Cloud-Based Taxi Booking Service

Norah Saleh O Alsulami

Abstract: The last few decades witnessed the historical evolution in terms of technological advancements and their applications in real life. A taxi booking service is one of the areas where technological advancement has impacted the social fabric beyond just customers and drivers. With the intervention of the sharing economy in the taxi business, it has opened the doors for further advancements and improvements in multi-dimensions. In this dissertation, we have attempted to cover a few aspects of the impact of technological evolution on the taxi service business. Furthermore, we proposed a design for the project for a taxi service. We implemented the project accordingly and then evaluated. We concluded our experience so far from the conceptualization of the project, designing, developing, and evaluating the project. We further proposed our future work in terms of improving and extending the project.

Index Terms: Booking Services, Cloud based, Customer app, Driver app, Transportation, Taxi Service.

1. INTRODUCTION

Ttransportation has been one of the crucial necessities of humankind over the years. Different countries, states and societies have addressed the issue in multiple diverse styles. Such methods may be classified into two broad approaches, i.e., Public and Private transports. Governments typically subsidies public transport systems, whereas private transport systems mostly complement public transport systems. Therefore, both systems have their significance for a vibrant and progressive society. This project intends to build an app on top of such technologies using state of the art tools and libraries available. Car taxi services are one of the vital components of Private transportation systems. Like other modes of transports, taxi services have also evolved over the years. The last couple of decades witness radical changes in taxi services. To book a taxi, initially, a phone call was made to a taxi service. The remarkable usage of smartphones and the internet has replaced a phone call with modern tools and technologies. Now customers can share a ride using a website or a mobile app, track their cabs using real time maps and make cashless payments via different payment gateways. Several taxi services are operating across the globe independently. Even a small city, particularly a tourist-attracted town, may have many operational taxi services. Their software may have used different programming languages, set of technologies and tools, but the central idea remains the same. However, newer technologies have improved the functional capabilities of such software and applications. That is a sure sign of the evolution of such applications. This project intends to stand tall while enhancing the application infrastructure by using newer technologies, introducing modern features, better user experience and taking account of scalability of an application.

2 BACKGROUND STUDY

2.1 Overview

This section gives an overview of how taxi services have evolved over a few decades. Taxi services used to operate on an individual basis in the beginning. With the evolving mechanisms of communications, for example, landline and mobile telephones, they started to work as small-level companies, where a centralized office would dispatch their taxis on receiving requests via phone calls. The evolution of internet-cloud based applications then started to capture a broad spectrum of businesses. So, for a taxi business, a centralized office for receiving the calls and dispatching taxis got redundant in the presence of cloud-based applications. Eventually, the world witnessed the sharing economy phenomenon in the form of Uber and Careem [1]. Such innovative initiatives have boosted taxi-services in terms of better and transparent competition. However, the emergence of such sharing-economy platforms has brought more centralization to the process where peer-drivers are unknown to each other and also has affected small independent taxi services, for example in [2]. Such small independent taxi services are equally important for the ecosystem, especially where giant companies have less influence and reach. Traditionally, a taxi-service has been operating with a limited number of drivers along with a central office to receive phone-calls for ride-requests. This process consisted of at least a person managing the calls in an office, commonly known as a dispatcher. The core responsibility of a dispatcher is to receive a taxi request and fulfil it accordingly. However, as a company grows from a single taxi driver to more, the process gets more complicated. To understand the complexities of the process as a software engineer, it requires a deep understanding of the taxi business. Therefore, it is needed to look into some of the existing popular cloud-based taxi services applications, their infrastructure, approach and solutions, for better understanding of the automation of ride-sharing cloud-based services, for instance [3].

2.2 Comparison and analysis of different applications

Uber has shown a drastic impact on taxi services. With the competitive and dynamic fare mechanism developed and introduced by Uber, they have attracted most of the regular taxi drivers and taken them on-board. Uber's presence has gradually reduced demand for traditional taxis, as reported by [4] and also shown in the graph below.

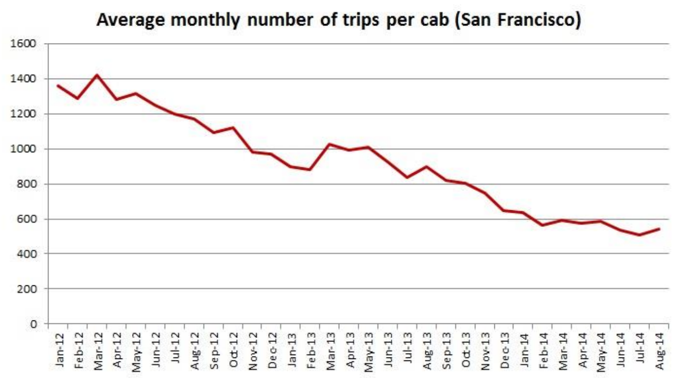

***Fig. 1Average** monthly number of trips per cab (San Francisco) *Source [4].*

However, some research articles have also opined about the destruction of traditional taxi services by sharing economy model adopted by different companies [5]. Whereas other articles like [6] have opined about how traditional firms must compete in the sharing economies, to keep the balance.

