Mobile Application Development

CPRG 303- F

Assignment: Architectural Decisions

Submitted by:

MAD Group
Aamir, Nida
Bates, Tristin
Dela Cruz, Catherine
Marfo, Isaac

Architectural Decision Record (ADR) for Scenario 3 - Food Ordering Mobile App

This document outlines the architectural decisions made for Scenario 3, covering critical choices such as the type of app, backend language, and frameworks. It includes the options considered, the rationale behind the decisions, their consequences, and the follow-up actions required for successful implementation and ongoing improvement.

Decision Title: Type of App

Options Consider

- Native
- Web
- Hybrid App

Decision

The decision was made to use the **Hybrid App** for Scenario 3.

Rationale

- Hybrid apps bring together the advantages of both native and web apps, letting you connect with customers on Android and iOS with a single codebase.
- This approach saves you time and money since you do not have to create separate apps for each platform.
- Hybrid apps provide excellent user experience and are quicker and easier to build. A hybrid app is an intelligent choice if you want a fast and affordable way to set up your online ordering system.

Consequences

- Hybrid apps are cost-effective, faster to build, and easier to maintain since one codebase works for Android, reducing development time and effort for updates.
- Hybrid apps might not perform as efficiently as native apps, especially for complex features, have limited access to specific device functions, and might not provide the same smooth user experience.

Follow-up Actions

- Monitor the app's performance on Android to ensure it meets user expectations and make optimizations as needed.
- Test thoroughly, collect members' feedback, and plan for future updates to ensure scalability and improvements.

Decision Title: UI Framework

Options to Consider

- React Native
- Flutter
- Swift (for iOS) / Kotlin (for Android)

Decision

The decision was made to use **React Native** for the UI framework.

Rationale

React Native lets us build mobile apps for both Android and iOS using a single codebase. This will help save time and money. It is also commonly used, has a big community and a lot of helpful documentation. React Native also works well with Expo, which we're already using in this project, and it integrates easily with Firebase. Overall, it's a flexible choice that fits our needs for this app.

Consequences

- **Positive:** One codebase for both Android and iOS, which accelerates development and cuts down on maintenance.
- **Negative:** React Native may not perform as well as native apps, especially with complex animations or heavy features. There could also be device-specific issues to fix.

Follow-up Actions

- Conduct regular performance testing on both Android and iOS to make sure it works well on different devices.
- Stay updated on React Native and Expo developments to ensure smooth app functionality.

Decision Title: Backend Language

Option Considered

- Node.js
- PHP
- Java

Decision

The decision was made to use **Node.js** for Scenario 3.

Rationale

- Node.js is great for a food ordering app because it handles multiple requests quickly and efficiently, perfecting real-time features like order tracking and notifications.
- Its scalability means you can add more services as your user base grows without slowing down the app.
- Many libraries and tools available through NPM help developers work faster by offering ready-to-use solutions.

Consequences

- Node.js can handle many users at once, making it ideal for a food ordering app with real-time features, and it can quickly grow with more users while speeding up development with its many libraries and tools.
- Using Node.js can result in complicated code when handling many asynchronous tasks, some libraries may not be reliable, and developers new to Node.js might struggle, which can slow down early development.

Follow-up Actions

Use best practices to manage asynchronous code and keep it organized. Regularly check the third-party libraries for quality and security and monitor the app's performance.

Decision Title: Permissions

Options Considered

- Location Access
- Push Notification
- Camera/Photo Access
- Microphone Access
- Contacts Access

Decision

The decision was made to require Location Access and Push Notifications permission for the food ordering app.

Rationale

- **Location Access:** This permission enables the app to locate nearby restaurants and offer accurate delivery times, enhancing the user experience by providing relevant options and estimated arrival times.

- **Push Notifications:** Notifications are essential for order updates, promotions, and reminders. Users are kept informed about their order status, which is critical for a seamless food ordering experience.
- Limiting permissions to these two reduces potential security and privacy concerns, ensuring users feel comfortable using the app without unnecessary access to personal data.

Consequences

- **Positive:** Minimal permissions make the app user-friendly and privacy-conscious, potentially increasing user trust. Notifications improve engagement by keeping users updated on order statuses, and location access enhances the relevance of restaurant suggestions.
- **Negative:** Users may choose to disable these permissions, impacting the app's ability to provide real-time updates or accurate delivery estimates. Additionally, future features requiring more permissions may need separate permission requests, which could affect user retention.

Follow-up Actions

- Continuously monitor user feedback to identify any issues related to permissions.
- Update the app to handle cases where permissions are denied, ensuring graceful degradation of features and notifications.
- Regularly audit permissions to ensure compliance with data protection laws and best practices.

Decision Title: Data Storage

Options Considered

- Cloud Database (e.g., Firebase)
- On-dev ice Storage
- Relational Database (e.g., MySQL)
- NoSQL Database (e.g., MongoDB)

Decision

The team decided to use a Cloud Database, specifically Firebase, for real-time data storage and synchronization.

Rationale

- Firebase offers real-time data synchronization, which is ideal for a food ordering app where users need instant updates on their orders.
- It provides managed authentication, analytics, and push notifications, which reduces the need for additional third-party services.
- Additionally, Firebase scales efficiently with an increasing user base, supporting our goal to manage data dynamically as the app grows**.**

Consequences

- **Positive:** Firebase's managed backend services reduce development time, making it easy to store and retrieve data in real time. Firebase also offers security and scalability, essential for maintaining app performance as user demand grows.
- **Negative:** The app's reliance on a third-party service could become a vulnerability if Firebase experiences downtime. Furthermore, data storage costs may increase significantly as user data and traffic grow, impacting long-term budgeting.

Follow-up Actions

- Establish data backup procedures to avoid data loss and explore redundancy options in case Firebase services are interrupted.
- Monitor usage patterns and storage costs to manage expenses effectively.
- Periodically review Firebase updates and security protocols to ensure compliance with best practices and enhance data protection.

Decision Title: Any Additional Frameworks or Technology stacks

Options to Consider

- Socket.io
- Firebase
- Expo

Decision

- Firebase
- Expo

Rationale

- Firebase and Expo both serve different purposes that would be beneficial for the app.
- Firebase gives a framework that allows easy integration of databases, authentication and push notifications.
- Firebase will allow easy integration of authentication and sign in methods including signing in with Google, Apple, Facebook, GitHub and others.
- Firebase will also help with handling push notifications, which will be beneficial for notifying the user with any updates regarding their order.
- Expo will also help increase the time it takes to create a prototype as Expo helps make React Native code more abstract and malleable.
- Expo will make debugging, testing and deployment easier, while also working directly with React Native.

Consequences

- Using Firebase will create more reliability on third-party services and could potentially cause problems for the app if the services ever go down.
- Everybody on the team will have to learn how to work with Firebase and how to integrate it throughout the whole app.

- If the app requires deeper React Native features that Expo does not support, the Expo may have to be reconsidered when it comes to its inclusion in the app.

Follow-up Actions

- Explore Documentation on both Firebase and Expo to determine if both are good matches for this app.
- Test both Firebase and Expo integration on different devices to see performance.