Amit Nayak 46/D15B

Experiment No. 9

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 :

Service Worker

Service Worker is a script that runs in the background of a web application, acting as a proxy between the application and the network. It enables features like offline browsing, push notifications, and background synchronization. Service workers are event-driven and can intercept and modify network requests, cache resources for offline use, and manage background tasks. They are only available to HTTPS sites for security reasons.

Fetch Event

The Fetch Event is a type of event that occurs in the context of a Service Worker. It is triggered whenever a network request is made from a web page that is controlled by a Service Worker. The main purpose of the Fetch Event is to give the Service Worker the opportunity to intercept and handle the request.

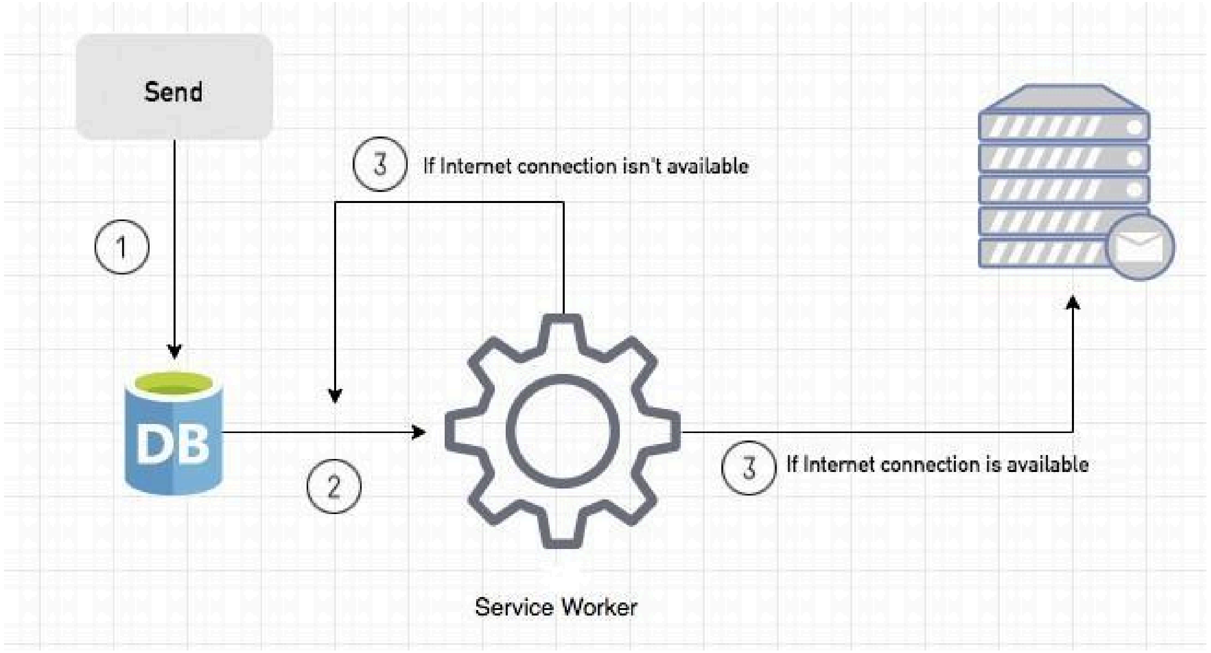
When a Fetch Event is fired, the Service Worker can respond to the request in a number of ways. It can fetch the requested resource from the network, it can retrieve it from the cache, or it can generate a synthetic response. This allows the Service Worker to implement strategies like "cache first" or "network first" for handling requests.

The Fetch Event is a powerful feature of the Service Worker API that enables advanced caching strategies and efficient network request handling, ultimately leading to improved performance and offline capabilities for web applications.

Sync Event

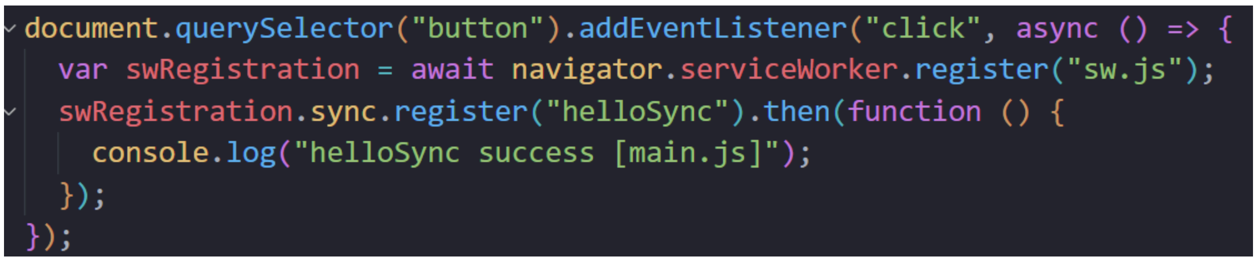
Background Sync is a Web API that enables delaying a process until a stable Internet connection is available. This is similar to a scenario where an email client application running in a browser is used to compose an email. If the Internet connection is lost while composing the email, the user may not be aware of it. When attempting to send the email, the lack of connectivity prevents the email from being sent.

