LEGACY SYSTEM: HAILO TAXI HAILING APP

SOFTWARE ENGINEERING ASSIGNMENT 2

May 26, 2024

Omphemetse Senna

University of Pretoria

INTRODUCTION

BACKGROUND INFORMATION

Hailo was a British tech-taxi hailing business[2][5] that connected taxi drivers and customers(passengers) through an Android and iOS mobile app[6], providing safe and convenient transportation[1]. Founded in late 2010, it debuted in London in 2011 and then moved to North America[3] by the end of 2012. The software enabled real-time tracking, secure payment processes, and wheelchair-accessible features.

Hailo began as a taxi booking service that brought together customers with drivers in urban areas. However, by the end of 2014, it had ceased operations in North America due to heavy competition and a failure to create a strong relationship with New York taxi drivers. Other competitors, like Uber, dominated in more expensive cities and towns, while Hailo was unable to adjust to the market, eventually leading to its collapse.

Hailo revolutionized[4] transportation by providing real-time tracking, safe payment methods, and wheelchair accessibility[2]. The Hailo mobile application was popular in the transportation industry, notably among taxi drivers who supplied transportation to consumers.

DEFINITION OF SCOPE

This project aims to implement the Hailo app by integrating cloud-based technologies, including a cloud database, machine learning capabilities, and analytics tools. The project will also incorporate external services, such as Google Geolocation, to enable seamless navigation and location-based features. The addition of cloud databases and machine learning will enable continuous updates, data analysis, and performance monitoring, ensuring the app's functionality and user experience are optimized.

2 HIGH-LEVEL NON-FUNCTIONAL REQUIREMENTS

2.1 Quality requirement

The intended users for the Hailo app are diversified, including customers, drivers, administrators, data scientists(analytics):

- **Performance**: The application should respond quickly to administrative activities. When connecting with the database through APIs for payments, sign-ups, sign-ins, or password recovery, the application should

listen and respond quickly.

- **Security**: Validations should occur during sign-up, sign-in, and password recovery. Sensitive data should be preserved and encrypted before being saved to a database or other data repository. There should be access controls in place, as well as regular security audits. When unknown situations occur, the system should be capable of responding promptly.
- **Scalability**: The programme should be capable of handling vast quantities of data going through databases, listening to data from the database, updating data, and accommodating an increasing number of users, service requests, and accepting service requests.
- **Reliability**: The system should be reliable and stable, with minimal downtime and no data loss. The system should shut down less frequently, and services should always be available at any time. There should be limited delays when requesting services and no incorrect payments
- **Compatibility**: The programme should be compatible with a variety of mobile operating systems, particularly for IOS and Android.
- **Compliance**: To ensure data protection, the system ought to adhere to the legislation of the country (location) where the services are provided, as well as data privacy regulations
- **User Interface**: The system should be user-friendly and simple to use, with typefaces that may be modified for better reading. The system should be simple to use, available across several devices, and include audio
- **Error Handling**: The system should have the capacity to handle unexpected errors and record them so that maintainers can locate where the error occurred while minimising hours of downtime
- **Maintainability**: The code should be fully documented and written to allow adaptation to subsequent updates and errors, as well as easy navigation. There should be system documentation that serves as a road map for changes and maintenance.
- **Accessibility**: The system should be accessible to all users with impairments by allowing them to modify the system's colours to assist their eyesight and, if possible, resize the text.
- **Localization**: The system should use English as a general language, as well as other spoken languages in the area where it will operate. The system should accept the money of the location in which it operates.

2.2 Quantification of the quality requirement.

Every quality feature needs to be measured with particular metrics in order to make sure the Hailo Taxi Hailing system satisfies its non-functional requirements. The quantitative quality requirements for each of the above criteria are listed below.

- **Performance**:

  Database Response Time - Response times for API calls to the database should be under 200 milliseconds for payments, sign-ups, sign-ins, and password recovery.

  API Response Time - For 95 percent of all queries, the application should react to user actions in 300 milliseconds or less.

