

EXPERIMENT 9

Ronak Katariya 23

Prajwal Pandey 32

Snehal Patil 38

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.

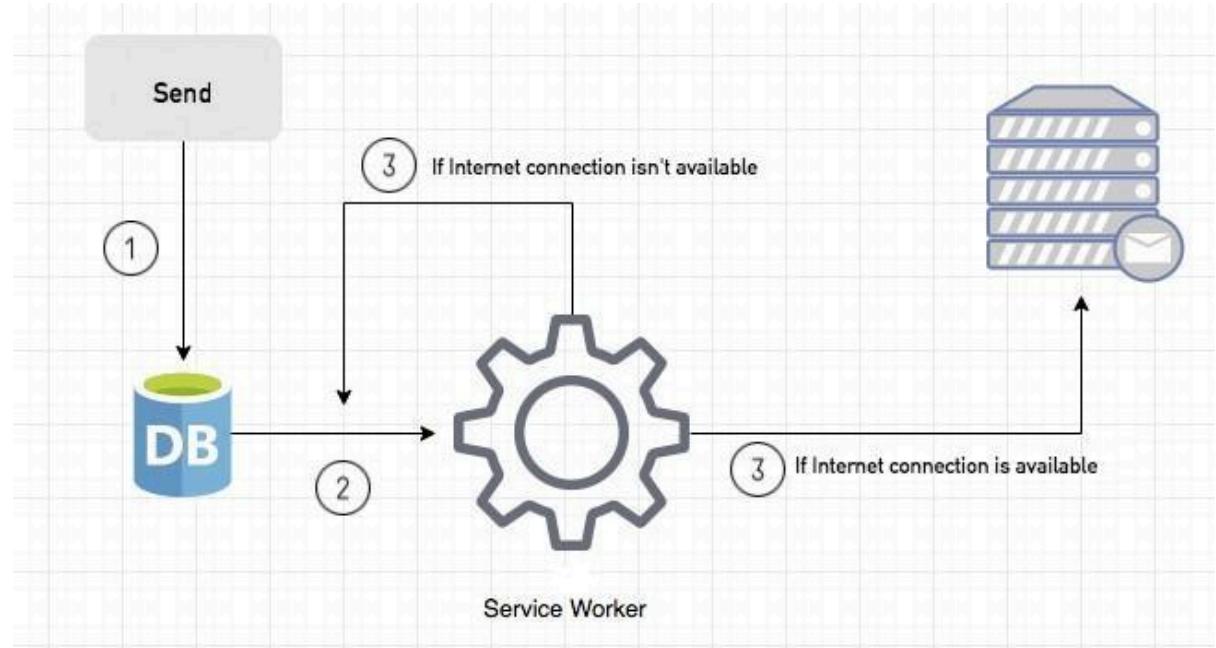
Sync Event

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.

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

```
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

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.

```
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
      }));
    }
  }
});
```

CODE:

```
Index.html
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8" />
<meta http-equiv="X-UA-Compatible" content="IE=edge" />
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
<title>Stopwatch</title>
<link rel="stylesheet" href="style.css">
<link rel="manifest" href="/manifest.json">
</head>
<body>
<div class="container">
<h1>Stopwatch</h1>
<p class="time">
<span id="minutes">00</span>:<span id="seconds">00</span>:<span id="tens">00</span>
</p>
<ul id="lapList"></ul>
<button id="start">Start</button>
<button id="stop">Stop</button>
<button id="reset">Reset</button>
<button id="lap">Lap</button>
</div>
<script src="stopwatch.js"></script>
<script>
if ('serviceWorker' in navigator) {
  window.addEventListener('load', () => {
    navigator.serviceWorker.register('/service-worker.js')
      .then(registration => {
        console.log('ServiceWorker registered with scope: ', registration.scope);
      })
      .catch(error => {
        console.log('ServiceWorker registration failed: ', error);
      });
  });
}

// Request Notification Permission and Show Popup
if ('Notification' in window) {
  Notification.requestPermission().then(permission => {
    if (permission === 'granted') {
      console.log('Notification permission granted');
      showNotification();
    }
  });
}
```

