Architectual Decision Record (ADR)

Group 7

Introduction

The architectural choices made while creating a mobile application for a retail business are described in this document. Customers will be able to browse and buy products, check the status of their deliveries, see their order history, and take part in a loyalty program through the app.

Scenario Overview

The following attributes are necessary for the retail company:

- View order history and browse products in offline mode.
- Push alerts for new product arrivals, order updates, and special deals.
- Connectivity with multiple payment gateways to ensure safe transactions.
- Tracking of user behavior for interactions with loyalty programs, purchases, and product views.
- Effective management and enhancement of product photos.
- Support for a variety of languages and cultural tastes.

Architectual Decision

1. App Type: Hybrid App

Rationale:

- Cross-Platform Compatibility: By utilizing a single codebase for both iOS and Android, hybrid apps cut down on development expenses and time.
- Performance: Near-native performance is sufficient for the needs of the app with modern hybrid frameworks like React Native and Flutter.
- Offline Mode: Hybrid frameworks offer strong offline functionality support.
- Cost: Compared to creating and maintaining individual native apps for each platform, hybrid development is more economical.

2. UI Framework: React Native

Rationale:

- Popularity and Community Support: There is a sizable community and copious documentation for React Native, which makes it widely used.
- Performance: React Native provides seamless user experience and near-native performance.
- Code Reusability: A high level of cross-platform code reuse.

- When required, simple integration with native modules is provided.

3. Backend Language: Node.js

Rationale:

- Scalability: Node.js is well known for its efficiency and scalability, which are crucial for managing multiple concurrent connections in applications like real-time order tracking and alerts.
- Speed: It is appropriate for real-time applications due to its non-blocking I/O operations.
- Community and Libraries: Vast array of libraries and packages covering a range of features.

4. Permissions: Use OAuth 2.0 to implement fine-grained permission management.

Rationale:

- Security: OAuth 2.0 can interface with several identity providers and offers secure access delegation.
- Flexibility: Accommodates various roles and permissions that can be customized to meet the requirements of apps.
- User Experience: By allowing users to selectively grant permissions, security and trust are increased.

5. Data Storage: PostgreSQL is used for server-side storage and SQLite for local storage.

Rationale:

- SQLite: Compact and effective for mobile devices' offline data storage.
- PostgreSQL: Capable of handling intricate queries for server-side operations, robust and scalable.
- Synchronization: When an internet connection is available, effective synchronization mechanisms between local SQLite databases and the central PostgreSQL database.

6. Push Notification: Firebase Cloud Messaging (FCM)

Rationale:

- Cross-Platform Compatibility: FCM works with both Android and iOS.
- Dependability: a track record of delivering messages with consistency.
- Integration: React Native integration is simple.

7. Payment Gateway Integration: Stripe And PayPal

Rationale:

- Security: Both gateways meet PCI requirements and are extremely safe.
- Easy to Use: Simple integration with mobile apps and user-friendly interfaces.
- Compatibility: Support for several currencies and geographical areas, in line with the retail company's goals for global expansion.

8. Analytics Tool: Google Analytics for Firebase

Rationale:

- Detailed Insights: Offers extensive analytics and tracking of user behavior.
- Integration: Smooth connection to additional Firebase services.
- Real-Time Data: Real-time analytics to track app performance and user engagement.

9. Image Storage and Optimization: Amazon S3 with CDN and image optimization methods via CloudFront

Rationale:

- Scalability: Keeping a lot of photos is made possible by Amazon S3, which is very scalable.
- Performance: Quick image delivery is guaranteed by CloudFront CDN.
- Optimization: Facilitates performance-enhancing methods such as compression and lazy loading.

10. Localization: i18next for localization

Rationale:

- Flexibility: i18next is simple to integrate with React Native and supports several languages.
- Community Support: Well, documented and supported by the community, this product is widely used.
- Adaptability: Able to manage diverse cultural preferences and intricate localization requirements.

Rationale:

To ensure cross-platform compatibility and lower development costs, the choices made in this ADR balance performance, scalability, cost-effectiveness, and user experience using a hybrid app approach with React Native. Because of its effectiveness with real-time features, Node.js was selected, and SQLite and PostgreSQL are used to manage server-side and local data, respectively, to ensure smooth offline functionality and synchronization. Secure permission management is facilitated by OAuth 2.0, and dependable push notifications are guaranteed by Firebase Cloud Messaging. For security and usability, Stripe and PayPal are chosen, and for image handling, Amazon S3 and CloudFront are the best options. Along with meeting all functional and non-functional requirements, the app is made robust, scalable, and user-friendly by i18next supporting the app's localization needs and providing detailed insights into user behavior through Google Analytics for Firebase.

Conclusion:

The above-mentioned architectural choices offer a strong basis for creating a dependable, adaptable, and user-friendly mobile application for the retail business. React Native and Node.js, along with the hybrid approach, guarantee effective development and maintenance while satisfying the business objectives and user requirements of the organization.