Background Sync can address this issue by allowing the email to be sent once the Internet connection is reestablished. This ensures that the user's action is not lost and that the email is sent successfully when connectivity is restored.
Here, you can create any scenario for yourself. A sample is in the following for this case.


Send
1
DB
2
3 If Internet connection isn't available
3 If Internet connection is available
Service Worker

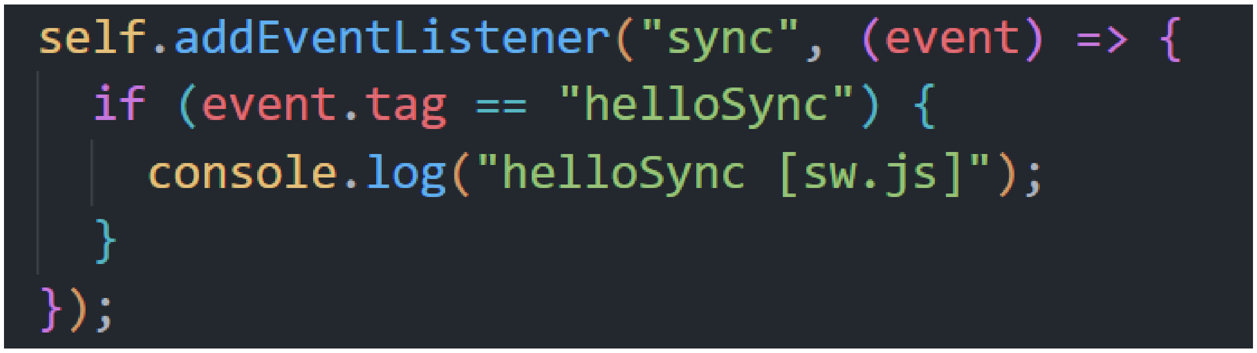
- When we click the “send” button, email content will be saved to IndexedDB.
- Background Sync registration.
- If the Internet connection is available, all email content will be read and sent to Mail Server.
- If the Internet connection is unavailable, the service worker waits until the connection is available even though the window is closed. When it is available, email content will be sent to Mail Server.
- You can see the working process within the following code block.

Event Listener for Background Sync Registration:



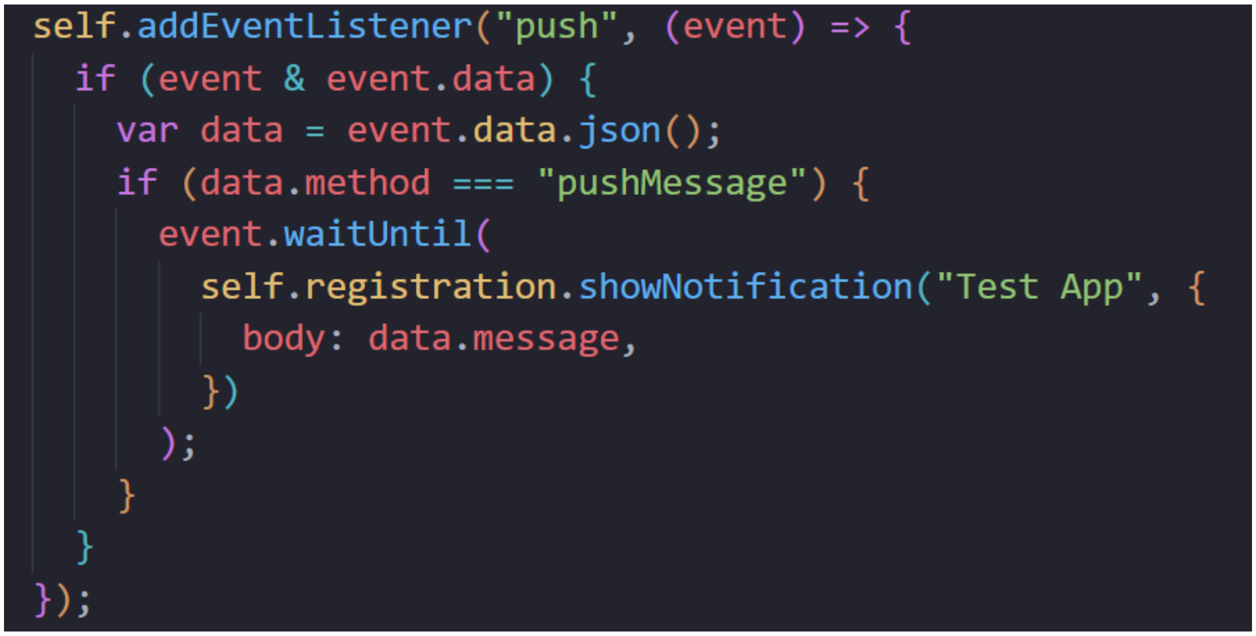
```
document.querySelector("button").addEventListener("click", async () => {
  var swRegistration = await navigator.serviceWorker.register("sw.js");
  swRegistration.sync.register("helloSync").then(function () {
    console.log("helloSync success [main.js]");
  });
});
```

