Kings Crossing

Architectural Decisions

Prepared for

Jaspreet Gill

Prepared by

Curtis Narong (Tyger), Ifechukwu Idemili (Anthony),

Jean-Pierre Nde-Forgwang (JP), Valentine Jingwa

Software Development – Cohort G

School of Advance Digital Technology

SAIT

6 October 2023

Architectural Decisions

Within your group, discuss the following architectural decisions (AD):

- Native, web, or hybrid app
- UI Framework
- Backend language
- Permissions
- Data storage
- Any additional frameworks or technology stacks

Scenario 1

– *Summary*

Retail company wants to create an app that allows customers to browse and purchase products, view order history, track deliveries, and participate in a loyalty program.

– *App Type*

We recommend using a Native app Rational since we would be dealing with a multitude of products with images or videos, customer interactions, and a few additional features. This framework will provide better flexibility for the developers and seamless integration with multiple devices.

– *UI Framework*

We decided on React Native for its cross-platform capabilities, strong community support, and ease of integration with native modules.

– *Backend Language*

We chose Node.js for the backend. Node.js is scalable, has a large ecosystem, and allows for quick development cycles.

– *Permissions*

The app will request only essential permissions such as Internet access, push notifications, and location.

– *Data Storage*

We chose a combination of SQLite for local storage on the device and MongoDB for the server-side database.

Scenario 2

- *Summary*

The development team is tasked with creating a cross-platform social networking mobile app for a university, enabling students and professors to interact, share academic information, and manage schedules. The app must ensure data privacy, support offline use, optimize for various devices, and provide secure login through Active Directory integration while also featuring accessibility options and push notifications. Ensuring a user-friendly experience, data synchronization, and adherence to data protection regulations are paramount.

- *Native/Web/Hybrid App*

Given the app's need to support both iOS and Android platforms (Requirement 1), we'd choose a Hybrid App approach using React Native. This allows us to write once and deploy on both platforms, optimizing development time.

- *UI Framework*

Reflecting on the importance of ensuring inclusivity (Requirement 7), if React Native was integrated, its rich UI component library would be ideal; React Native Elements or NativeBase can offer a responsive UI that integrates accessibility features.

- *Backend Language*

Considering the social networking nature of the app, Node.js with Express.js would be our choice. This setup can manage the demands of real-time updates and multiple concurrent connections efficiently.

- *Permissions*

Addressing the need to integrate with an existing Active Directory system (Requirement 4), we'd use OAuth 2.0. This ensures secure, token-based authentication, with students and professors having distinct permissions.

- *Data Storage*

Given the importance of offline capabilities (Requirement 2) and handling sensitive information (Requirement 6), we’d choose a mix of MongoDB for cloud storage and a secure local database like SQLite or Realm. This balance ensures offline access and secure data storage.

Scenario 3

– *Summary*

Our client is a local restaurant who do delivery and pickups. They want an app capable of saving user relevant information.

– *Native, Web, or Hybrid App*

We chose to develop a Native app for both Android and iOS platforms. Native apps provide superior performance and better access to device features like GPS, which is crucial for our location-based services.

– *UI Framework*

For the UI framework, we unanimously agreed on using Flutter. It allows us to write the UI code once and deploy it on both Android and iOS, ensuring a consistent user experience across platforms. Flutter also has a rich set of pre-designed widgets that can speed up the development process.

– *Backend Language*

We chose Python with Django for our backend language and framework. Django offers robust features like an admin panel and ORM, which can speed up the development process. Python's extensive libraries also make it easier to integrate with various APIs for payment and inventory management.

– *Permissions*

The app will request the following essential permissions:

- Location services for accurate restaurant recommendations and delivery tracking.
- Internet access for data synchronization and order placement.
- Push notifications for order updates and promotions.

– *Data Storage*

For data storage, we decided on a combination of local SQLite databases for client-side storage and a centralized PostgreSQL database for server-side storage. SQLite will handle features like caching restaurant menus for offline use, while PostgreSQL will manage user profiles, order history, and reviews.

Scenario 4

- *Summary*

The team is tasked with developing a mobile app for a transportation company, enabling secure ride bookings, real-time tracking, and payment processing. Ensuring accurate and continuous geolocation services, secure transactions, and data protection is crucial due to the sensitive nature of user and payment information involved. Therefore, strategic technological and architectural decisions are paramount to create a seamless, secure, and efficient user experience for both drivers and passengers.

- *Native/Web/Hybrid App*

Given the indispensable need for pinpoint accuracy in geolocation services (Requirement 1) and the imperative of non-stop, real-time tracking (Requirement 2), our inclination leans decidedly toward adopting a Native App approach. This choice, while potentially demanding a heftier investment in development time for varied platforms, staunchly assures peak performance, exacting geolocation precision, and fluid, uninterrupted real-time updates.

- *UI Framework*

To cater to the app's need to visually represent ride routes, estimated arrival times, and other geolocated elements effectively on a map (Requirement 3), we would opt for React Native due to its ability to deliver a near-native performance and user experience, while also providing a large array of libraries and community support which can expedite the development process.

- *Backend Language*

In adherence to the app’s crucial need to adeptly visualize ride routes, anticipated arrival times, and other pertinent geolocated entities on a map (Requirement 3), we staunchly advocate for selecting React Native. This choice is underpinned by its prowess in delivering a nearly unbridled native performance and user experience, further enriched by a substantial library and an expansive community support network, which can significantly streamline and expedite the development trajectory.

- *Permissions*

In light of the app's unwavering responsibility to fortify user information and transaction data (Requirement 8), the deployment of OAuth 2.0 for steadfast token-based authentication, coupled with a potent role-based access control (RBAC) system, emerges as a judicious choice. This approach not only staunchly guards access but also meticulously delineates permissions and scopes for drivers and passengers, thereby ensuring an uncompromised, contextually aware functionality tailored to different user categories.

- *Data Storage*

Navigating the necessity to securely store sensitive user profiles, ride histories, and payment details while ensuring data integrity and scalability (Requirement 6), we endorse a dual-approach. Selecting PostgreSQL, recognized for its ACID compliance, guarantees reliable transactions and robust data integrity, particularly for payment and ride data. Concurrently, for user profiles and other nontransactional data, a NoSQL database like MongoDB is pivotal, offering schema flexibility and scalability to efficiently manage burgeoning user data and ride metrics.