Aim : To code and register a service worker, and complete the install and activation process for a new service worker for the E-commerce PWA.

Theory :

A service worker is a JavaScript file that runs in the background of a web application, separate from the main browser thread. It acts as a proxy between the web application and the network, allowing developers to intercept and handle network requests, cache resources for offline use, and provide a seamless user experience even when the network is unreliable or unavailable.

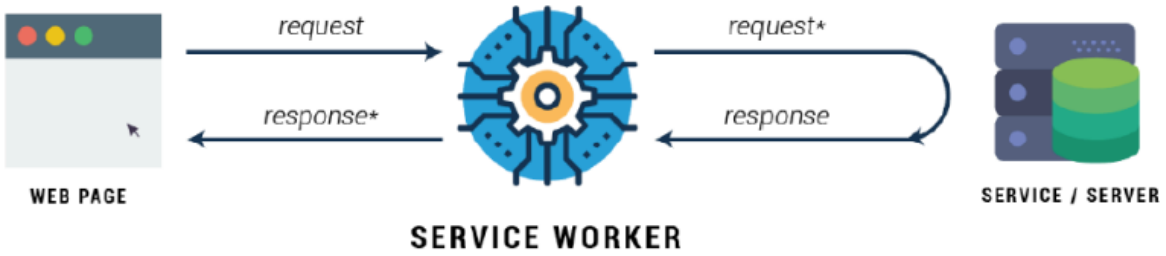
Service workers enable Progressive Web Apps (PWAs) to offer features such as offline functionality, push notifications, and background synchronization. They are a key component of PWAs, helping to make web apps faster, more reliable, and engaging for users. In a service worker, events such as fetch, sync, and push play crucial roles in enabling various functionalities.

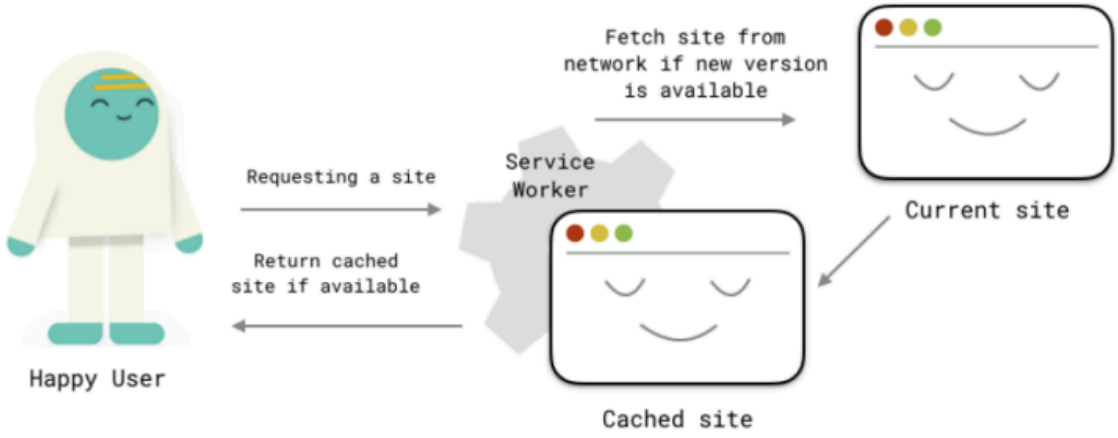
Here's an overview of each:

1. Fetch Event: The fetch event occurs when the browser fetches a resource (e.g., an image, script, or API request). Service workers can intercept fetch requests and provide custom responses, allowing for advanced caching strategies (like serving from a cache or making a network request). This enables offline functionality and performance optimizations.

2. Sync Event: The sync event allows service workers to perform background synchronization tasks, even when the app is not in use or the browser is closed. This is useful for tasks like updating data from a server or sending analytics data.

3. Push Event: The push event occurs when a push notification is received from a server. Service workers can intercept push events and show notifications to users, even when the app is not open.





