APP DEVELOPMENT

Challenge 1: Hello World App

Objective:
Create a simple app that displays "Hello, World!" on the screen.

Key Learning Areas:

- **Basic Terms:** Understand what an IDE is and set up your development environment.
- **Project Structure:** Familiarize yourself with the main project files (main.dart for Flutter or app.js for React).
- **Dart/JS Basics:** Learn about data types, variables, and simple functions.
- **Widgets/Components:** Use basic widgets (Flutter) or components (React) to build the UI.
- **Styling:** Apply simple styling to your text.

Steps:

1. **Set Up Environment:** Install an IDE like VS Code or Android Studio, and set up Flutter or React Native.
2. **Create a New Project:** Initialize a new Flutter or React Native project.
3. **Build the UI:**
    - **Flutter:** Use a Text widget to display "Hello, World!".
    - **React Native:** Use a Text component to display "Hello, World!".
4. **Run the App:** Use an emulator to run your app and see the output.
5. **Style the Text:** Change the font size, color, or alignment to enhance the appearance.

Outcome:
A basic app that successfully displays a styled "Hello, World!" message, reinforcing your understanding of project setup and basic UI components.

Challenge 2: Static Profile Page

Objective:
Build a static profile page that includes an image, text fields, and basic layout.

Key Learning Areas:

- **Layout Widgets:** Use Row, Column, and Stack (Flutter) or Flexbox (React) to arrange components.
- **Assets Management:** Include and display images and custom fonts.

- **User Inputs:** Add text fields to display user information.
- **Styling:** Apply styles to layout and components for a polished look.

Steps:

1. **Design the Layout:**
    - Include a profile picture at the top.
    - Add text fields for name, email, and a short bio.
2. **Implement the Layout:**
    - **Flutter:** Use Column for vertical layout and Row for horizontal arrangements.
    - **React Native:** Use Flexbox for arranging components.
3. **Add Assets:**
    - Import a profile image and include it in your app.
    - Use custom fonts for text styling.
4. **Style Components:** Adjust padding, margins, colors, and fonts to enhance the UI.
5. **Run and Test:** Ensure all components display correctly on different screen sizes using the emulator.

Outcome:
A visually appealing static profile page that demonstrates your ability to structure layouts, manage assets, and style components effectively.

Challenge 3: Interactive To-Do List App
Objective:
Create a simple to-do list app where users can add and remove tasks, demonstrating state management and user interaction.

Key Learning Areas:

- **Stateful vs. Stateless Widgets/Components:** Learn how to manage and update state.
- **User Inputs:** Handle text inputs and button clicks.
- **Lists & Arrays:** Store tasks in a list and dynamically update the UI.
- **Basic Object-Oriented Programming (OOP):** Organize code using classes and objects (especially in Dart).

Steps:

1. **Set Up the UI:**
    - Display a list of tasks (initially empty).
    - Add a text input field where users can type a task.

- Add an "Add Task" button to add tasks to the list.
- Each task should have a delete button to remove it from the list.

2. **Manage State:**
   - **Flutter:** Use a StatefulWidget and setState to update the list of tasks.
   - **React Native:** Use the useState hook to manage the task list state.
3. **Handle User Input:**
   - Capture the user's task input from the text field.
   - Add the task to the list when the "Add Task" button is pressed.
   - Remove a task when the delete button next to it is clicked.
4. **Enhance Functionality (Optional):**
   - Add a "Clear All" button to remove all tasks.
   - Add task editing functionality (allow users to modify tasks).
5. **Run and Test:**
   - Ensure the task list updates dynamically with new tasks and tasks can be deleted.
   - Test for responsiveness and correct state handling.

Outcome:
A fully functional to-do list app that demonstrates your understanding of state management, handling user inputs, list updates, and basic UI interactions.

Challenge 4: Fetch and Display Data from a Public API

Objective:
Build a simple app that fetches data from a public API (e.g., FakeStore API) and displays it in a list format.

Key Learning Areas:

- **Asynchronous Programming:** Use async/await and handle Futures or Promises.
- **HTTP Requests:** Make GET requests to fetch data from the API.
- **JSON Parsing:** Parse the fetched JSON data and map it to Dart/JS objects.
- **UI Rendering:** Display the data using list views or scrollable components.
- **Error Handling:** Manage potential errors during data fetching.
- **Navigation:** (Optional) Navigate to a detailed view when an item is selected.

Steps:

1. **Choose an API:** Select a public API like FakeStore API for fetching product data.
2. **Set Up HTTP Requests:**
    - **Flutter:** Use the http package to make GET requests.
    - **React Native:** Use fetch or Axios to retrieve data.
3. **Fetch Data:**
    - Create functions to fetch data asynchronously.
    - Handle loading states and potential errors.
4. **Parse JSON Data:**
    - Convert JSON responses into Dart classes or JavaScript objects.
5. **Display Data:**
    - Use ListView (Flutter) or FlatList (React Native) to display the list of items.
    - Show essential information like product name, price, and image.
6. **Enhance the UI:**
    - Add styling to list items for better presentation.
    - (Optional) Implement navigation to a detail page when an item is tapped.
7. **Run and Test:** Ensure data is fetched and displayed correctly, and handle edge cases like network failures.

Outcome:
A functional app that interacts with an external API, demonstrating your ability to perform asynchronous operations, handle data fetching, parse JSON, and present dynamic data within the UI.