- **Security**:

  Encryption - Before being kept in the database, any sensitive data (such as passwords and payment details) must be encrypted using AES-256.

  Authentication - All admin accounts have to use multi-factor authentication. Perform security audits at least once every three months.

  Security Audits - Conduct security audits at least once per quarter.

  Incident Response - Within five minutes, the system need to identify and address security incidents.

- **Scalability**:

  User Capacity - Up to 1,000,000 users simultaneously should not cause performance issues on the system.

  Service Requests - Up to 10,000 service requests should be handled by the system in a second.

  Data Throughput - 500,000 read/write operations per second should be supported by the database.

- **Reliability**:

  Uptime - Less than 52.56 minutes of downtime should occur annually if the system maintains an uptime of 99.99 percent.

  Error Rate - Less than 0.01 percent should be the system's service request error rate.

  Data Loss - By putting strong backup and recovery procedures in place, the system should guarantee that no data is lost.

- **Compatibility**:

  Operating System - The programme should be fully compatible with the latest three versions of iOS and Android.

Device Support - To ensure maximum performance on smartphones and tablets, the application should support a variety of device screen sizes and resolutions.

- **Compliance**:

  Data Privacy - Quarterly compliance checks are necessary to ensure adherence to local, state, and federal data protection laws as well as the GDPR.

  Payment Compliance - For all payment processing, the system must be PCI-DSS compliant.

- **User Interface**:

  User Satisfaction - In user feedback surveys, obtain a user satisfaction score of at least 90 percent. *Font Adjustability* - A user interface that supports font sizes between 12 and 24 pixels should be included.

  Multi-Device Support - The user interface (UI) ought to offer a uniform experience on mobile devices.

- **Error Handling**:

  Error Detection - Within a second of an error occurring, the system ought to identify and record 99 percent of them.

  Minimise Downtime - In most cases, errors may be fixed in 30 minutes or less.

  Error Reporting - By giving maintainers comprehensive error reports, you can cut down on troubleshooting time by half.

- **Maintainability**:

  Documentation Coverage - Ensures that external documentation and inline comments provide 100 percentage code coverage.

  Code Modularity - Aim for a code modularity metric where reusable components make up at least 70 percentage of the codebase.

  Update Cycle - Introduce new features when needed and updates as soon.

- **Accessibility**:

  Colour Customisation - With at least 2 preset themes and support for unique colour schemes, the system should enable users to alter the colours of the user interface.

  Text Resizing - Give users the option to resize text between 12 and 36 pixels.