Event Listener for sw.js



```
self.addEventListener("sync", (event) => {
  if (event.tag == "helloSync") {
    console.log("helloSync [sw.js]");
  }
});
```

Push Event
The Push Event is triggered when a push notification is received from the server. It allows developers to handle the incoming data and perform actions based on the received notification.
For example, in a service worker (sw.js), you can use the Push Event to display a notification to the user. First, you would request permission to show notifications using Notification.requestPermission(). Then, when a push notification is received, you can extract the message from the notification and display it to the user using the showNotification() method.



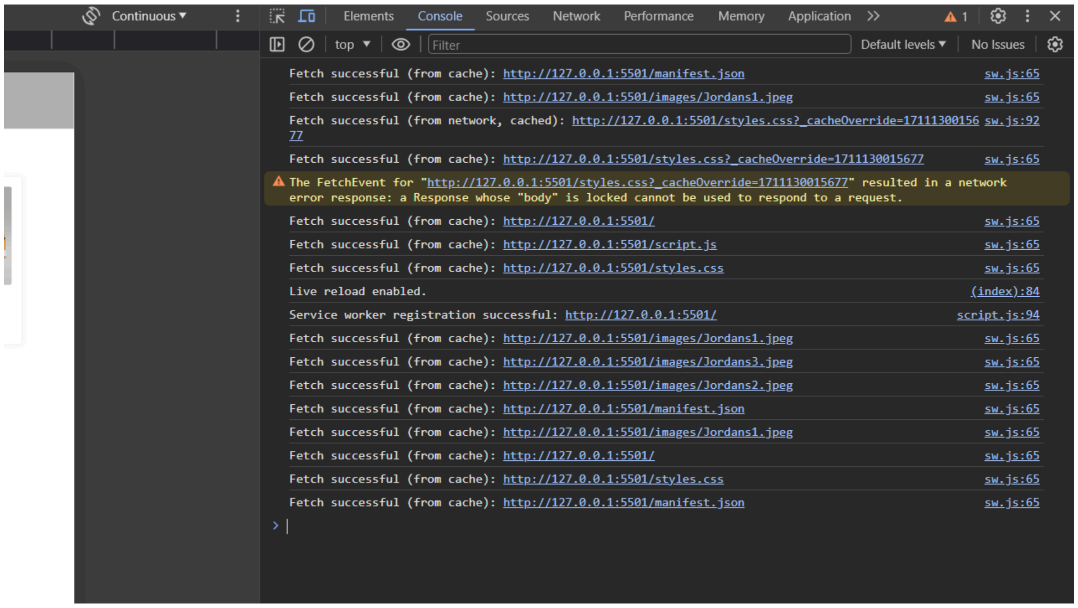
```
self.addEventListener("push", (event) => {
  if (event & event.data) {
    var data = event.data.json();
    if (data.method === "pushMessage") {
      event.waitUntil(
        self.registration.showNotification("Test App", {
          body: data.message,
        })
      );
    }
  }
});
```

