**JavaScript Assignment: Advanced Asynchronous Programming**

**Instructions:**

This assignment is designed to test your understanding of asynchronous programming concepts, including **callbacks**, **promises**, **async/await**, and **fetch()**. Complete the tasks in the given order as they progressively increase in difficulty. Make sure to handle errors appropriately wherever required.

**Section 1: Callbacks and Promises**

1. **Callback Basics**  
   Write a function processNumber(num, callback) that takes a number and a callback function.
   * If num is even, the callback should log "Even number!"
   * If num is odd, the callback should log "Odd number!"
2. **Promise Creation**  
   Create a promise-based function isPrime(num) that resolves if a number is prime and rejects otherwise. Test it with various inputs.
3. **Chaining Promises**  
   Write a function doubleNumber(num) that returns a promise resolving the double of the given number.  
   Chain this promise to double the number 4 times and log the result.
4. **Promise.all Example**  
   Create three promises:
   * One resolves to "Resolved after 1 second" after 1 second.
   * Another resolves to "Resolved after 2 seconds" after 2 seconds.
   * The third rejects with "Rejected after 3 seconds" after 3 seconds.  
     Use Promise.allSettled() to log the status of all promises.

**Section 2: Async/Await Basics**

1. **Basic Async/Await**  
   Create an async function waitAndLog() that:
   * Waits for 2 seconds (using setTimeout wrapped in a promise).
   * Logs "Done waiting!" after 2 seconds.
2. **Chaining Async Functions**  
   Write two asynchronous functions:
   * fetchUser(id) simulates fetching a user by ID and returns a user object.
   * fetchPosts(userId) simulates fetching posts of a user.  
     Chain these functions using async/await to log the posts of a user.
3. **Error Handling in Async/Await**  
   Create a function fetchData(url) that fetches JSON data from a given URL using fetch(). Handle the following cases:
   * The URL is invalid.
   * The response is not OK (status >= 400).

**Section 3: Advanced Fetch and Async Concepts**

1. **Fetching and Rendering Data**  
   Fetch user data from the JSONPlaceholder API and render the following information in a table:
   * Name
   * Email
   * Address (city and street)
2. **Sequential Fetch Requests**  
   Using async/await, fetch data sequentially from the following endpoints:
   * /users (fetch user data)
   * /posts (fetch posts of users).  
     Ensure the posts belong to the users you fetched.
3. **Fetching with Pagination**  
   Fetch paginated data from the JSONPlaceholder API.
   * Create a function to fetch the next page of data.
   * Simulate clicking "Next" to fetch and log the next set of posts.

**Section 4: Combining Concepts**

1. **Retry Logic with Promises**  
   Write a function fetchWithRetry(url, retries) that:
   * Tries to fetch data from a URL.
   * Retries the fetch up to retries times if it fails.
   * Resolves with the fetched data or rejects after all retries fail.
2. **Concurrency Control**  
   Create a function fetchConcurrently(urls) that:
   * Takes an array of URLs.
   * Fetches data from each URL concurrently but limits the number of concurrent fetches to 3.
3. **File Upload Simulation**  
   Simulate a file upload function uploadFile(file) that:
   * Takes 1–5 seconds to "upload" based on the file size.
   * Returns a promise resolving with "File uploaded" if successful.
   * Rejects with "File upload failed" if the upload time exceeds 4 seconds.  
     Use this function to simulate uploading 5 files concurrently and log the results.

**Section 5: Real-World Scenarios**

1. **Dependent Data Fetching**  
   Fetch posts and comments from the JSONPlaceholder API.
   * Fetch a specific post by ID.
   * Then fetch comments for that post.  
     Display the post along with its comments in the console.
2. **Polling with Fetch**  
   Create a function pollStatus(url) that:
   * Fetches a status from a given URL every 3 seconds.
   * Stops polling when the status is "Complete".  
     Use a mock API or simulate the behavior using a promise.
3. **Dynamic API Calls**  
   Given an array of user IDs, fetch user details for each ID in parallel. Then:
   * Fetch posts for each user sequentially.
   * Log user details and their posts together.

**Section 6: Fun Projects**

1. **Quiz App**  
   Build a quiz app that:
   * Fetches questions from an API.
   * Displays one question at a time.
   * Waits for the user to answer before fetching the next question.  
     Use async/await and fetch().
2. **Weather Dashboard**  
   Create a weather dashboard that:
   * Fetches weather data for a given city using an API like OpenWeatherMap.
   * Displays the current temperature and weather conditions.
   * Includes error handling for invalid city names.
3. **To-Do List with Remote Storage**  
   Build a to-do list that:
   * Fetches tasks from a mock API.
   * Allows users to add, delete, and mark tasks as completed.
   * Updates the mock API with the changes (simulate using fetch()).