Assignment Submission: Fetch API – Blog Posts & Todo List

1. Approach

The objective of this assignment was to gain practical experience with the **Fetch API**, **JSON handling**, **DOM manipulation**, **error handling**, and **JavaScript design patterns**.

To achieve this, the following approach was adopted:

1. **Understand the APIs:**
    - Familiarized with **JSONPlaceholder API** endpoints:
        - /posts → for blog posts
        - /todos → for todo items
2. **Setup HTML structure:**
    - Created sections for **Posts** and **Todos**.
    - Added containers to dynamically display content and error messages.
3. **Fetch data using Fetch API:**
    - Implemented fetchPosts() and fetchTodos() functions with **async/await**.
    - Handled JSON responses and converted them to JavaScript objects.
4. **DOM Manipulation:**
    - Dynamically created cards for posts and todos.
    - Displayed relevant data like post title/body and todo title/status.
5. **Error Handling:**
    - Added try...catch blocks to handle network or server errors.
    - Checked response.ok to detect HTTP errors.
    - Displayed user-friendly error messages on the page.
6. **JavaScript Design Pattern:**
    - Used **Revealing Module Pattern** to organize code into a single App module.
    - Encapsulated all internal functions and exposed only an init() function.

7. **Integration and Testing:**
    - On DOMContentLoaded, initialized the app using App.init().
    - Tested functionality with normal internet connection and offline mode to verify error handling.

2. Code Implementation

HTML (index.html)

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Blog & Todo Dashboard</title>
  <style>
    body { font-family: Arial, sans-serif; margin: 20px; }
    h1 { color: #2c3e50; }
    .container { display: flex; gap: 50px; }
    .section { width: 50%; }
    .card { background: #ecf0f1; padding: 10px; margin-bottom: 10px; border-radius: 5px; }
    .error { color: red; font-weight: bold; }
  </style>
</head>
<body>
  <h1>Blog Posts & Todo List</h1>
  <div class="container">
    <div class="section" id="posts-section">
      <h2>Posts</h2>
```

```
            <div id="posts-container"></div>
            <div id="posts-error" class="error"></div>
        </div>
        <div class="section" id="todos-section">
            <h2>Todos</h2>
            <div id="todos-container"></div>
            <div id="todos-error" class="error"></div>
        </div>
    </div>
    <script src="app.js"></script>
</body>
</html>
```

JavaScript (app.js)

```
// App module using Revealing Module Pattern
const App = (function() {
    // API endpoints
    const postsUrl = 'https://jsonplaceholder.typicode.com/posts';
    const todosUrl = 'https://jsonplaceholder.typicode.com/todos';

    // DOM elements
    const postsContainer = document.getElementById('posts-container');
    const todosContainer = document.getElementById('todos-container');
    const postsError = document.getElementById('posts-error');
    const todosError = document.getElementById('todos-error');

    // Fetch posts from API
```

```
async function fetchPosts() {
    try {
        const response = await fetch(postsUrl);
        if (!response.ok) throw new Error(`Error: ${response.status}`);
        const data = await response.json();
        displayPosts(data);
    } catch (error) {
        postsError.textContent = `Failed to load posts. ${error.message}`;
    }
}

// Display posts in DOM
function displayPosts(posts) {
    postsContainer.innerHTML = '';
    posts.slice(0, 10).forEach(post => {
        const postDiv = document.createElement('div');
        postDiv.classList.add('card');
        postDiv.innerHTML = `<h3>${post.title}</h3><p>${post.body}</p>`;
        postsContainer.appendChild(postDiv);
    });
}

// Fetch todos from API
async function fetchTodos() {
    try {
        const response = await fetch(todosUrl);
        if (!response.ok) throw new Error(`Error: ${response.status}`);
```

```
            const data = await response.json();
            displayTodos(data);
        } catch (error) {
            todosError.textContent = `Failed to load todos. ${error.message}`;
        }
    }

    // Display todos in DOM
    function displayTodos(todos) {
        todosContainer.innerHTML = '';
        todos.slice(0, 10).forEach(todo => {
            const todoDiv = document.createElement('div');
            todoDiv.classList.add('card');
            todoDiv.innerHTML = `<strong>${todo.title}</strong> - ${todo.completed ? '✅ Completed' : '❌ Pending'}`;
            todosContainer.appendChild(todoDiv);
        });
    }

    // Initialize the app
    function init() {
        fetchPosts();
        fetchTodos();
    }

    return {
        init
```

```
    };
})();

// Start the app when DOM is loaded
document.addEventListener('DOMContentLoaded', App.init);
```

3. Challenges Faced

1. **Understanding API response structure**
    - Needed to check the JSON response format to know which fields to display (title, body, completed).
2. **Error handling**
    - Initially only network errors were caught; later added response.ok checks for HTTP status errors.
3. **DOM updates**
    - Ensuring old content is cleared before rendering new data (innerHTML='') to avoid duplication.
4. **Module Pattern**
    - Learning to encapsulate functions properly and expose only what is needed (init()).

4. Reflection

- Learned how to **fetch data from remote APIs** using fetch() and async/await.
- Practiced **dynamic DOM manipulation** by creating elements on the fly.
- Implemented **robust error handling** to enhance user experience.
- Gained experience using the **Revealing Module Pattern** for scalable and maintainable JavaScript code.
- Understood the importance of **clean code** and modular design in real-world web applications.

This assignment strengthened my understanding of **front-end API integration** and **modern JavaScript patterns**, which are essential skills for web development and full-stack applications.

5. Result: