**Team Members**

Edward F. Magtoto

Anh Duc Hoang

Melody Shih

Ching Yin Lam (Sean)

**Project Phase 2: App Architecture**

**UI toolkit:**

**Navigation Strategy:**

**Hardware:**

**Database Storage:**

**Architectural Decision Record (ADR)**

**Architectural Decision Record (ADR)**

**Title:** Selection of Native Base as the UI Toolkit for React Native App

**Context:** We are in the process of selecting a UI toolkit for our React Native mobile app. The UI toolkit choice will significantly impact the app's user interface design, components, and overall user experience.

**Decision:** We have decided to adopt Native Base as the UI toolkit for our React Native app.

**Rationale:**

* **Cross-Platform Support:** Native Base is known for its cross-platform support for both iOS and Android, which aligns with our requirement to reach a wide audience on different devices.
* **Rich Component Library:** Native Base provides a comprehensive library of pre-designed UI components, including buttons, forms, headers, and more. This extensive component library will significantly accelerate the development process and reduce the need for custom design and development.
* **Customization and Theming:** Native Base offers a high level of customization and theming capabilities. This flexibility enables us to create a unique and visually appealing user interface that aligns with our app's branding and design requirements.
* **Community and Documentation:** Native Base has an active developer community and provides extensive documentation, making it easier for our development team to access resources, receive support, and troubleshoot issues as they arise during the development process.
* I**ntegration Ease:** Native Base integrates seamlessly with React Native, offering ease of integration into our existing development workflow. This will help streamline development and reduce potential integration challenges.
* **Scalability:** The scalability of Native Base allows us to accommodate future feature expansions and the app's potential growth while maintaining design consistency.

**Consequences:**

* **Integration Challenges:** While Native Base integrates with React Native, there is a potential for integration challenges or conflicts with other third-party libraries and packages. These challenges may lead to delays in development and troubleshooting efforts.
* **Performance Impact:** Extensive customization and theming can introduce performance overhead, particularly on lower-end devices. This may impact the app's responsiveness and user experience, requiring additional optimization efforts.
* **Complexity:** Native Base's customization capabilities, while beneficial, can also introduce complexity into the app's codebase. This may lead to challenges in maintaining and extending the app in the long term.

**Considered Alternatives:**

We considered alternative UI toolkits like React Native Elements and UI Kitten. While these options have their strengths, we chose Native Base due to its extensive component library, customization capabilities, and strong community support.

**In summary**, the decision to adopt Native Base as the UI toolkit for our React Native app was based on its cross-platform support, rich component library, customization and theming options, community and documentation support, and ease of integration. This choice will contribute to the efficient and visually appealing development of our mobile app.

**Architectural Decision Record (ADR)**

**Title:** Hybrid of Tab and Bottom Tab Navigation for React Native App

**Context:** We are in the process of designing the navigation strategy for our React Native mobile app. The navigation strategy plays a crucial role in ensuring a user-friendly and intuitive user experience.

**Decision:** We have decided to implement a hybrid of Tab Navigation and Bottom Tab Navigation as the primary navigation strategy for our React Native app.

**Rationale:**

* **Tab Navigation:** We are incorporating Tab Navigation to provide easy access to different sections or categories within the app. This navigation approach is suitable for our app, which contains distinct and essential sections that users frequently access.
* **Bottom Tab Navigation:** We are opting for Bottom Tab Navigation to place the primary navigation tabs at the bottom of the screen. Placing tabs at the bottom is a common design pattern on mobile platforms, particularly on Android, and it aligns with user expectations.
* **User-Friendly Experience:** The hybrid approach combines the strengths of Tab and Bottom Tab Navigation to create a user-friendly and intuitive experience. Users can access primary sections with a single tap, while the bottom tab layout ensures easy access with one-handed usage.
* **Cross-Platform Consistency:** The hybrid approach is designed to provide a consistent user experience across both iOS and Android platforms. This consistency is crucial to make navigation patterns familiar to users on both platforms.
* **Ease of Implementation:** React Navigation, which supports both Tab and Bottom Tab Navigation, allows us to implement this hybrid strategy efficiently. The development team is familiar with React Navigation, simplifying integration.
* **Customization:** The chosen navigation approach offers a high level of customization, enabling us to tailor the user interface and navigation experience to align with our app's branding and design requirements.

**Consequences:**

* **Integration Challenges:** Combining Tab and Bottom Tab Navigation may introduce integration challenges. Ensuring smooth transitions between different navigational patterns may require additional development effort and testing.
* **Complexity:** The hybrid approach can introduce complexity into the codebase, especially when customizing navigation behavior. This complexity can lead to challenges in maintaining and extending the app in the long term.
* **Performance Impact:** Extensive customization and complex navigation structures can introduce performance overhead, potentially affecting the app's responsiveness and user experience, necessitating additional optimization efforts.

**Considered Alternatives:**

We considered alternative navigation strategies, including Stack, Drawer, or Top Tab Navigation, but the hybrid approach aligns best with our app's structure, user expectations, and user-friendliness.