```

        }
    });
}

function showNotification() {
    if(Notification.permission === 'granted') {
        const options = {
            body: 'Welcome Snehal, Ronak, Prajjwal !',
            icon: '/images/icon.png',
            badge: '/images/icon.png'
        };

        // Display the notification
        new Notification('Welcome Snehal, Ronak, Prajjwal ', options);
    }
}
</script>
</body>
</html>

```

Manifest.json

```
{
    "name": "PWA",
    "short_name": "RPS",
    "description": "A simple and elegant stopwatch application.",
    "start_url": "/index.html",
    "display": "standalone",
    "background_color": "#ffffff",
    "theme_color": "#000000",
    "icons": [
        {
            "src": "/images/icon2.png",
            "sizes": "192x192",
            "type": "image/png"
        },
        {
            "src": "/images/icon.png",
            "sizes": "512x512",
            "type": "image/png"
        }
    ]
}
```

```
serviceworker.js
const CACHE_NAME = 'prodigy-stopwatch-cache-v1';
const urlsToCache = [
  '/',
  '/index.html',
  '/style.css',
  '/stopwatch.js',
  '/images/icon2.png',
  '/images/icon.png'
];
// Install Service Worker and Cache Files
self.addEventListener('install', event => {
  event.waitUntil(
    caches.open(CACHE_NAME)
      .then(cache => {
        console.log('Opened cache');
        return cache.addAll(urlsToCache);
      })
  );
});
// Cache and Return Requests
self.addEventListener('fetch', event => {
  event.respondWith(
    caches.match(event.request).then(response => {
      return response || fetch(event.request)
        .then(networkResponse => {
          if (!networkResponse || networkResponse.status !== 200) {
            return networkResponse;
          }
          return caches.open(CACHE_NAME).then(cache => {
            cache.put(event.request, networkResponse.clone());
            return networkResponse;
          });
        })
        .catch(() => {
          if (event.request.mode === 'navigate') {
            return caches.match('/index.html');
          }
        });
    })
  );
});
```

```
// Activate & Remove Old Caches
self.addEventListener('activate', event => {
  const cacheWhitelist = [CACHE_NAME];
  event.waitUntil(
    caches.keys().then(cacheNames => {
      return Promise.all(
        cacheNames.map(cacheName => {
          if (!cacheWhitelist.includes(cacheName)) {
            return caches.delete(cacheName);
          }
        })
      );
    })
  );
});
```

stopwatch.js

```
window.onload = function () {
  let minutes = 0;
  let seconds = 0;
  let tens = 0;
  let appendMinutes = document.querySelector('#minutes');
  let appendTens = document.querySelector('#tens');
  let appendSeconds = document.querySelector('#seconds');
  let startBtn = document.querySelector('#start');
  let stopBtn = document.querySelector('#stop');
  let resetBtn = document.querySelector('#reset');
  let lapBtn = document.querySelector('#lap');
  let lapList = document.querySelector('#lapList');
  let Interval;

  const startTimer = () => {
    tens++;
    if (tens <= 9) {
      appendTens.innerHTML = '0' + tens;
    }
    if (tens > 9) {
      appendTens.innerHTML = tens;
    }

    if (tens > 99) {
      seconds++;
      appendSeconds.innerHTML = '0' + seconds;
      tens = 0;
    }
  }

  startBtn.addEventListener('click', () => {
    Interval = setInterval(startTimer, 1000);
  });

  stopBtn.addEventListener('click', () => {
    clearInterval(Interval);
  });

  resetBtn.addEventListener('click', () => {
    minutes = 0;
    seconds = 0;
    tens = 0;
    appendMinutes.innerHTML = minutes;
    appendSeconds.innerHTML = '00';
    appendTens.innerHTML = '00';
  });

  lapBtn.addEventListener('click', () => {
    lapList.innerHTML += `

lap ${seconds + 1}: ${tens}.