Steps :

Fetch Event:

This event is triggered whenever a resource is fetched from the network.

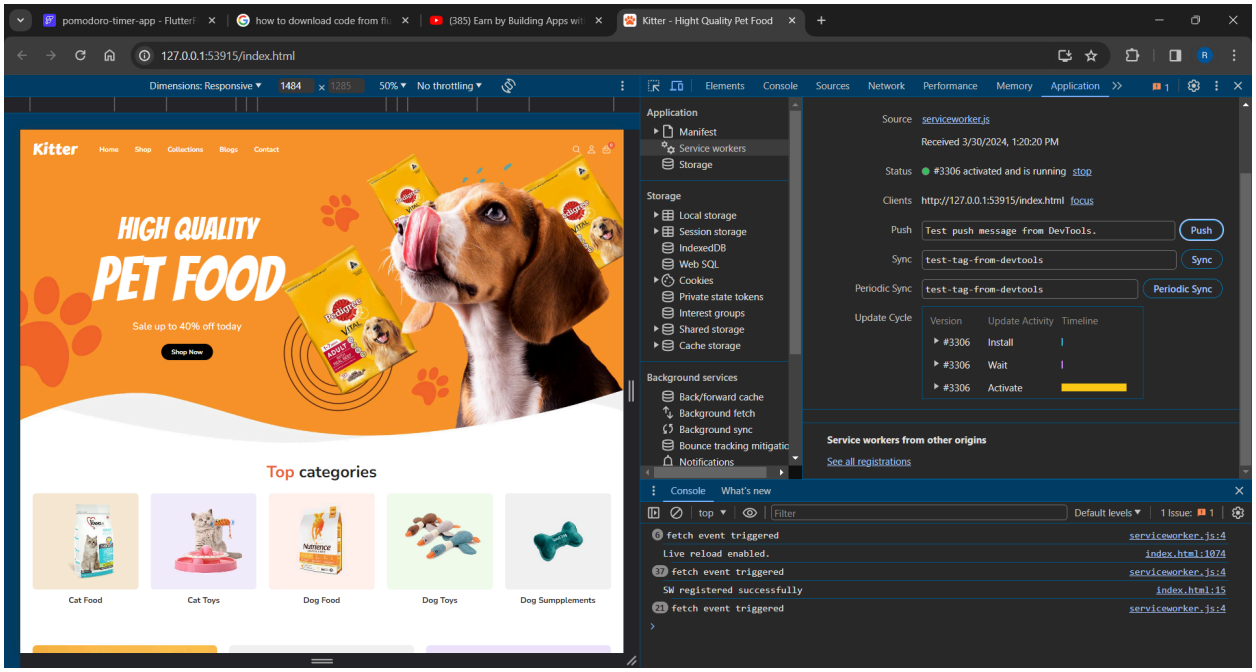
- CacheFirst - In this function, if the received request has cached before, the cached response is returned to the page. But if not, a new response requested from the network.
- NetworkFirst - In this function, firstly we can try getting an updated response from the network, if this process completed successfully, the new response will be cached and returned.

