Experiment 9

Aim: To implement Service worker events like fetch, sync and push for PWA.

Theory:
Service Workers operate as a programmable network proxy between web applications and external networks, executing in a separate thread from the main browser process. This isolation enables three critical capabilities:

- Network request interception and modification
- Persistent background processing
- Event-driven execution model

Event-Driven Paradigm
The Service Worker implements an observer pattern, responding to discrete lifecycle and functional events:

- Installation Phase: Initial setup and asset caching
- Activation Phase: Resource management and version control
- Runtime Events: Network interactions and background operations

Core Event Types

1. Fetch Event Mechanics
   The fetch event intercepts all network-bound requests originating from the controlled scope, enabling:
   a. Cache Resolution Strategies:
      i. Cache-first (offline priority)
      ii. Network-first (freshness priority)
      iii. Hybrid approaches (stale-while-revalidate)
   b. Request Transformation:
      i. URL rewriting
      ii. Header modification
      iii. Response substitution

2. Background Sync Implementation
   The sync event provides reliable execution guarantees through:
   a. Task Registration: Tag-based operation queuing
   b. Deferred Execution: Network availability detection
   c. Retry Semantics: Automatic failure recovery

3. Push Notification System
   Push events enable asynchronous messaging via:

a. Subscription Model: Cryptographic endpoint registration
b. Payload Handling: Data decryption and processing
c. Notification Display: Integration with Web Notifications API

Technical Constraints and Requirements

1. Security Model
    a. Mandatory HTTPS enforcement
    b. Scope-limited execution boundarie
    c. Explicit user consent requirements (notifications)

2. Resource Management
    a. Memory and storage quotas
    b. Background execution time limits
    c. Process prioritization rules

3. Lifecycle Considerations
    a. Versioned worker updates
    b. Graceful termination handling
    c. State persistence mechanisms

Code:

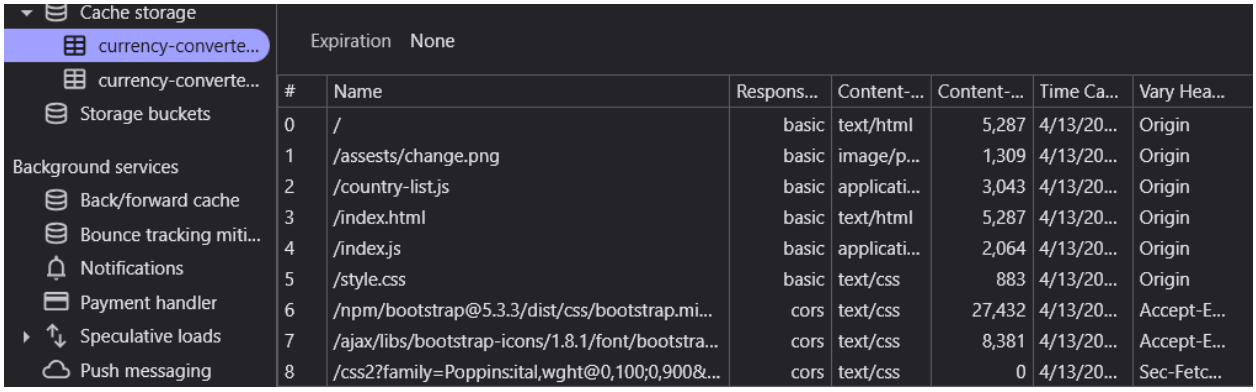
1. **Fetch Event (offline caching)**
   Cache first strategy for API calls

```
self.addEventListener('fetch', (event) => {
  // Skip non-GET requests (e.g., POST)
  if (event.request.method !== 'GET') return;

  event.respondWith(
    caches.match(event.request)
      .then((cachedResponse) => {
        // Cache hit: Return cached response
        if (cachedResponse) return cachedResponse;

        // Network fallback for API calls
        return fetch(event.request)
          .then((networkResponse) => {
            // Cache dynamic API responses (e.g., exchange rates)
            if (event.request.url.includes('/api/rates')) {
              const clone = networkResponse.clone();
              caches.open(CACHE_NAME)
                .then(cache => cache.put(event.request, clone));
            }
            return networkResponse;
```

```
        })
        .catch(() => {
          // Fallback for offline: Return cached placeholder
          return caches.match('/offline.html');
        });
    })
  );
});
```



#	Name	Respons...	Content-...	Content-...	Time Ca...	Vary Hea...
0	/	basic	text/html	5,287	4/13/20...	Origin
1	/assests/change.png	basic	image/p...	1,309	4/13/20...	Origin
2	/country-list.js	basic	applicati...	3,043	4/13/20...	Origin
3	/index.html	basic	text/html	5,287	4/13/20...	Origin
4	/index.js	basic	applicati...	2,064	4/13/20...	Origin
5	/style.css	basic	text/css	883	4/13/20...	Origin
6	/npm/bootstrap@5.3.3/dist/css/bootstrap.mi...	cors	text/css	27,432	4/13/20...	Accept-E...
7	/ajax/libs/bootstrap-icons/1.8.1/font/bootstra...	cors	text/css	8,381	4/13/20...	Accept-E...
8	/css2?family=Poppins:ital,wght@0,100;0,900&...	cors	text/css	0	4/13/20...	Sec-Fetc...

2. Background Sync

Retry failed updates:

```
function registerSync() {
  if ('sync' in navigator.serviceWorker) {
    navigator.serviceWorker.ready
      .then(registration => {
        registration.sync.register('sync-rates')
          .then(() => console.log('Sync registered'))
          .catch(err => console.error('Sync failed:', err));
      });
  }
}

// Example: Retry on button click
document.getElementById('retry-button').addEventListener('click', registerSync);
```

Handle sync update:

```
self.addEventListener('sync', (event) => {
  if (event.tag === 'sync-rates') {
    event.waitUntil(
      fetch('https://api.exchangerate-api.com/v4/latest/USD')
        .then(response => response.json())
        .then(data => {
          // Update UI or cache
```

```
        console.log('Rates synced:', data);
      })
    );
  }
});
```

3. Push Notifications

Request permissions:

```
function requestNotificationPermission() {
  Notification.requestPermission()
    .then(permission => {
      if (permission === 'granted') {
        console.log('Notification permission granted');
      }
    });
}

// Call on app startup
requestNotificationPermission();
```

Handle Push events

```
self.addEventListener('push', (event) => {
  const data = event.data.json();
  const title = 'Currency Update';
  const options = {
    body: `1 USD = ${data.rates.INR} INR`,
    icon: '/icons/icon-192x192.png',
    badge: '/icons/icon-512x512.png'
  };

  event.waitUntil(
    self.registration.showNotification(title, options)
  );
});
```

Github link: https://github.com/Rakshit5467/Currency-Converter.git

Conclusion:

This experiment effectively demonstrated the implementation of key Service Worker events (fetch, sync, and push) to transform a standard web application into a fully-featured Progressive Web App. By leveraging the fetch event for intelligent caching strategies, we achieved offline functionality and improved performance. The background sync implementation enabled reliable data synchronization even in unstable network conditions, while push notifications enhanced user engagement through timely updates. These implementations collectively showcase how modern web technologies can bridge

the gap between web and native app experiences, providing users with app-like reliability, performance, and functionality while maintaining the accessibility and reach of web applications.