sw.js:

```
self.addEventListener("install", function (event) { event.waitUntil(preLoad());
});

self.addEventListener("fetch", function (event) {
event.respondWith(checkResponse(event.request).catch(function () {
console.log("Fetch from cache successful!") return returnFromCache(event.request);
}));
console.log("Fetch successful!") event.waitUntil(addToCache(event.request));
});

self.addEventListener('sync', event => { if (event.tag === 'syncMessage') {

console.log("Sync successful!")
}
});

self.addEventListener('push', function   (event)   { if (event && event.data) {
var data = event.data.json();
if (data.method == "pushMessage") { console.log("Push notification sent");
event.waitUntil(self.registration.showNotification("Omkar Sweets Corner", { body: data.message
}))
}
}
})



var filesToCache = [ '/',
'/menu', '/contactUs', '/offline.html',
];

var preLoad = function () {
return   caches.open("offline").then(function   (cache)   {
// caching index and important routes
return cache.addAll(filesToCache);
});
};

var checkResponse =   function   (request)   { return new Promise(function (fulfill, reject) {
fetch(request).then(function    (response)    { if (response.status !== 404) {
fulfill(response);
} else {
reject();
}
}, reject);
});
};

var addToCache = function (request) {
return caches.open("offline").then(function (cache) { return fetch(request).then(function
(response) {
return cache.put(request, response);
});
});
};

var returnFromCache = function (request) {
return  caches.open("offline").then(function   (cache)   { return
cache.match(request).then(function (matching) {
if (!matching || matching.status == 404) { return cache.match("offline.html");
} else {
return matching;
}
});
});
};
```

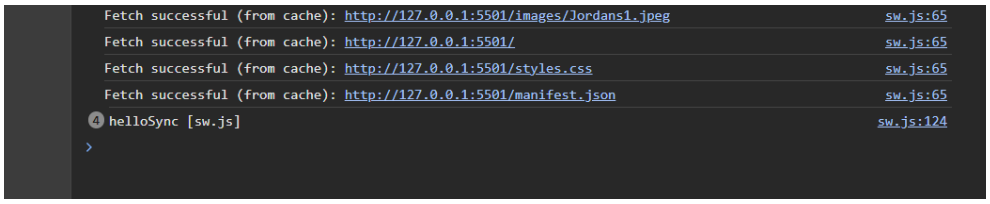
Output:

FetchEvent



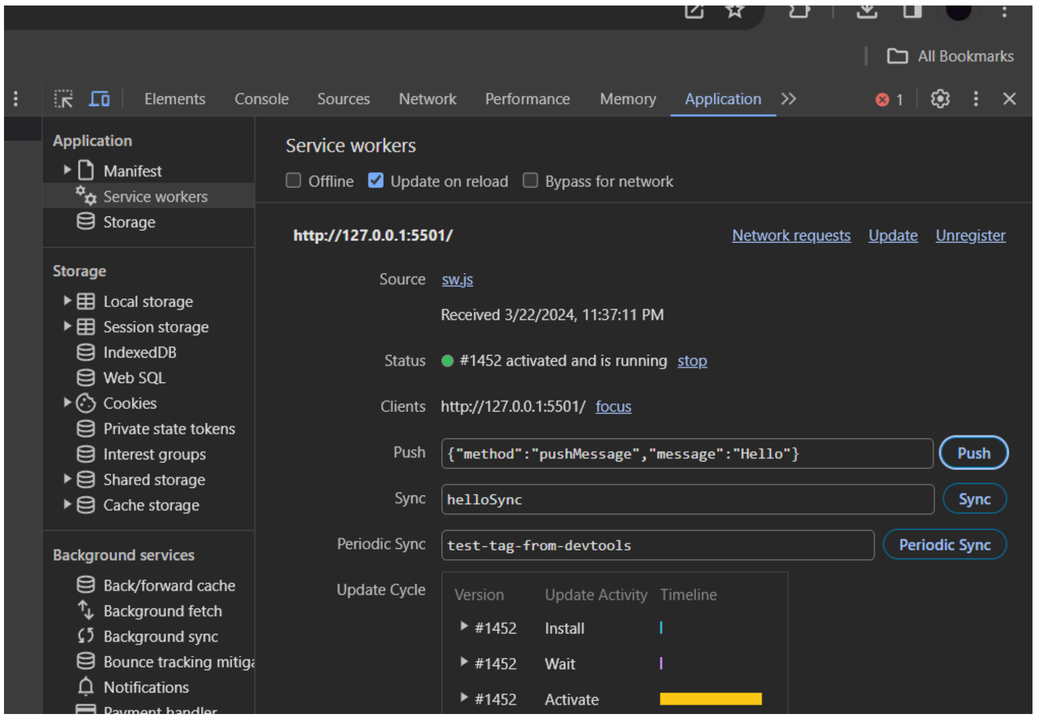
```
Fetch successful (from cache): http://127.0.0.1:5501/manifest.json                                    sw.js:65
Fetch successful (from cache): http://127.0.0.1:5501/images/Jordans1.jpeg                             sw.js:65
Fetch successful (from network, cached): http://127.0.0.1:5501/styles.css?_cacheOverride=17111300156 sw.js:92
77
Fetch successful (from cache): http://127.0.0.1:5501/styles.css?_cacheOverride=1711130015677          sw.js:65
⚠ The FetchEvent for "http://127.0.0.1:5501/styles.css?_cacheOverride=1711130015677" resulted in a network
error response: a Response whose "body" is locked cannot be used to respond to a request.
Fetch successful (from cache): http://127.0.0.1:5501/                                                 sw.js:65
Fetch successful (from cache): http://127.0.0.1:5501/script.js                                        sw.js:65
Fetch successful (from cache): http://127.0.0.1:5501/styles.css                                       sw.js:65
Live reload enabled.                                                                               (index):84
Service worker registration successful: http://127.0.0.1:5501/                                   script.js:94
Fetch successful (from cache): http://127.0.0.1:5501/images/Jordans1.jpeg                             sw.js:65
Fetch successful (from cache): http://127.0.0.1:5501/images/Jordans3.jpeg                             sw.js:65
Fetch successful (from cache): http://127.0.0.1:5501/images/Jordans2.jpeg                             sw.js:65
Fetch successful (from cache): http://127.0.0.1:5501/manifest.json                                    sw.js:65
Fetch successful (from cache): http://127.0.0.1:5501/images/Jordans1.jpeg                             sw.js:65
Fetch successful (from cache): http://127.0.0.1:5501/                                                 sw.js:65
Fetch successful (from cache): http://127.0.0.1:5501/styles.css                                       sw.js:65
Fetch successful (from cache): http://127.0.0.1:5501/manifest.json                                    sw.js:65
>
```