**In summary**, the decision to implement a hybrid of Tab and Bottom Tab Navigation as the primary navigation strategy for our React Native app was made, acknowledging potential integration challenges, increased complexity, and potential performance impacts as potential negative consequences. These concerns will need to be addressed during the development process to ensure the app's success.

**Architectural Decision Record (ADR)**

**Title: Hardware Components for the Expense Tracking App**

**Context:** We are designing the architecture of our expense tracking app, which involves capturing and managing expense-related data, including spending location, receipts, authentication, and media file access.

**Decision:** We have decided to include the following hardware components in our app:

**GPS (Global Positioning System):**

* **Use Case:** GPS will be used to capture the spending location when users record their expenses.
* **Benefits:** Provides location context to expense entries, enhancing expense tracking and analysis.
* **Considerations:** We acknowledge that the use of GPS can impact device battery life and may lead to user concerns about privacy, as location data is sensitive.

**Camera:**

* **Use Case:** The camera feature will allow users to take pictures of receipts and attach them to their expense entries.
* **Benefits:** Simplifies expense tracking by enabling users to capture and store images of receipts, providing visual proof of transactions.
* **Considerations:** The camera feature may consume significant storage space and potentially slow down devices with limited resources.

**Fingerprint Sensor:**

* **Use Case:** The fingerprint sensor will enable quick and secure user authentication for accessing the app.
* **Benefits:** Enhances user convenience by providing a secure and efficient authentication method.
* **Considerations:** Some devices may not have a fingerprint sensor, leading to authentication disparities between users with and without this hardware feature.

**Media File Access:**

* **Use Case:** Access to media files is required for users to import pictures, including receipts and other images.
* **Benefits:** Enables users to attach media files to their expense entries, making expense tracking more comprehensive.
* **Considerations:** Requesting media file access permissions may raise privacy concerns among users, and managing imported media files may consume additional storage space.

**Consequences:**

* **Battery Drain and Privacy Concerns:** The use of GPS for spending location may lead to increased battery consumption and user concerns about data privacy.
* **Storage and Performance Impact:** Camera functionality may result in significant storage usage and potentially impact app performance on resource-constrained devices.
* **Authentication Disparities:** Relying on a fingerprint sensor for authentication can exclude users with devices lacking this feature.
* **Privacy Considerations:** Requesting media file access permissions may lead to privacy concerns, and managing imported media files may add complexity and resource usage to the app.

**Considered Alternatives:**

We considered alternative hardware components, such as microphone for voice input, but concluded that GPS, camera, fingerprint sensor, and media file access are the most relevant and beneficial for the app's core functionality and user experience.

**In summary, t**he decision to incorporate GPS, camera, fingerprint sensor, and media file access as hardware components in our expense tracking app was made while acknowledging potential negative consequences, including battery drain, storage usage, authentication disparities, and privacy concerns. These concerns will need to be addressed during the development process to ensure a balanced user experience.

**Architectural Decision Record (ADR)**

**Title:** Database Storage Solution for the Expense Tracking App

**Context:** We are designing the architecture of our expense tracking app, which involves the storage of financial data, including expense entries and images of receipts.

**Decision:** We have decided to use Firebase Firestore as the database storage solution for our expense tracking app.

**Rationale:**

* **NoSQL Structure:** Firebase Firestore is a NoSQL database that provides flexibility in handling structured financial data, such as expense entries and categories, while also efficiently storing binary image data for receipts.
* **Real-Time Synchronization:** Firestore offers real-time data synchronization, allowing users to access and update their expense data from multiple devices and ensuring consistency across platforms.
* **Offline Capabilities:** Firestore's offline data access feature is vital for our expense tracking app, enabling users to record expenses and attach images of receipts even when they are offline. The app will automatically sync the data when an internet connection is available.
* **Scalability:** Firestore automatically scales to accommodate growing data volumes and user bases. This is important as our app may expand in the future.
* **Security:** Firebase provides robust authentication and security features, ensuring the privacy and protection of financial and image data.
* **Image Storage:** Firestore's integration with Firebase Cloud Storage makes it straightforward to store and retrieve images of receipts securely. This aligns perfectly with our requirement to save pictures of receipts.

**Consequences:**

* **Efficient Data Management:** Firestore efficiently handles both structured financial data and binary image data, ensuring a seamless user experience.
* **Real-Time and Offline Functionality:** Users can access, update, and synchronize their expense data and receipt images in real-time, even when offline.
* **Scalability and Security:** Firestore's scalability and security features provide room for app growth while protecting user data.

**Considered Alternatives:**

We considered alternatives such as SQLite and Realm, but Firestore was chosen due to its combination of NoSQL flexibility, real-time synchronization, offline capabilities, and seamless image storage.

**In summary,** the decision to use Firebase Firestore as the database storage solution for our expense tracking app was made to efficiently manage financial and image data, provide real-time and offline capabilities, ensure scalability, and enhance security. This choice contributes to a comprehensive and user-friendly app experience.