UTT
UNIVERSIDAD TECNOLÓGICA DE TIJUANA
GOBIERNO DE BAJA CALIFORNIA

Assignment:

Selection of Design Patterns

BY:

Daniel Chavez Madrigal

GROUP:

9-B

SUBJECT:

Desarrollo Móvil Integral

PROFESSOR:

Ray Brunett Parra Galaviz

Tijuana, Baja California, 10 de enero del 2025

Selection of Design Patterns: Component Composition and Context API in React Native with Expo

1. Introduction to Design Patterns: Design patterns are reusable solutions to common problems in software development. They provide a structured approach to solving issues, making code more maintainable and scalable. In the context of React Native with Expo, design patterns help in building robust mobile applications.

2. Component Composition: Component composition is a core concept in React development. It involves building complex UIs by combining smaller, reusable components. This approach promotes code reuse, improves readability, and makes the application easier to maintain.

- **Advantages:**
    - **Reusability:** Components can be reused across different parts of the application, reducing duplication.
    - **Maintainability:** Smaller components are easier to manage and update.
    - **Testability:** Individual components can be tested in isolation, improving the reliability of the application.
- **Disadvantages:**
    - **Complexity:** Managing a large number of small components can become complex.
    - **Overhead:** Breaking down the UI into many small components can introduce some performance overhead.

3. Context API: The Context API is a React feature that allows you to share state across the entire application without passing props down manually at every level. It is particularly useful for managing global state, such as user authentication, themes, and settings.

- **Advantages:**
    - **Simplified State Management:** Reduces the need for prop drilling, making state management more straightforward.

    - **Global State:** Easily manage global state that needs to be accessed by multiple components.
    - **Flexibility:** Can be used in combination with other state management libraries if needed.
- **Disadvantages:**
    - **Performance:** Frequent updates to the context can cause performance issues, as all-consuming components will re-render.
    - **Complexity:** Managing multiple contexts can become complex in larger applications.

4. Combining Component Composition and Context API: Using Component Composition and Context API together can create a powerful and flexible architecture for your React Native application.

- **Example Use Case:**
    - **Component Composition:** Break down the UI into small, reusable components. For instance, create separate components for buttons, forms, and navigation elements.
    - **Context API:** Use the Context API to manage global state, such as user authentication. Create a context for the user state and provide it to the entire application. Components can then consume this context to access user information without prop drilling.

5. Factors to Consider When Selecting Design Patterns:

- **Project Requirements:** Assess the specific needs of your project, such as the complexity of the UI and state management requirements.
- **Scalability:** Choose patterns that support scalability, ensuring that your application can grow and evolve without becoming difficult to manage.
- **Maintainability:** Select patterns that promote clean, readable, and maintainable code.
- **Performance:** Consider the performance implications of the design patterns you choose.

6. Conclusion: By leveraging Component Composition and Context API, you can build a robust, maintainable, and scalable React Native application with Expo.

These design patterns help in managing complexity, improving code reuse, and simplifying state management.

Refences:

Mittal, A. (2019, September 29). *How to use React Context API to build React Native, Expo and Firebase apps*. DEV Community. https://dev.to/amanhimself/how-to-use-react-context-api-to-build-react-native-expo-and-firebase-apps-bei