`;
  });
};
```

```
appendTens.innerHTML = '0' + 0;
}

if (seconds > 9) {
    appendSeconds.innerHTML = seconds;
}

if (seconds > 59) {
    minutes++;
    appendMinutes.innerHTML = '0' + minutes;
    seconds = 0;
    appendSeconds.innerHTML = '0' + 0;
}
};

startBtn.onclick = () => {
    clearInterval(Interval);
    Interval = setInterval(startTimer, 10);
};

stopBtn.onclick = () => {
    clearInterval(Interval);
};

resetBtn.onclick = () => {
    clearInterval(Interval);
    tens = '00';
    seconds = '00';
    minutes = '00';
    appendTens.innerHTML = tens;
    appendSeconds.innerHTML = seconds;
    appendMinutes.innerHTML = minutes;
    lapList.innerHTML = "";
};

lapBtn.onclick = () => {
    var li = document.createElement('li');
    li.innerHTML = appendMinutes.innerHTML + ':' + appendSeconds.innerHTML + '.' +
appendTens.innerHTML;
    lapList.appendChild(li);
};

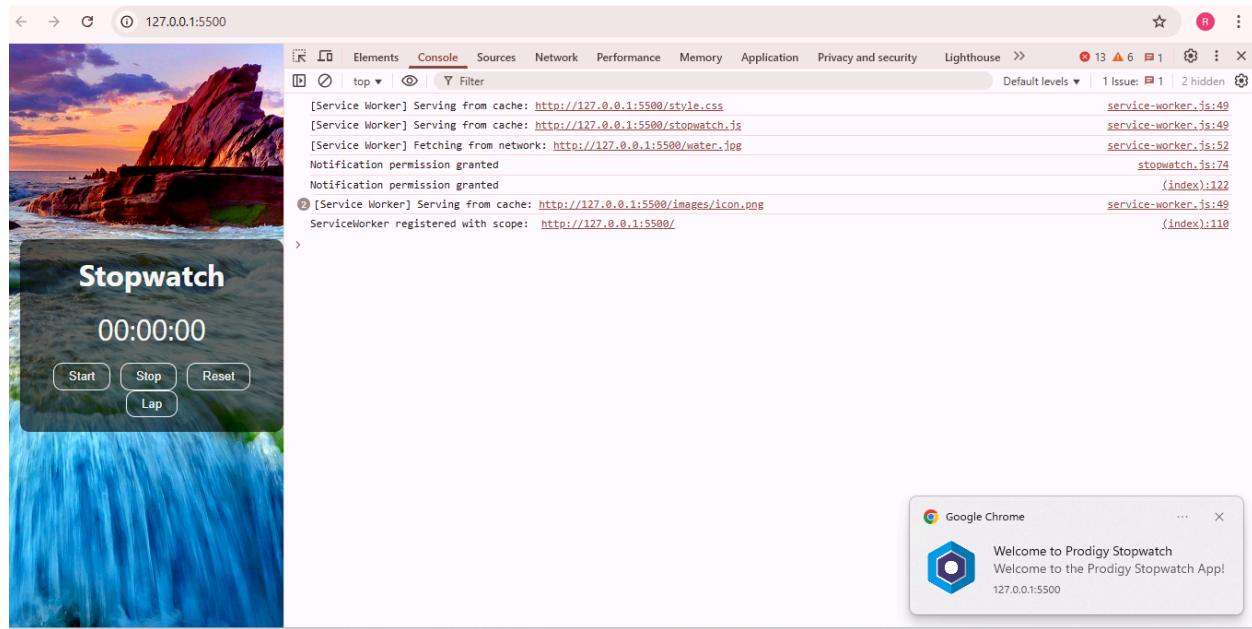
// Request Notification Permission when the app is loaded
```

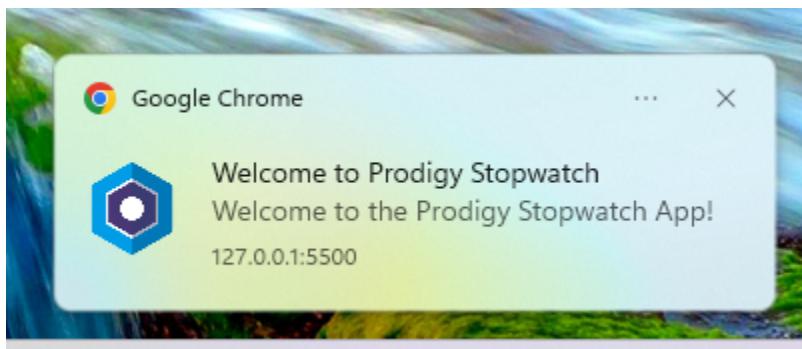
```