3 Requirements

3.1 High-level requirements

3.1.1 Admin Panel

The role of an admin is to manage the system and its appropriate settings. An admin can perform these operations.

1. Manage drivers (Create/Update).
2. View ride bookings (View Details/View payments).

3.1.2 Technical requirements

1. Website for company.
2. Mobile apps for both Drivers and Customers.
3. Development and management of a set of APIs to communicate with websites and mobile apps.
4. Implementation of a messaging queue system to send messages to and from web/mobile apps and server.
5. There should not be any glitches and hindrances between server and its clients (e.g., website, admin panel and mobile apps). The system requires its clients to work in real-time efficiently. Therefore, the connection and communication between the server and its clients should remain one of the top priorities of the project.

3.1.3 Essential requirements

These requirements are necessary for achieving the objectives of this project.

1. Liberty and autonomy of drivers: Using this system, the drivers of the taxi service should have the liberty to choose their timings, working hours independently. Drivers should accept/reject the ride instead of the taxi service forcing on drivers.
2. Implementing an efficient algorithm to find and select a set of suitable active drivers in the required area to notify them as the taxi service receive a ride notification. The system needs to be an efficient platform which operates between customers, drivers, and the taxi service in real-time without any lag.
3. The website and mobile applications should comply with the modern-day requirements so that they can work effectively on all set of operational mobile phones of same operating system.

3.1.4 Recommended requirements

1. Learning and implementing the newer concepts and technologies to be used in this project regarding the use of microservices and how they communicate to each other.

2. Designing and developing a website for administration purpose to setup the system and register drivers. The website will also manage the various aspects of the system, for example, viewing customers data, their profiles, trip history etc. and viewing analytical tools to support business decisions and insights, based on the earnings and payments given to drivers.

3.1.5 Optional requirements

1. Consuming a messaging queue which is used to send messages to the microservice APIs to perform tasks that would normally be directly used to call APIs which results in processing load.

4 Design

As following architecture diagram shows, this project has three main components: frontend, APIs, and Database. The frontend and APIs are communicating via a Messaging Queue, whereas the APIs are responsible for communicating with the database directly. This section will describe these three major components of the project

4.1 Frontend

There were three leading roles identified for the application, intended to interact with the system. These roles will have their interfaces (graphical as well) through which they communicate with the application. Such graphical interfaces are divided into three sub-components depending upon their usage. An interface for drivers, an interface for customers and an

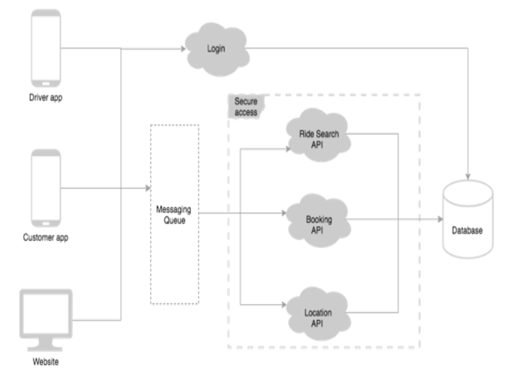

Fig. 1 General Architecture

interface for service managers/owners are these three sub-components of Frontend. The sub-components will have respective functionalities according to the defined roles added to communicate with each other through the rest of the system. As shown in **Error! Reference source not found.**, a driver has an interface to interact with the system through a mobile app. A driver registered with the system has to login into the system before interacting with the system. After a successful login, a driver's app can receive a notification from the system upon when Ride Search API looks for available drivers within the vicinity of a booked ride. Upon receiving a notification, a driver may accept the ride or cancel it. The allocation of rides works upon the first come, first-serve basis. So, a driver who accepts the ride first will receive further information of the customer, for example, their location etc. On successful completion of the ride, a driver will mark the ride as completed through its app, and thus backend system updates in the database. Similarly, a customer interacts with the system through the customers' mobile app. The workflow follows a similar track by accessing features of backend through login into the system. A registered customer will log in into the system through the mobile app before they could book a ride. Upon booking a ride, the backend will formulate a set of available drivers to send notifications of a new ride. A customer can also see their profile and history of trips in their mobile app. Now finally, a user with the managerial access role will have a website interface to interact with the backend system. The login component of the system keeps distinctive information of customers and managers. A user with managerial access can interact with the system and perform various tasks related to customers, drivers, and rides.

4.2 Messaging Queue

This component is critical to the application as it is responsible for acting as a bridge between the frontend and the rest of the system. This component is added to the system to avoid congestion at the server end. This queue will ensure all the communication between components. The subsequent requests generated by different components should reach their destination and should not get lost for any reasons.