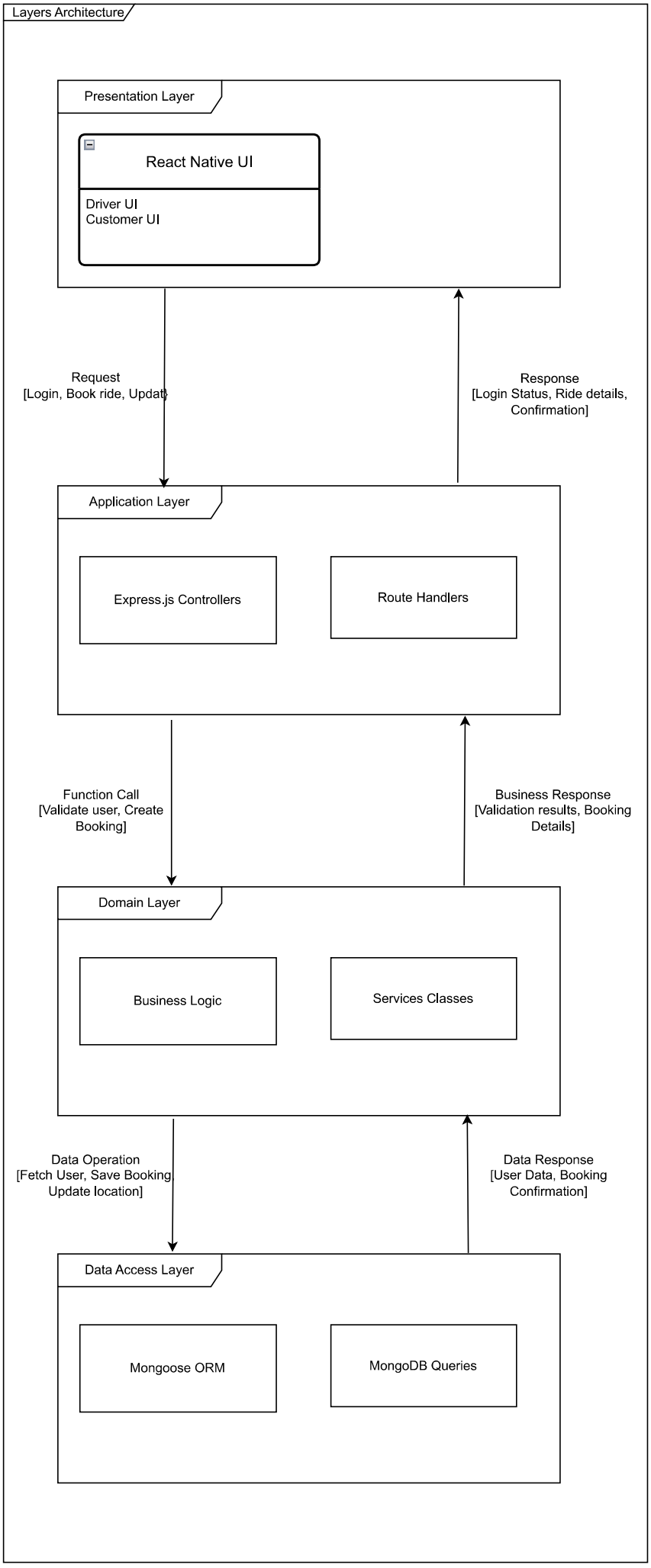
- **Localization**:

  Language Support - The system should support English and at least 2 additional languages spoken in the operating regions.

Currency Handling - The system should handle at least 5 different currencies, depending on the region of operation.

3 ARCHITECTURAL DESIGN

3.1 Architectural patterns


Layers Architecture
Presentation Layer
React Native UI
Driver UI
Customer UI
Request
[Login, Book ride, Updat]
Response
[Login Status, Ride details, Confirmation]
Application Layer
Express.js Controllers
Route Handlers
Function Call
[Validate user, Create Booking]
Business Response
[Validation results, Booking Details]
Domain Layer
Business Logic
Services Classes
Data Operation
[Fetch User, Save Booking Update location]
Data Response
[User Data, Booking Confirmation]
Data Access Layer
Mongoose ORM
MongoDB Queries

3.2 Architectural Styles (tactics)

The Hailo application was built with React Native Expo for mobile UI, Node.js and Express for backend, and MongoDB for data storage. The design uses a variety of components and procedures to provide performance, security, scalability, and maintainability.

Layered Architecture

- **Presentation Layer**: Consists of the React Native UI/Expo applications for both drivers and clients.
- **Application Layer**: Contains the Express.js controllers and route handlers.
- **Domain Layer**: Encapsulates business logic and service classes.
- **Data Access Layer**: Utilizes Mongoose ORM and MongoDB queries for database operations.

Quality Requirements and Tactics 2. Performance

- **Concurrency**: The system will be built with Node.js, a non-blocking, event-driven framework. This enables the system to handle several requests at the same time without interrupting the execution thread, enhancing overall performance.
- **Load Balancing**: A load balancer distributes incoming traffic over numerous server instances, ensuring that no single server becomes a bottleneck, enhancing scalability and stability.

3. Security

- **Authentication and Authorization**: The system employs JWT (JSON Web Tokens) to authenticate users. Role-based access control guarantees that only authorised users have access to specified resources.
- **Encryption**: Sensitive data, such as passwords and payment information, are protected in transit and at rest via SSL/TLS and database encryption, respectively.
- **Audit Trail**: Logs of user actions and system events are kept for auditing purposes, which aids in detecting any unauthorised or suspect activity.