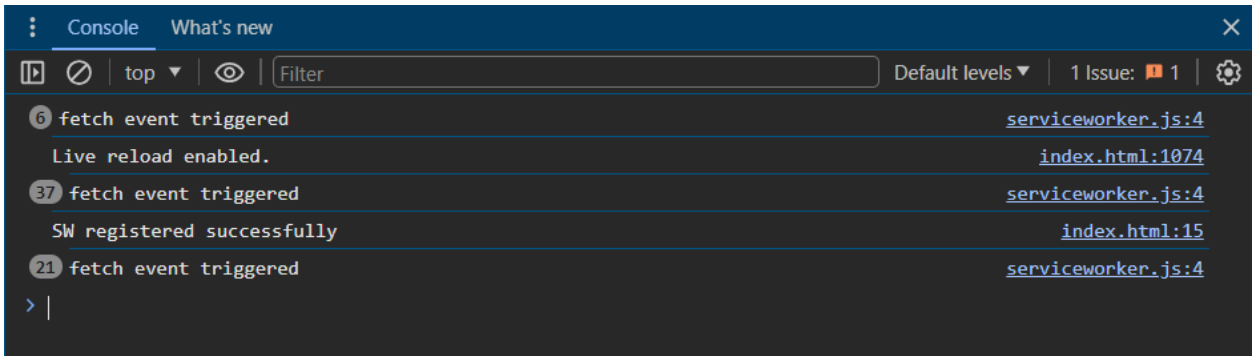
But if this process fails, we check whether the request has been cached before or not. If a cache exists, it is returned to the page, but if not, this is up to you. You can return dummy content or information messages to the page.

Code:

serviceworker.js

```
self.addEventListener('fetch', function (event) {
  console.log('fetch event triggered');
  event.respondWith(
  caches.match(event.request).then(function (response) {
  return response || fetch(event.request);
  })
  );
  });
```



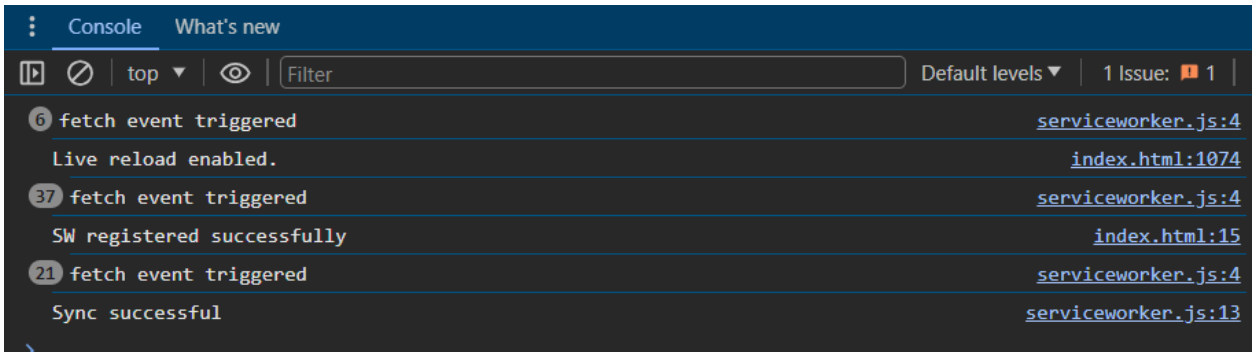


Sync Event: This event is triggered when the service worker receives a sync event from the browser.

When we click the “send” button, email content will be saved to IndexedDB. Background Sync registration.

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.

```
self.addEventListener("sync", function (event) {
  console.log("Sync successful");
  if (event.tag === "syncData") {
    event.waitUntil(syncDataWithServer());
  }
});
```



