RV College of Engineering®
Mysore Road, RV Vidyaniketan Post,
Bengaluru - 560059, Karnataka, India

Department of Master of Computer Applications (MCA)

Mobile Application Development (MCA221IA)

Hand Notes

Unit - 2	**Topic: Fetch**
Editor: 1RV24MC086 – Rohit Koujalagi	

List of Questions

Question 1: What is Fetch API and its basic syntax (02 Marks)
Question 2: Write the syntax to make a basic GET request using the Fetch API (02 Marks)
Question 3: Fetch API with different HTTP methods GET and POST (02 Marks)
Question 4: How does Fetch API handle errors? (02 Marks)

Question 5: Explain Fetch API with async/await and Promises (04 Marks)
Question 6: Fetch API in Service Workers context (04 Marks)
Question 7: Explain the role of the Fetch API in enabling offline access in PWAs (04 Marks)

Question 8: How does the Fetch API support asynchronous data retrieval in service workers? (06 Marks)
Question 9: What is the significance of .then() and .catch() in a Fetch API call? (06 Marks)

Question 10: Explain how the Fetch API is used in a service worker to intercept network requests and cache them for offline usage. Include code and example.(08 Marks)

Question 11: Write and explain a code snippet that fetches JSON data using the Fetch API.(10 Marks)
Question 12: Design a flow where the Fetch API handles API data, caches it dynamically, and serves the cached version when the device is offline. Provide appropriate code and explanation.(10 Marks)

02 Marks Questions:

Q1: What is Fetch API and its basic syntax

Answer: The Fetch API is a modern web API that provides a more powerful and flexible way to make HTTP requests compared to XMLHttpRequest. It returns Promises and has a cleaner, more intuitive syntax for handling asynchronous operations.
Code:

```
// Basic Fetch syntax
fetch('https://api.example.com/data')
  .then(response => response.json())
  .then(data => console.log(data))
  .catch(error => console.error('Error:', error));
```

Q2: Write the syntax to make a basic GET request using the Fetch API

Answer:
The fetch() function is used to make network requests. A basic GET request syntax using the Fetch API is:
Code :

```
fetch('https://api.example.com/data')
  .then(response => response.json())
  .then(data => console.log(data))
  .catch(error => console.error('Error:', error));
```

Q3: How does Fetch API handle different HTTP methods GET and POST? (02 Marks)

Answer:
Fetch API supports all HTTP methods through the method option in the request configuration. Common methods include GET (default), POST, PUT, DELETE, PATCH, etc.
Code:

```
// GET request (default)
fetch('/api/users');

// POST request
fetch('/api/users', {
  method: 'POST',
  headers: {'Content-Type': 'application/json'},
  body: JSON.stringify({name: 'John', age: 30})
});
```

Q4: How does Fetch API handle errors? (02 Marks)

Answer:
Fetch API only rejects promises for network errors or if the request couldn't be completed. HTTP error status codes (404, 500, etc.) don't automatically reject the promise. You need to check response.ok or response.status manually.

Code:

```
fetch('/api/data')
  .then(response => {
    if (!response.ok) {
      throw new Error(`HTTP error! status: ${response.status}`);
```

```
    }
    return response.json();
  })
  .catch(error => console.error('Fetch error:', error));
```

04 Marks Questions:

Q5: How does Fetch API work with async/await and Promises? (04 Marks)

Answer:
Fetch API naturally integrates with both Promise chains and async/await syntax. Async/await provides a more synchronous-looking code structure while maintaining asynchronous behavior, making error handling and code flow more readable.

Code:

```
// Using Promises
function fetchWithPromises() {
  return fetch('/api/users')
    .then(response => {
      if (!response.ok) throw new Error('Network error');
      return response.json();
    })
    .then(users => {
      console.log('Users:', users);
      return users;
    })
    .catch(error => console.error('Error:', error));
}

// Using async/await
async function fetchWithAsync() {
  try {
    const response = await fetch('/api/users');
    if (!response.ok) throw new Error('Network error');
    const users = await response.json();
    console.log('Users:', users);
    return users;
  } catch (error) {
    console.error('Error:', error);
  }
}
```

Q6: Explain Fetch API usage within Service Workers context. (04 Marks)

Answer:
In Service Workers, Fetch API is used to intercept network requests, implement caching strategies, and provide offline functionality. Service Workers can modify requests, serve cached responses, or fetch from network based on custom logic.