if ('Notification' in window && 'serviceWorker' in navigator) {
  Notification.requestPermission().then(permission => {
    if (permission === 'granted') {
      console.log('Notification permission granted');
    }
  });
}

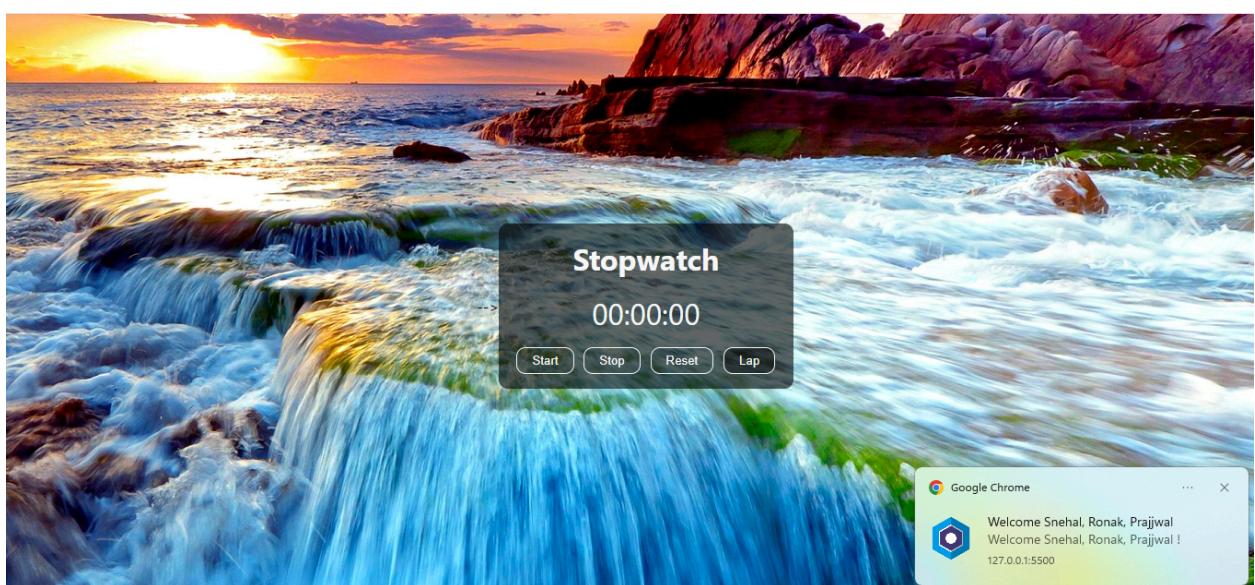
```

Screenshots:





```
Notification permission granted                                              stopwatch.js:74
Notification permission granted                                              (index):122
ServiceWorker registered with scope: http://127.0.0.1:5500/                  (index):110
```



iY_WD_02/index.html

The screenshot shows the Chrome DevTools Application tab for the URL `http://127.0.0.1:5500/`. The left sidebar lists various application components: Manifest, Service workers (which is selected), Storage, Background services, and others. The main panel displays information about the active service worker, including its source (`service-worker.js`), status (activated and running), clients (HTTP://127.0.0.1:5500/PRODIGY_WD_02/index.html), and various event handlers (Push, Sync, Periodic sync). It also shows the update cycle with three entries: Install, Wait, and Activate.

The screenshot shows the Chrome DevTools Console tab. The logs display several messages related to service workers and notifications:

- Live reload enabled.
- Notification permission granted
- Notification permission granted
- ServiceWorker registered with scope: `http://127.0.0.1:5500/`

On the right side, there are links to `index.html:68`, `stopwatch.js:74`, `index.html:122`, and `index.html:110`.