4. Scalability

- **Microservices**: The system can be broken down into microservices for various tasks such as user management, ride booking, and payment processing. This enables autonomous scalability of each service in response to demand.

- **Database Sharding**: MongoDB may use sharding to distribute data across different servers, allowing it to handle enormous datasets more efficiently.

5. Maintainability

- **User Interface (UI)**: The layered architecture's separation of concerns encourages a modular codebase, with each layer responsible for a certain component of the system.
- **Code Reusability**: Service classes and business logic encapsulated in the domain layer can be reused throughout the application, lowering code duplication.

6. Usability

- **Modular Code Structure**: The React Native UI provides a responsive and consistent user experience across devices. Expo offers tools for rapid prototype and testing, which improve the development workflow.
- **Feedback Mechanism**: The rate and feedback system enables users to offer ratings and feedback, which may then be used to improve system and security quality.

3.3 Architectural Constraints

Architectural constraints are limitations or restrictions placed on the design and implementation of a system. These constraints can result from a variety of sources, including hardware limitations, standard compliance, and organisational policies. The architectural restrictions for hailo taxi hailing are as follows:

Hardware Constraints

1. Mobile Devices

1. **Performance Variability**: The system must run smoothly on a wide range of mobile devices with different performance capabilities (CPU, RAM, battery life). The introduction of React Native helps address issue by allowing for optimised cross-platform development, however care must be taken when optimising for low-end devices.
2. **Network Connectivity**: The programme must smoothly handle changing network conditions, such as limited bandwidth and inconsistent connectivity. This necessitates effective data synchronisation and offline support functionality.

2. Servers

1. **Scalability**: The back-end infrastructure must be able to scale horizontally to manage changing demands, particularly during peak usage periods. This entails deploying the system on cloud platforms such as AWS, Azure, or GCP, which include auto-scaling capabilities.

2. **Latency**: Minimising delay is crucial for real-time updates (such as ride status and driver position). Servers deployed in globally spread data centres can assist reduce latency for consumers in various countries.

Software and Standards Constraints

1. Regulatory Compliance

1. **Data Privacy**: The system must adhere to data privacy rules such as POPIA act. This involves the implementation of user permission processes, data anonymization, and safe data storage methods.

2. **Motivation**: Payment systems must adhere to PCI-DSS (Payment Card Industry Data Security Standard) regulations to ensure the security of payment information.

2. Development Standards

1. **Code Quality**: Adherence to coding standards and best practices is critical for maintaining code quality and fostering developer collaboration. Tools like ESLint for JavaScript and Prettier for code formatting can be enforced via continuous integration (CI) pipelines.

2. **API Design**: RESTful API design principles enable consistent, scalable, and maintainable system APIs. This involves standardising HTTP methods (GET, POST, PUT, and DELETE), status codes, and endpoint naming guidelines.

3. Security Standards:

1. **Authentication and Authorization**: The system must employ secure authentication protocols like OAuth 2.0 and guarantee that authorization is role-based, with access control depending on user roles (e.g., admin, driver, customer).

2. **Encryption**: All data in transit must be secured with SSL/TLS, and sensitive data at rest should be encrypted with robust encryption techniques. Compliance with industry standards, such as FIPS 140-2, is recommended.

Organizational Constraints

1. **Time Constraints**: Project timetables and deadlines must be met, which may necessitate prioritising features and implementing agile development techniques. Continuous integration and deployment (CI/CD) pipelines can help shorten development and deployment times.

2. **Budget Constraints**: The development and operations costs must be kept within the specified budget. This influences technological stack selections, cloud service provider selection, and third-party integrations. Cost-effective options, such as open-source tools and frameworks, are chosen.

Technical Constraints

1. **System Integration**: Integration with third-party services, such as payment gateways, mapping services (e.g., Google Maps), and notification systems (e.g., Firebase Cloud Messaging), must be easy to use and dependable. Compliance with the APIs supplied by these services is required to assure compatibility.