Code:

```
// In service worker
self.addEventListener('fetch', (event) => {
  event.respondWith(
    caches.match(event.request)
```

```
    .then(response => {
      // Return cached version or fetch from network
      return response || fetch(event.request)
        .then(fetchResponse => {
          // Clone and cache the response
          const responseClone = fetchResponse.clone();
          caches.open('my-cache').then(cache => {
            cache.put(event.request, responseClone);
          });
          return fetchResponse;
        });
    })
  );
});
```

Q7: Explain the role of the Fetch API in enabling offline access in PWAs (04 Marks)

Answer:
The **Workspace API** plays a crucial role in enabling offline access in Progressive Web Apps (PWAs) because it allows Service Workers to intercept and manage network requests. When a PWA attempts to fetch a resource (like an image, CSS file, or data), the Service Worker can intercede. Instead of directly going to the network, the Service Worker can use the Workspace API to:

Serve from Cache: Check if the requested resource is already stored in its cache using the Cache API. If it is, the Service Worker can directly serve the cached version, providing instant access even if the device is offline.
Cache Network Responses: If the resource is not in the cache, the Service Worker can proceed to fetch it from the network. Upon a successful network response, the Service Worker can then use the Cache API to store a copy of that response for future offline use.

Code:

```
// service-worker.js
self.addEventListener('fetch', event => {
  event.respondWith(
    caches.match(event.request) // Try to find the request in the cache
      .then(response => {
        if (response) {
          return response; // If found, return the cached response
        }
        // If not found in cache, fetch from the network
        return fetch(event.request);
      })
  );
});
```

06 Marks Questions:

Q8: How does the Fetch API support asynchronous data retrieval in service workers?

Answer:

The Fetch API supports asynchronous data retrieval by returning Promises, allowing service workers to handle network requests without blocking the main thread. When a service worker intercepts a fetch event, it can:

- Asynchronously fetch data from the network, or
- Respond with cached data, or
- Combine both for caching strategies like stale-while-revalidate.
- This asynchronous behaviour ensures that service workers can:
- Wait for a network response.
- Serve cached content immediately if available.

Use async/await or .then() chaining for handling responses and errors gracefully.

Code:

```
self.addEventListener('fetch', event => {

  event.respondWith(

    (async () => {

      const cachedResponse = await caches.match(event.request);

      if (cachedResponse) {

        return cachedResponse;

      }

      const networkResponse = await fetch(event.request);

      return networkResponse;

    })()

  );

});
```

Q9: What is the significance of .then() and .catch() in a Fetch API call?

Answer:
.then() and .catch() are methods used to handle **Promises**, which the Fetch API returns. They allow developers to manage asynchronous operations like handling data once it is received or dealing with errors gracefully.

.then() – Handling the Fulfilled State

- It is used to process the response returned by the Fetch API after a successful network call.
- It can be chained multiple times to perform sequential actions.
- Typically used to convert the response to JSON, text, or blob.

.catch() – Handling Errors

- It is used to catch and handle any errors that occur during the fetch operation or in the .then() chain.
- Useful for logging errors or providing fallback behavior.

Code:

```
fetch('https://api.example.com/data')
  .then(response => {
    if (!response.ok) {
      throw new Error('Network response was not ok');
    }
    return response.json();
  })
  .then(data => console.log('Data:', data))
  .catch(error => console.error('Fetch Error:', error));
```

08 Marks Questions:

Q10: Explain how the Fetch API is used in a service worker to intercept network requests and cache them for offline usage. Include code and example.

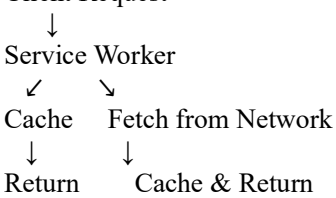
Answer:
The **Fetch API** in combination with **Service Workers** allows developers to intercept network requests and serve responses from the **Cache API**, enabling offline functionality in Progressive Web Apps (PWAs).
A service worker listens for the fetch event and uses the Fetch API to:

1. Intercept the request.
2. Check if the resource exists in the cache.
3. If it exists, return the cached response.
4. If not, fetch from the network and optionally cache it for future use.

