Name: Atif Ansari Roll no: 04 Class: D15B

MPL Assignment - 2

05/05

1] Define Progressive Web App (PWA) and explain its significance in modern web development. Discuss the key characteristics that differentiate PWAs from traditional mobile apps.

→ Definition of Progressive Web App (PWA):

A Progressive Web App (PWA) is a type of web application that combines the best features of web and mobile apps. PWAs offer a native app-like experience using modern web technologies while remaining accessible via a web browser.

Significance of PWAs in Modern Web Development:

- Cross-Platform Compatibility: Works on any device with a modern web browser.
- Improved Performance: Uses caching and ~~backgrouded~~ background sync for fast loading.
- Offline Functionality: Service Workers allow access to content even when offline.
- Better User Engagement: Supports push notifications and home screen installation.
- Cost-Effective: Reduces the need for seperate development for Android and iOS.

Key Characteristics that Differentiate PWAs from Traditional Mobile Apps:

Feature	PWA	Traditional Mo App
Installation	No need to download from an app store; can be added to home screen	Requires installati via Google Play St or App Store
Offline Support	Works offline using Service Workers	Requires explicit offline functionalit
Performance	fast loading using caching techniques	Depends on native platform optimizatio
Updates	Automatic updates through the web	Requires manual updates from stor
Device Access	Limited access to device features	Full access to native device featur
Platform Dependency	Runs on any browser across platforms	Seperate development needed for Android and iOS.

Q.2] Define responsive web design and explain its importance in the context of Progressive Web Apps. Compare and contrast responsive, fluid, and adaptive web design approache

Definition of Responsive Web Design (RWD):

Responsive Web Design (RWD) is a web development approach that ensures a website adapts to different screen sizes and devices using flexible layouts, CSS media queries, and scalable images.

Importance of RWD in PWAs:

- Ensures PWAs look good on all devices, from desktops to mobile phones.
- Enhances user experience by providing seamless interaction across screen sizes.
- Improves SEO rankings, as search engines favor mobile-friendly sites.
- Reduces development effort, as a single codebase works across all devices.

Comparison of Responsive, Fluid, and Adaptive Web Design:

Approach	Definition	Key features	Best use case
Responsive	Uses media queries to adjust layout based on screen size	flexible grids, scalable images, CSS breakpoints	PWAs, modern websites needing cross-device support
Fluid	Uses percentage-based widths instead of fixed units	Adapts proportionally to screen size without breakpoints	Simple layouts that need smooth resizing

Adaptive	Uses predefined layouts for different screen sizes.	Detects device type and loads a specific design	Websites requiring precise control for different de

Q.3) Describe the lifecycle of Service Workers, includi registration, installation, and activation phases

→ A Service Worker is a JavaScript script that runs in the background, enabling features like offline support, caching, and push notifications in PWAs.

Service Worker Lifecycle:

1. Registration : The service worker is register in the JavaScript file, allowing the brow to recognize it.

- Example :

```
if ('serviceWorker' in navigator) {
  navigator.serviceWorker.register('/sw.js').the
   reg => console.log('Service Worker registered',
    .catch (err =>
     console.log('service Worker registration
     failed', err));
}
```

2. Installation : The service worker is downloaded and installed in the background. Static ass can be pre-cached for offline use.

- Example:

```
self. addEventListener ('install',
event => {
    event. waitUntil (
caches. open ('pwa-cache-v1'). then (
 cache => {
          return  cache. addAll ([
             '/', '/index.html', '/styles.css',
              '/script.js'
          ]);
         })
        );
       });
```

3. Activation: The service worker is activated and starts controlling pages. Old caches may be deleted to manage storage efficiently.

- Example:

```
~~E~~   self. addEventListener ('activate', ~~ea~~
      event => {
          event. waitUntil (
              caches. key (). then (keys => {
                 return  Promise. all (
                      keys. filter( key => key !
          == 'pwa- cache - v1') . map (key =>
          caches. delete (key))
          );
        })
       );
      });
```

4. Fetch Event Handling (optional): The Ser worker intercepts network requests and serve cached responces when offline.

- Example :

```
  self.addEventListener('fetch',
  event => {
      event.respondWith(
  catches.match(event.request).then(
  response => {
          return response ||
   fetch(event.request);
    })
   );
  });
```

Q.4) Explain the use of IndexedDB in the Servi Worker for Data storage.

→ IndexedDB is a low-level NoSQL databas built into modern browsers. It allows we applications, including PWAs, to store structured data efficiently for offline IndexedDB in Service Workers is used because of features like:

- Persistent Storage: Unlike localStorage, IndexedDB can storage large amounts of data.
- Asynchronous Operations: Uses Promises to handle data efficiently without blocking the main thread.

- **Indexed Storage**: Allows fast querying using keys and indexes.

Using IndexedDB in Service Workers:

1. **Opening a Database:**

```
let db;
const request = indexedDB.open('PWA_Database', 1);
request.onsuccess = event => {
  db = event.target.result;
};
request.onupgradeneeded = event => {
let db = event.target.result;
db.createObjectStore ('users',
{ key Path : 'id' });
};
```

2. **Adding Data to IndexedDB:**

```
function addUser (user) {
    let transaction = db.transaction (['user'],
    'readwrite');
    let store = transaction.objectStore ('users');
    store.add (user);
}
addUser ({ id: 1, name: 'Atif',
  email : 'atif123@gmail.com' });
```

3. **Retrieving Data from IndexedDB:**

```
function getUser (id) {
    let transaction = db.transaction ([ 'users' ],
    'readonly');
let store = transaction.ObjectStore ('users');
 let request = store.get (id);
 request.onsuccess = () => {
     console.log (request.result);
   };
}
 getUser (1);
```

Data Synchronization in Service Workers:
When offline, data is stored in IndexedDB
When online, a background sync process sends stored data to the server.

- Example:

```
Self.addEventListener ('sync', event => {
  if (event.tag === 'sync-data') {
        event.waitUntil (getUser (1).then(user=
        return fetch ('/sync', {
            method : 'POST',
            body : JSON.stringify (user),
            headers : {'Content-Type' :
            'application/json'}
          });
        })
      );
   }
});
```

IndexedDB in Service Workers enables efficient offline data storage and synchronization, ensuri seamless functionality and data persistence.