3.4 Actor-System Interaction models

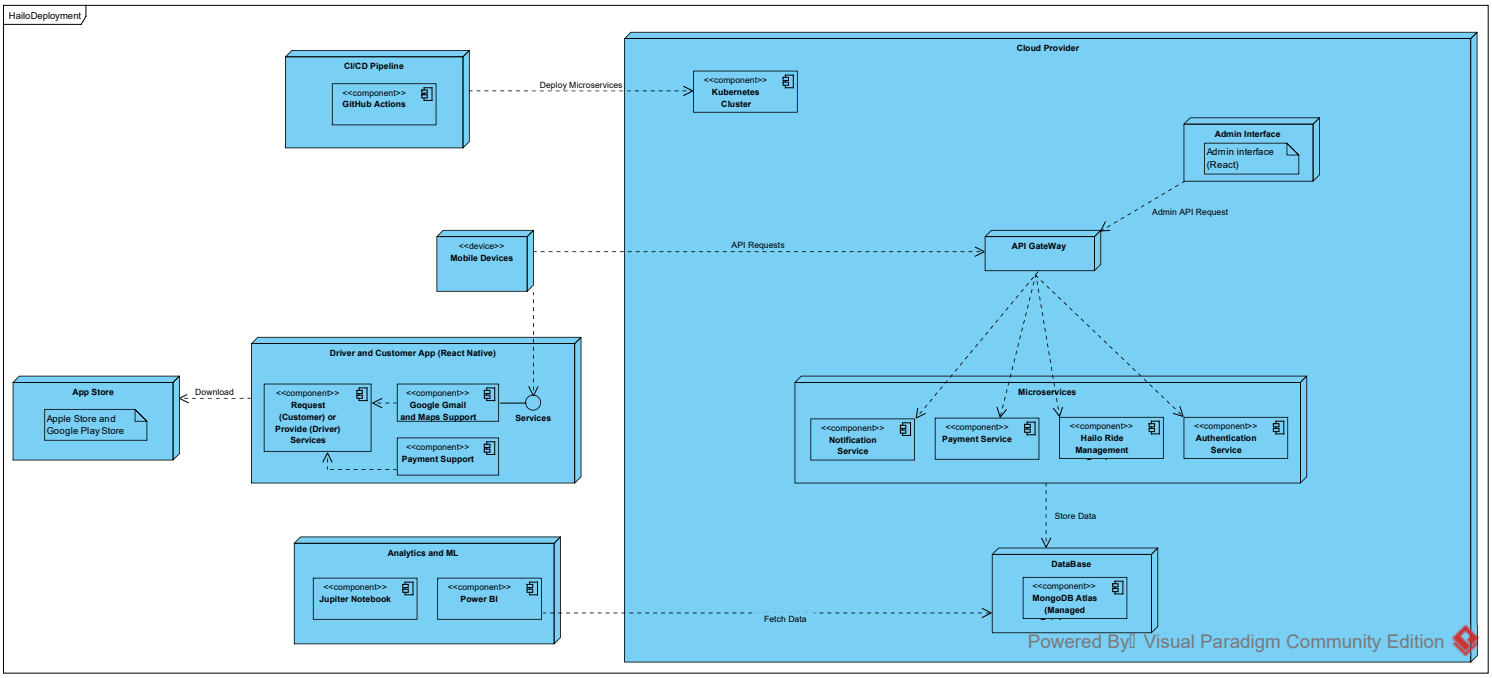
Pre-Condition	The user must have access to the internet before interacting with the system.
Actor	**System**
1. The user opens the Hailo mobile app and clicks on the "sign up" button. 3. User provides the required information by filling and choosing options and also submitting.	2. The system(Hailo mobile app) displays the sign up page for the User to enter their personal details and passwords. 4. System validates information provided and creates a new account for the user. 5. The system sends verification email message (and cellphone message).
Post Condition	The user successfully signs up for the Hailo mobile app.

