INNOVUS LABS

Take-Home Coding Exercise(s)

Position: Software Engineer, FrontEnd (IC1)
Time Expectation: ~2 hours total

Exercise 1: Next.js + React + MUI Basics (TypeScript)

Goal

Demonstrate basic usage of **Next.js** routing, **React hooks**, **Material UI** components, and **TypeScript**.

Instructions

1. **Initialize a Next.js TypeScript Project**
   - You can run `npx create-next-app@latest --ts` to bootstrap a TypeScript-enabled Next.js project.
   - This will set up `.tsx` files, TypeScript config, etc.
2. **Create a Simple Page** (`pages/index.tsx`)
   - Fetch a small set of sample data from any public API (e.g., JSONPlaceholder, PokéAPI, etc.) and **render** it on the page.
   - Use **`getServerSideProps`** or **`getStaticProps`** in TypeScript. Show how you define the types for the data you receive.
3. **Material UI Styling**
   - Style the fetched data in a **list** using Material UI (e.g., `List` and `ListItem`).
   - Include a **custom MUI theme** that overrides at least one style (e.g., override primary color or adjust `ListItem` padding).
4. **Minor Interactivity** (Optional)
   - Use a **React hook** (like `useState`) if you want to toggle a detail view or highlight an item.
   - Keep your component typed with TypeScript interfaces or types for props, state, etc.

Key Points

- Show correct usage of **TypeScript** in Next.js (e.g., properly typed `getServerSideProps`/`getStaticProps`, typed React components).

- Display an understanding of **MUI** theming and basic **UI** structuring.
- Keep it simple but **typed**.

Estimated Time

~30–45 minutes

Exercise 2: Capacitor Plugin Usage (TypeScript)

Goal

Show basic familiarity with **Capacitor** by integrating **one** native API call in TypeScript.

Instructions

1. **Capacitor Setup**
   - If not already done, install Capacitor:
     ```
     npm install --save @capacitor/core @capacitor/cli
     npx cap init
     ```
   - Ensure your `capacitor.config.ts` (or `.json`) is properly set up.
2. **Create or Modify a Page** (e.g., `pages/capacitor.tsx`)
   - Demonstrate usage of **one** Capacitor plugin. Common examples:
     - **Camera**: Provide a button labeled “Take Photo” which triggers the camera plugin (on web fallback to file picker).
     - **Geolocation**: Provide a button labeled “Get Location” to retrieve coordinates.
3. **Display the Data**
   - Show the resulting data (photo preview or latitude/longitude) in a simple UI. Use MUI components (e.g., `Card`, `Typography`, etc.) to present it.
4. **Error Handling**
   - Handle potential errors or fallback gracefully (e.g., “Camera not supported in this environment”).
   - Type your asynchronous function properly in TypeScript (e.g., `async function handleTakePhoto(): Promise<void>`).

Key Points

- Show correct **import** and **plugin usage** from Capacitor in TypeScript.

- Use MUI to display the result.
- Demonstrate minimal but clear **error handling**.

Estimated Time

~30 minutes

Exercise 3: React Hooks & State Management (TypeScript)

Goal

Demonstrate the ability to manage state in multiple components and handle user input in a **TypeScript** React environment with MUI. This exercise is intentionally more **complex** to mimic a scenario such as analytics tracking or shared UI state.

Instructions

1. **Create a React State "Provider"**
    - Build a custom React Context (e.g., `AnalyticsContext`) or use a top-level component to hold **shared state**. For example, create an interface:

      ```
      interface AnalyticsContextType {
            hoverEvents: number;
            clicks: number;
            logHover: () => void;
            logClick: () => void;
      }
      ```

    - Implement this in a `useReducer` or `useState` + context combination to track how many times something was hovered or clicked.
2. **Build a Multi-Component UI**
    - **Component A**: A list of items (e.g., "Todo" list, "Favorite Books" list, or placeholders). When the user **hovers** over an item, it calls `logHover()` from the context to update the analytics state.
    - **Component B**: A simple button or an action element. When clicked, it calls `logClick()` from the context.
3. **Analytics Dashboard Display**

- In a separate component (e.g., `AnalyticsPanel`), display the **current** count of `hoverEvents` and `clicks`.
- Show that changes in **Component A** or **Component B** update the analytics data in real time.

4. **Form Integration (MUI)**
   - Optionally, you can add a small form (with `TextField` and a `Button`) that **adds** new items to the list in **Component A**.
   - Use **TypeScript** to define the shape of these items (e.g., an interface `TodoItem { id: number; title: string; }`).
5. **Type Definitions**
   - Ensure your React components, context, and any data objects are properly typed (e.g., `React.FC<Props>`).
   - If you're using a `useReducer`, define your action types and state interface clearly.

Key Points

- **React hooks** (Context, useReducer, useState) to manage a shared analytics-like state.
- **TypeScript** everywhere (strict typing for props, state, and actions).
- **MUI** for form elements, list, or other UI parts.
- Show how **multiple components** can interact with **shared state** (e.g., logging hovers/clicks).

Estimated Time

~30 minutes

Submission Guidelines

1. **Repository Structure**
   - You can create a **single Git repo** with subdirectories (`exercise-1`, `exercise-2`, `exercise-3`) **OR** a single Next.js project with multiple pages and folders for each exercise.
2. **README**
   - Include a top-level README or per-exercise README with:

INNOVUS LABS

        - How to install & run: `npm install && npm run dev` (or `yarn install && yarn dev`).
        - Any relevant setup steps for Capacitor (e.g., `npx cap sync`).
        - A brief explanation of how you approached each exercise and any challenges you faced.
3. **Time Constraint**
    - Each exercise is designed to be small. Aim for around **2 hours total** if you combine all three. Don't worry if some polish is missing—focus on **demonstrating knowledge** and **writing clean, typed code**.

What We're Looking For

1. **TypeScript Proficiency**
    - Typed data fetching, typed components, well-defined interfaces or types for props/state/actions.
2. **Next.js Fundamentals**
    - Basic understanding of routing and data fetching (`getServerSideProps` / `getStaticProps`).
3. **Material UI Usage**
    - Familiarity with core components, ability to apply theming or overrides.
4. **Capacitor Knowledge**
    - Basic ability to set up and use at least one plugin.
5. **React State Management**
    - Clean usage of hooks (`useState`, `useReducer`, `Context`), multi-component interaction.
6. **Code Quality**
    - Clear, readable, and maintainable code.

Final Note

The goal is to **sample your capabilities** with Next.js, React, TypeScript, MUI, and Capacitor in a **manageable timeframe**. We look forward to seeing your approach! If you have any questions, feel free to reach out. Good luck!