Code:

```
self.addEventListener('fetch', event => {
  event.respondWith(
    caches.match(event.request).then(cachedResponse => {
      if (cachedResponse) {
        return cachedResponse; // Serve from cache
      }
      return fetch(event.request).then(networkResponse => {
        return caches.open('dynamic-cache').then(cache => {
          cache.put(event.request, networkResponse.clone()); // Cache new response
          return networkResponse; // Serve network response
        });
      });
    }).catch(error => {
      console.error('Fetch failed:', error);
    })
  );
});
```

Diagram :
Request Flow:



10 Marks Questions:

Q11: Write and explain a code snippet that fetches JSON data using the Fetch API.

Answer:
The Fetch API is used to make asynchronous HTTP requests in web applications. It returns a Promise and is commonly used to retrieve JSON data from web servers or APIs. Below is a complete example of how to use it to fetch JSON data.

Code:

```
// Fetch JSON data from a sample API
fetch('https://jsonplaceholder.typicode.com/posts/1')
  .then(response => {
    if (!response.ok) {
      throw new Error('Network response was not ok');
    }
    return response.json(); // Parse the response body as JSON
  })
  .then(data => {
    console.log('Fetched JSON Data:', data);
    // Example: Display title
    document.getElementById('title').textContent = data.title;
  })
  .catch(error => {
    console.error('Fetch error:', error);
  });
```

Explanation:

- fetch('URL')
  Initiates a GET request to the given API endpoint.
- .then(response => response.json())
  Converts the HTTP response into a JavaScript object using .json().
- if (!response.ok)
  Checks for any HTTP errors like 404 or 500 and throws an error if present.
- .then(data => { ... })
  Handles the actual JSON data and performs operations like displaying it.
- .catch(error => { ... })
  Catches and handles any network errors or problems in the fetch chain.

Q12: Design a flow where the Fetch API handles API data, caches it dynamically, and serves the cached version when the device is offline. Provide appropriate code and explanation.

Answer:
This flow demonstrates how to use the **Fetch API** along with a **Service Worker** and the **Cache API** to:

- Fetch data from an API
- Dynamically cache it
- Serve the cached version when offline

Flow Description:

1. The user requests data (e.g., from an API).
2. The service worker intercepts the request using fetch event.
3. It first checks the cache for a response using caches.match().
4. If found, it serves the cached response.
5. If not found, it uses the Fetch API to get fresh data from the network.
6. The response is cloned and stored in a dynamic cache.
7. If offline, the service worker directly serves the cached response if available.

Code:

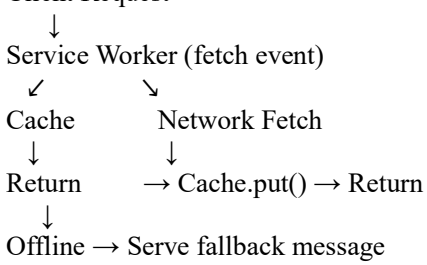
```
self.addEventListener('fetch', event => {
  event.respondWith(
    caches.match(event.request)
      .then(cachedResponse => {
        // Serve from cache if available
        if (cachedResponse) {
          return cachedResponse;
        }

        // Otherwise fetch from network and cache it
        return fetch(event.request).then(networkResponse => {
          return caches.open('dynamic-api-cache').then(cache => {
            cache.put(event.request, networkResponse.clone()); // Store in cache
            return networkResponse; // Serve the response
          });
        });
      })
      .catch(error => {
        // Fallback when both network and cache fail
        console.error('Offline and no cache found', error);
        return new Response("You're offline and the requested resource is not cached.");
      })
  );
});
```

Explanation:

- caches.match() checks if the requested API response is already in cache.
- fetch(event.request) fetches fresh data from the network.
- cache.put() stores a clone of the network response in the cache for future offline use.
- A fallback Response is served if both cache and network fail.

Diagram



Caution: These question and answers are just for the reference and are not the BIBLE of Mobile Application Development feel free to explore.