4.3 APIs

The APIs are responsible for performing specific tasks delegated to them. Based on the CRUD (Create Read Update Delete) model, their primary usage is to read, write and update data to the database. The brief description of these APIs is below.

4.3.1 Auth/Login website

Auth is a short form for Authentication and is responsible for ensuring the security of the backend of the system. A set of functions are performed by this API, including login existing drivers, customers and ensuring that the driver cannot login to customer app and customer cannot login to driver app as both of the apps have their own logins, and also authorizing for executing specific tasks allowed to particular roles defined by the system. As the **Error! Reference source not found.** shows, three other APIs are secured; therefore, only authorized users can access these APIs. One of the core responsibilities of Login API is to ensure only authenticated and authorized pass through it for accessing other APIs. Auth API is directly communicating with the database of the system.

4.3.2 Ride Search API

Once a customer requests for a ride, it invokes the ride search API. This API receives a set of input from the customer's mobile app and then decides to send a notification to a set of available drivers. This search will use different parameters received from the customer's mobile app and from the booking API and Location API. This API interacts with other APIs in the system to get relevant information, and then process that information to search for rides and available drivers.

4.3.3 Booking API

As explained before, APIs typically follow CRUD mechanism to interact with a database; thus, this Booking API enables actors to perform a set of actions including creating a ride booking, complete a ride booking, cancel a ride booking from the system and submit rating for the ride booking.

4.3.4 Location API

This API is responsible for managing the location streams coming from customers and drivers to keep the system updated. This API receives the coordinates of both customers and drivers, sent via their respective mobile applications. The location API plays a vital role in providing location parameters to Ride Search API, which helps it in finding a list of available drivers within the vicinities of those locations.

5 Testing

5.1 Registration and Login

The functionality testing of the registration and the login process starts with registering a driver and a customer with the system. We evaluated this by logging into both customers' and drivers' apps. On the admin panel, we could see the newly registered customer and driver in real-time.

5.2 Book a Ride

We evaluated the project through registering and login with a customer's credentials and booked a ride. On another phone, we logged into the driver's app with a driver's credentials. In the customer app, we gave our postcodes for our pickup point and destination as an input. As we clicked the book ride button, we noticed there was a notification of a ride in the driver's app. Since we were on-job as a driver, we accepted the ride, and it updated on both customer and driver's apps.

5.3 Accept a Job

After when we accepted the job on driver's app, we went to the actual location we entered as the destination. On driver's app, we clicked finished job button, and it showed the job details, both on driver and customer's apps. In this way, we evaluated that both apps and services work efficiently in real-time and respond to each other without any significant delays.

5.4 Ride History

On our both apps (driver and customer) - we checked ride history, and both apps had consistent and same data updated. This consistency shows our both apps and services are in line with syncing and updating their data in real-time.

6 FUTURE WORK

These are the areas I would like to put in more time and efforts for this project in future.

6.1 User Interface
Since both apps and the admin panel have basic or default user interfaces provided by Xamarin forms, I would like to extend both interfaces in terms of using more graphics, images, layouts, and improved user experience.

6.2 Feedback portal
Feedback from both customers and drivers will help in improving a taxi service; therefore, in future, I would like to extend this application to have a portal where customers and drivers could share their feedback with the taxi service provider.

6.3 Advanced level development and deployment
With the actual deployment of the project in a real-life scenario, it will enable more opportunities to look into advanced level development and deployment of the application, for example, its scalability, if there is a large number of users and traffic.

7 CONCLUSIONS
We designed, developed, deployed, and tested the three different clients of the project, namely Customer app, Driver app and the Admin panel. We evaluated this project by focusing on two main things, the functionality and responsiveness of the apps in real-time. We conclude our testing and evaluation by presenting a few testing scenarios for the developed apps.

REFERENCES

[1] S. a. S. L. Cannon, “How Uber and the sharing economy can win over regulators.,” 2014.

[2] L. L. T. W. J. a. Y. D. Ma, “The impact of E-hailing competition on the urban taxi ecosystem and governance strategy from a rent-seeking perspective: The China E-hailing platform.,” Journal of Open Innovation: Technology, Market, and Complexity, vol. 4(3), p. 35, 2018.

[3] D. S. L. L. B. C. C. P. G. L. S. a. W. Z. Zhang, “Understanding taxi service strategies from taxi GPS traces.,” IEEE Transactions on Intelligent Transportation Systems, vol. 16(1), pp. 123-135, 2014.

[4] S. Golovin, “The economics of Uber,” 2014. [Online]. Available: https://www.bruegel.org/2014/09/the-economics-of-uber/. [Accessed 3 8 2020].

[5] C. B. J.-D. L. Kibum Kim, “Creative destruction of the sharing economy in action: The case of Uber,” Transportation Research Part A: Policy and Practice, vol. 110, pp. 118-127, 2018.

[6] M. A. Cusumano, “How traditional firms must compete in the sharing economy,” Communications of the ACM, 2014