Sync event



```
Fetch successful (from cache): http://127.0.0.1:5501/images/Jordans1.jpeg       sw.js:65
Fetch successful (from cache): http://127.0.0.1:5501/                           sw.js:65
Fetch successful (from cache): http://127.0.0.1:5501/styles.css                 sw.js:65
Fetch successful (from cache): http://127.0.0.1:5501/manifest.json              sw.js:65
4 helloSync [sw.js]                                                             sw.js:124
>
```

Push event



Service workers

☐ Offline ☑ Update on reload ☐ Bypass for network

http://127.0.0.1:5501/

Source sw.js

Received 3/22/2024, 11:37:11 PM

Status ● #1452 activated and is running

Clients http://127.0.0.1:5501/

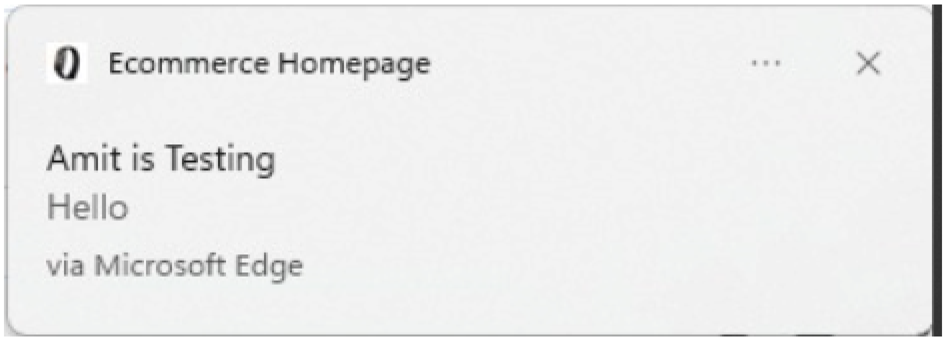
Push {"method":"pushMessage","message":"Hello"}

Sync helloSync

Periodic Sync test-tag-from-devtools

Update Cycle

Version	Update Activity	Timeline
#1452	Install	
#1452	Wait	
#1452	Activate	


Ecommerce Homepage

Amit is Testing

Hello

via Microsoft Edge

Conclusion : We have understood and successfully implemented events like fetch , push and sync for our ecommerce pwa.

Conclusion : I have understood and successfully registered a service worker, and completed the install and activation process for a new service worker for the E-commerce PWA.