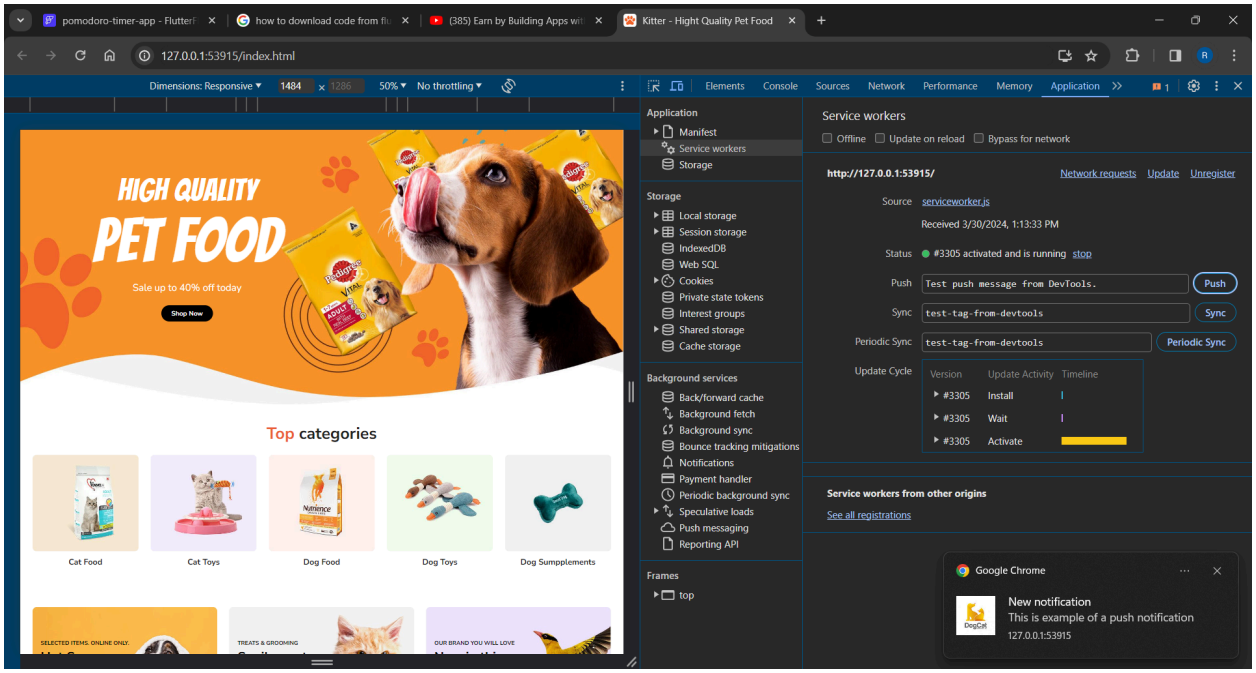
Push Event:
This event is triggered when a push notification is received. We can check in the following example. “Notification.requestPermission();” is the necessary line to show notification to the user.
If you don’t want to show any notification, you don’t need this line. In the following code block 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.

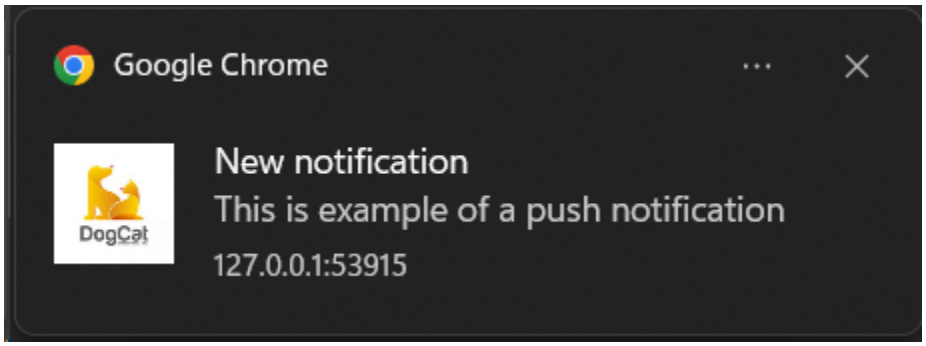
Allow notifications :
Index.js:

```
Notification.requestPermission().then(function(permission) {
  if (permission === 'granted') {
  // Permission granted, you can now show notifications
  }
  });
```

serviceworker.js:

```
self.addEventListener('push', function(event) {
  const options = {
  body: 'This is example of a push notification',
  icon: './assets/images/brand-1.png'
  };
  event.waitUntil(
  self.registration.showNotification('New notification', options)
  );
  });
```





Conclusion : Hence we have understood and implemented the working of service worker event listeners such as ‘fetch’, ‘sync’, and ‘push’ notifications in E-commerce Progressive web application.