Pre-Condition	The user must have downloaded and installed the Hailo mobile app and must have a stable internet connection. The user has forgotten the password of the hailo mobile app.
Actor	**System**
1. The user opens the Hailo mobile app and clicks the forgot password link on the log in screen. 3. The user enters their Email address and their Identity number in the text fields and click submit button. 5. The user enters a new password and click submits button.	0. The system (Hailo mobile app) displays the log in screen. 2. The system displays the forgot passwords screen to the user. 4. System authenticates the input credentials whether they exist in the system and match. 6. The system sends a password reset code to the user's email and redirects the user to the page to enter new passwords and a code.
Post Condition	The user successfully reset the password and can sign in on the Hailo mobile app with their new credentials.

Pre-Condition	The user(driver) has logged into the Hailo mobile app and is available to provide services to the customers.
Actor	**System**
1. The driver indicates their availability by clicking on the status button. 3. The Driver receives the ride requests notification from the customer with amount, pick-up location, estimated time of arrival to the customer and to the drop-off location, drop-off location and method of payment and accept request service. 5. The user clicks on the start button once arrived to the customers location and picked the customer. 7. The user clicks on the status button to update their status once delivered the customer.	0. The system (Hailo mobile app) displays the Provide service screen. 2. The system sends notification to the user about service request from the customers. 4. The system displays a map with direction to the customer. 6. The system displays the map with direction from the pick up location to the destination location and change availability status to "Busy". 8. The system displays the map with direction from the pick up location to the destination location.
Post Condition	The driver successfully provides a ride to the customer through the app. The driver updates their status to be available for the system to detect and assign the customers. The driver either receives rates and tips for the service.

Pre-Condition	The user (customer) has logged into the hailo mobile app and has a valid account with payment details added especially for card payment.
Actor	**System**
1. The user (customer) clicks on the "where to go" text field to choose their destination. 3. The user clicks on the right location. 5. The user clicks on the type of service they like and click confirm button. 7. The user can click on whether to reject the driver.	0. The system (Hailo mobile app) displays the request service screen with the map and nearby cars. 2. The system displays current location and various destination, allowing the user to choose the right location. 4. The system displays types of services and amount for each service. 6. The system displays the map with direction from the pick up location to the destination location, the driver's details, amount, time of arrival and kilometers to the location.
Post Condition	The customer successfully requests a ride through the app, and the driver accepts the request. The customer tracks the drivers location to the pick-up location, the ride progress on the way, and to the drop-off location. The customer can rate the driver's service and make tips.

3.5 Deployment Model


HailoDeployment
CI/CD Pipeline
<<component>>
GitHub Actions
Deploy Microservices
Cloud Provider
<<component>>
Kubernetes
Cluster
Admin Interface
Admin interface
(React)
Admin API Request
<<device>>
Mobile Devices
API Requests
API GateWay
Driver and Customer App (React Native)
App Store
Apple Store and
Google Play Store
Download
<<component>>
Request
(Customer) or
Provide (Driver)
Services
<<component>>
Google Gmail
and Maps Support
Services
<<component>>
Payment Support
Microservices
<<component>>
Notification
Service
<<component>>
Payment Service
<<component>>
Hailo Ride
Management
<<component>>
Authentication
Service
Store Data
Analytics and ML
<<component>>
Jupiter Notebook
<<component>>
Power BI
DataBase
<<component>>
MongoDB Atlas
(Managed
Fetch Data
Powered By Visual Paradigm Community Edition

3.6 Technology requirements (technical)

Front-end (Mobile Applications)

1. React Native with Expo

1. **Choice**:

   React Native is a popular framework for developing cross-platform mobile applications (Android and Apple) with JavaScript and React. Expo is a platform and framework that sits on top of React Native, offering tools and services to help developers.

2. **Motivation**:

   Cross-platform Development: The write once, run anywhere capabilities minimises development time and effort.

