Name : Shivam.R.Prajapati **Div: D15C** **Roll No: 41**

Aim: To implement Service worker events like fetch, sync and push for E-commerce PWA

Theory:

Service Workers are a foundational technology of Progressive Web Apps (PWAs), providing capabilities such as offline support, background synchronization, and push notifications. These capabilities are made possible through a set of built-in events that the Service Worker listens to and responds to. The most commonly used functional events include fetch, sync, and push. Each serves a unique purpose in enhancing web app reliability, responsiveness, and user engagement.

fetch Event:

Definition:

The fetch event is triggered every time a web page controlled by a Service Worker initiates a network request. This includes requests for HTML pages, CSS files, images, scripts, or API calls.

Why It's Important:

The fetch event allows developers to intercept and control outgoing network requests. This control is critical for enabling offline support, intelligent caching, and fallback mechanisms.

Use Cases:

- Serve resources from cache to improve load speed.
- Provide fallback pages when offline.
- Customize responses for different request types.

Sample Code:

self.addEventListener('fetch', function(event) { event.respondWith(caches.match(event.request).then(function(response) {

// If a cached version of the request exists, return it.

```
// Otherwise, fetch it from the network.

return response || fetch(event.request);

})

);

});
```

Explanation:

- self.addEventListener('fetch', ...): Registers a listener for fetch events.
- event.respondWith(): Tells the browser to use a custom response for this request.
- caches.match(event.request): Searches the browser cache for a match to the current request.
- fetch(event.request): Makes a network request if no cached version is found.
- This approach is called the cache-first strategy, which prioritizes loading from cache and falls back to the network.

sync Event:

Definition:

The sync event is triggered when the browser regains network connectivity. It is used to execute deferred tasks (like sending data to the server) that couldn't complete due to being offline.

Why It's Important:

Many web apps require user actions (like form submissions) to be sent to the server. If the user is offline when this happens, the data can be temporarily stored and automatically synchronized once the device reconnects.

Use Cases:

- Retry failed form submissions or API calls.
- Sync data in the background without user interaction.
- Improve reliability of critical workflows.

Sample Code – Service Worker:

```
self.addEventListener('sync', function(event) { if (event.tag === 'sync-data') {
event.waitUntil(sendDataToServer());

}

});

function sendDataToServer() { return fetch('/submit-data', { method: 'POST',

body: JSON.stringify({ key: 'value' }), headers: {

'Content-Type': 'application/json'

}

});

}
```

Sample Code – Main JS Thread:

```
navigator.serviceWorker.ready.then(function(registration) { return
registration.sync.register('sync-data');

});
```

Explanation:

- The main script registers a sync event using registration.sync.register('sync-data').

- The Service Worker listens for the sync event with that tag.
- When triggered, the function sendDataToServer() is called to perform the actual data transfer.
- event.waitUntil() ensures the browser keeps the Service Worker alive until the task finishes.
- This ensures reliable data delivery, even if the user originally attempted it while offline.

push Event

Definition:

The push event is triggered when a server sends a push message to a client that has subscribed via the Push API. It enables the app to display notifications even when the web page is closed.

Why It's Important:

Push notifications allow web apps to maintain engagement with users by sending updates, alerts, or reminders — even when the app isn't currently open.

Use Cases:

- Send real-time updates or announcements.
- Notify users of background events (e.g., new messages).
- Deliver alerts while the app is closed or minimized.

Sample Code – Service Worker:

self.addEventListener('push', function(event) { const data = event.data ? event.data.json() : {};

const options = {

body: data.body || 'Default message body', icon: '/images/icon.png',

badge: '/images/badge.png'

```
};
event.waitUntil(
self.registration.showNotification(data.title || 'Default Title', options)
);
});
```

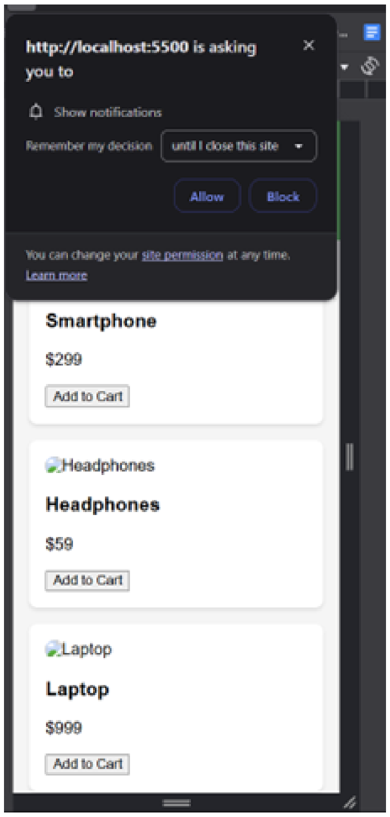
Explanation:

- event.data.json() parses the JSON payload received from the server.
- showNotification() uses the Notifications API to show the user a system-level notification.
- event.waitUntil() ensures the task completes before the Service Worker is terminated.
- This code ensures notifications can be received even if the app isn't currently open, which is a key feature of modern PWAs.

