Architectural Decision Record for Retail Company Mobile App

Prepared for

Nick Hamnett, CPRG 303 Professor

Prepared by

Tamara Mahmoud

Software Development - Cohort A

School of Advanced Digital Technology

SAIT

Requested by

Nick Hamnett, CPRG 303 Professor

8 October 2023

Architectural Decision Record for Retail Company Mobile App

Scenario 1:

- Native, web, or hybrid app

Given the need for offline support, I decided on native because it would be wise to develop native applications. Native apps run better and better use the hardware and features unique to the device. Swift can be used for iOS, whereas Kotlin is recommended for Android.

- UI Framework

iOS: To construct user interfaces declaratively, use SwiftUI, which improves the code's readability and maintainability.
Android: To improve the UI development process, use Kotlin with Jetpack Compose, a fully declarative component-based method.

- Backend language

Considering Node.js scalability, performance, and the sizable npm ecosystem, it might be a viable pick for the backend. Additionally, it is non-blocking, which is crucial for effectively managing concurrent requests and enhancing the app's efficiency.

- Permissions

The software will ask users for several permissions:
Internet: To make purchases, update order status, and sync data.
Push Notifications: To deliver order updates, announcements of brand-new merchandise, and special deals.
Storage: To save local copies of the order history and product photos for offline mode.

- Data storage

Local: To support offline mode, use Realm or SQLite to store data locally on the devices.
Remote: A NoSQL database like MongoDB and a relational database like PostgreSQL can work well together for unstructured data. You can store product photos in AWS S3.

Issue:

Determining the kind of application development (native, web, or hybrid) for a retail company's mobile app that facilitates browsing, shopping, order tracking, and loyalty program elements is a crucial architectural challenge. This problem is now addressed to provide the groundwork for future scalability, performance, and ideal user experience.

Decision:

I have chosen to create a Native app using React Native for the user interface, Node.js for the backend, SQLite for offline storage, AWS S3 for image storage, MongoDB for data storage, Redux for state management, Firebase for push notifications, i18next for internationalization.

Status:
Approved.

Group:
Application Type.

Assumptions:
There will be a variety of users and different operating systems (Android, iOS).
For best performance, the app needs access to device-specific functionality.
Native applications offer superior support for the app's features, such as offline browsing and push notifications.

Constraints:
Separate iOS and Android development is necessary, which could add time and money.
Platform-specific maintenance and updates are required.

Positions:
Native App: Better user experience, improved performance, and accessibility to device-specific functionality.
Web apps are easy to deploy and require little programming, but performance and feature access limitations exist.
Hybrid: A hybrid app combines both aspects, providing balance, but it might not fully utilize the device's capabilities.

Argument:
The decision to utilize a Native app is driven by the desire for an enhanced user experience, high performance, and the seamless integration of sophisticated features like offline mode and push notifications. The long-term advantages of user engagement, feature optimization, and overall app quality outweigh any potential development time and expense increases.

Implications:
Requires programmers with experience in both the iOS and Android programming environments.
Increased start-up expenses for development.
The enhanced user experience might lead to more user engagement and retention.
Seamlessly incorporates complicated features and functionalities.

Related Decisions:
This setting affects and limits the UI/UX frameworks that can be used and how data is stored and push notifications are handled.

Related Requirements:
Offline browsing must be supported by the app.
A dependable push notification service integration.

Support in many languages for clients from outside.

Related Artifacts:
Feature integration documentation.
UI/UX Design Prototypes
Data Flow Diagrams

Related Principles:
User-Centric Design: The architecture is selected with care for offline use, performance, and internationalization to guarantee the best user experience.
User-Centric Design: Choosing a native app ensures the user experience is prioritized.
Performance efficiency: Select a strategy that ensures excellent performance and easy feature integration.

Scalability:
In order to support a growing user base and a variety of products, scalability is a crucial consideration when selecting the backend and data storage options.