   Rich Ecosystem: There is a vast community and a plethora of libraries and plugins.

   Ease of Use: Expo streamlines development with features like as hot reloading, simple setup, and managed workflow.

   performance: React Native provides near-native performance, which is suitable for a real-time application such as Hailo.

Back-end

1. Node.js with Express

1. **Choice**:

   Node.js is a JavaScript runtime based on Chrome's V8 engine, whereas Express is a simple and versatile Node.js web application framework.

2. **Motivation**:

   Non-Blocking I/O: Non-blocking I/O: Node.js' event-driven, non-blocking I/O model is efficient and capable of managing concurrent queries.

   Single Language Stack: Single Language Stack: Using JavaScript for both the front-end and backend simplifies development and allows code reuse.

   Scalability: Node.js is ideal for microservices and scalable network applications.

   Rich Ecosystem: Using npm, you can access a wide amount of libraries and middlewares.

Database

1. MongoDB

1. **Choice**:

   MongoDB is a NoSQL database known for its flexibility, scalability, and performance.

2. **Motivation**:

   Schema Flexibility: Document-oriented storage enables flexible schema design, which is useful for changing data models.

   Horizontal Scalability: The built-in support for sharding allows for horizontal scaling, which can accommodate big datasets and high traffic.

   Performance: Capable of managing massive amounts of unstructured data while performing quick read/write operations.

   Integration: Seamless integration with Node.js via the Mongoose ORM.

Payment Processing

1. Stripe API

1. **Choice**:

   Stripe is a comprehensive payment processing platform that provides APIs for handling various payment methods.

2. **Motivation**:

   Security: PCI-DSS compliant, ensuring secure handling of payment information.

   Ease of Integration: Well-documented APIs and SDKs for easy integration into the application.

   Global Reach: Supports multiple currencies and payment methods, facilitating international transactions.

Data Analytics

1. Power BI

1. **Choice**:

   Power BI is a business analytics service by Microsoft that provides interactive visualizations and business intelligence capabilities.

2. **Motivation**:

   Visualization: Effective tools for producing thorough reports and dashboards.

Integration: Simple integration with a variety of data sources, including MongoDB.

Collaboration: Makes it easier to share thoughts and work together within the organisation.

2. Jupyter Notebook

1. **Choice**:

   Jupyter Notebook is an open-source web application that allows for creating and sharing documents containing live code, equations, visualizations, and narrative text.

2. **Motivation**:

   Interactive Data Analysis: Suitable for data exploration, visualisation, and machine learning experiments.

   Integration: Supports a variety of computer languages, including Python, which is commonly used in data science.

Bibliography

[1] K. Dunken. A mobile app that makes hailing a taxi easier and more efficient. https://www.entrepreneur.com/science-technology/a-mobile-app-that-makes-hailing-a-taxi-easier-and-more/226684, 2013. Accessed: Apr. 04, 2024.

[2] Hailo. "taxi heaven for london's black cab drivers". https://web.archive.org/web/20120104040211/http://hailocab.com/press-releases/taxi-heaven-for-londons-black-cab-drivers/, 2011. Accessed: Apr. 04, 2024.

[3] J. Kasperkevic. "hailo: the taxi app that gets people off the streets". https://www.theguardian.com/business/2013/dec/15/hailo-taxi-hailing-app-global-growth, 2017. Accessed: Apr. 04, 2024.

[4] K. Meghang. "hailo confirms its 30m round from richard branson, union square ventures". https://venturebeat.com/entrepreneur/hailo-funding-nyc/, 2013. Accessed: Apr. 04, 2024.

[5] P. Pessok. "perh(apps) this is the answer". https://web.archive.org/web/20111013063423/http://hailocab.com/docs/Hailo_Taxi_June2011.pdf, 2011. Accessed: Apr. 04, 2024.

[6] M. Warman. "hailo android app review". https://www.telegraph.co.uk/technology/mobile-app-reviews/9041116/Hailo-Android-app-review.html, 2012. Accessed: Apr. 04, 2024.