Code: https://github.com/Kingmaker-2/PWA-9.git

Output:

Name : Shivam.R.Prajapati **Div: D15C** **Roll No: 41**


http://localhost:5500 is asking you to

Show notifications

Remember my decision: until I close this site

You can change your site permission at any time. Learn more

Smartphone

$299

Headphones

Headphones

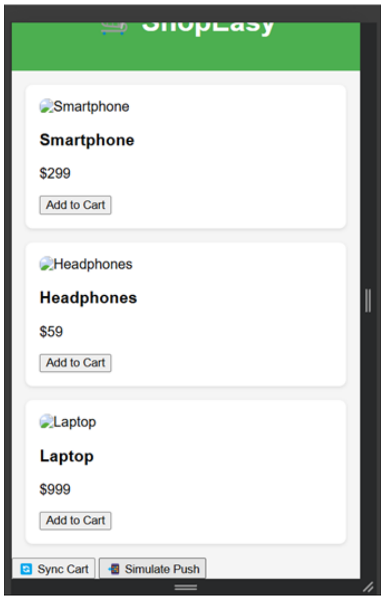
$59

Laptop

Laptop

$999

Request permission to send notifications


ShopEasy

Smartphone

Smartphone

$299

Headphones

Headphones

$59

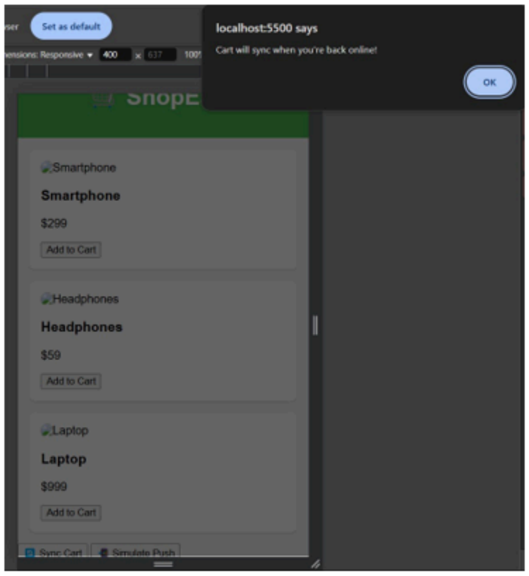
Laptop

Laptop

$999

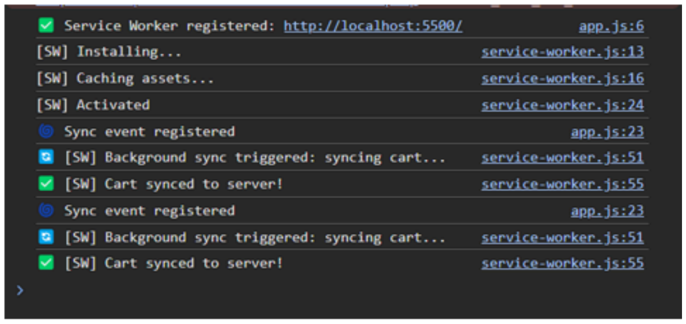
Push object received

Name : Shivam.R.Prajapati **Div: D15C** **Roll No: 41**



localhost:5500 says

Cart will sync when you're back online!



```
✅ Service Worker registered: http://localhost:5500/          app.js:6
[SW] Installing...                                 service-worker.js:13
[SW] Caching assets...                             service-worker.js:16
[SW] Activated                                     service-worker.js:24
🌀 Sync event registered                                     app.js:23
🔄 [SW] Background sync triggered: syncing cart... service-worker.js:51
✅ [SW] Cart synced to server!                     service-worker.js:55
🌀 Sync event registered                                     app.js:23
🔄 [SW] Background sync triggered: syncing cart... service-worker.js:51
✅ [SW] Cart synced to server!                     service-worker.js:55
>
```

Sync Notification

Conclusion: Service Worker events such as fetch, sync, and push play a vital role in enhancing the functionality and reliability of modern web applications. The fetch event empowers developers to intercept and manage network requests, enabling offline capabilities and performance optimizations through caching strategies. The sync event ensures that important tasks, such as data submission or background synchronization,

are reliably completed even after temporary network interruptions. Meanwhile, the push event enables real-time communication by allowing servers to deliver updates and notifications to users, regardless of whether the application is open. Together, these events form the backbone of Progressive Web Apps (PWAs), providing seamless experiences across various network conditions and device states, and ultimately contributing to improved user