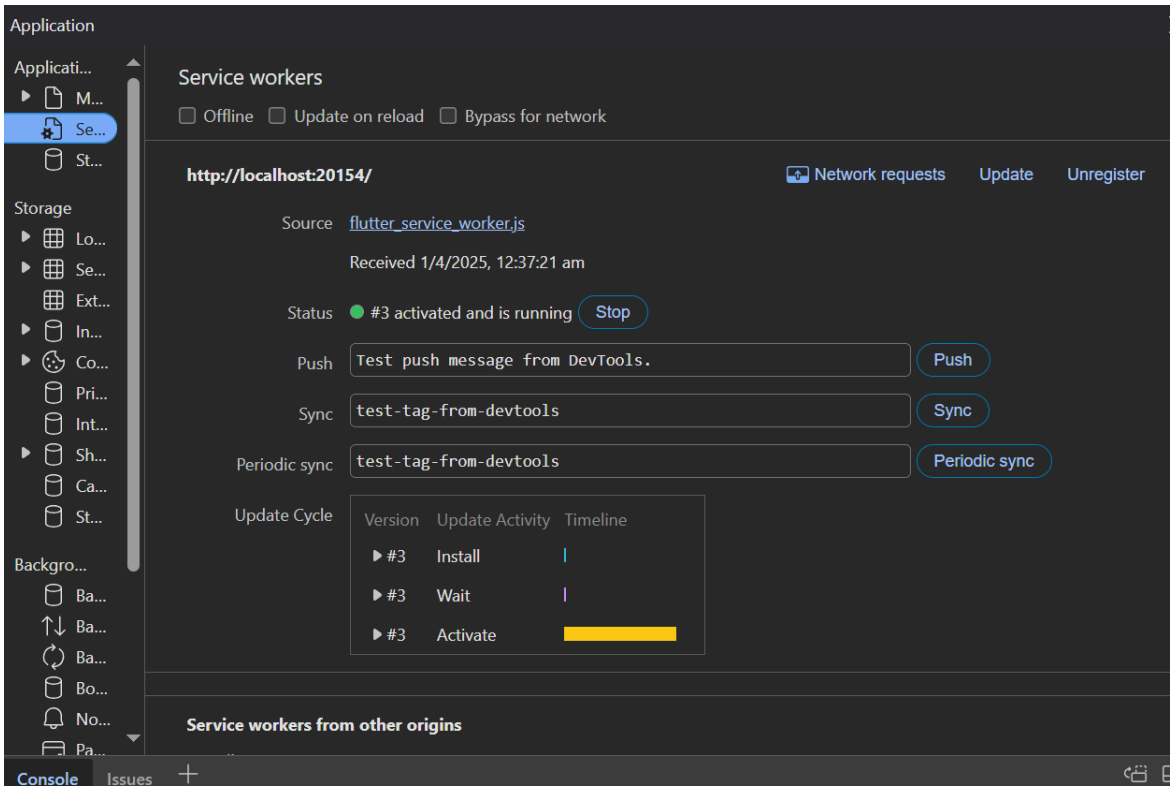
In the following code block is in sw.js file. You can handle push notifications with this event. In this example, I kept it simple. We send an object that has “method” and “message” properties. If the method value is “pushMessage”, we open the information notification with the “message” property.

```javascript
self.addEventListener('push', (event) => {
  const options = {
    body: 'New update in your society!',
    icon: '/icons/icon-192x192.png',
    badge: '/icons/icon-72x72.png'
  };
  event.waitUntil(
    self.registration.showNotification('Society App', options)
  );
});
```

```dart
Future<void> subscribeToPushNotifications() async {
  FirebaseMessaging messaging = FirebaseMessaging.instance;

  // Get the FCM token, used for push notifications
  String? token = await messaging.getToken();
  if (token != null) {
    print('FCM Token: $token');  // Token for push notifications
  }

  // Handling background messages
  FirebaseMessaging.onBackgroundMessage((message) async {
    print('Received background message: $message');
    // Handle background notification here
  });

  // Handling foreground messages
  FirebaseMessaging.onMessage.listen((message) {
    print('Received foreground message: $message');
    // Handle foreground notification here
  });
}
```

Name: Mohit Kerkar **Div: D15C** **Roll No: 23**

You can use Application Tab from Chrome Developer Tools for testing push notification.



Conclusion:

Implementing these events boosted our PWA's responsiveness, reliability, and user communication via push alerts.