High level Architecture

P07 : Hitcher

Student ID	Name
23100337	Irtza Tariq
22100021	M Muneeb Arshad
23100040	Mustafa Afzal
23100333	Nashrah Shaukat

Table of Contents

1. Introduction

<Give an overview of the project here. The overview must highlight the overall objectives of the project and its potential users and customers.>

The project hitcher has been created keeping in mind the global fuel hike. Social media and general surveys suggest that people prefer carpooling over long distances in order to save travel cost. A cab hailed from an application may cost a customer over PKR 1000 for a distance of more than 25 kilometers. This application aims to reduce that cost by accommodating multiple customers looking to commute across the same route.

The niche customer base for this project comprises of people who commute over long distances everyday and wish to save on cost. According to financial classes, this application aims to accommodate lower and middle class customers. A survey at LUMS showed that people who commute from Bahria Town and Johar Town prefer a carpooling service due to the high cost of commute.

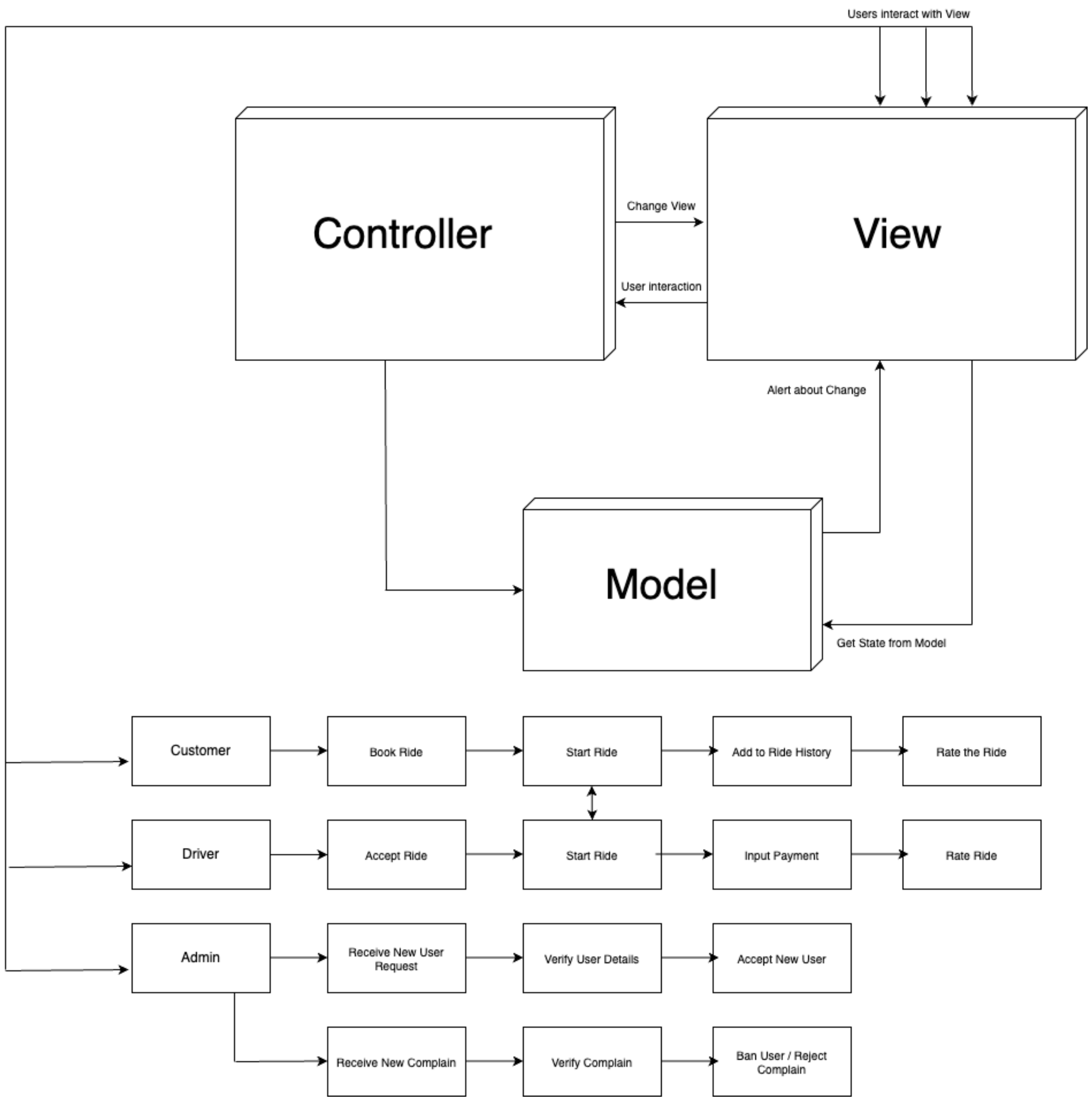
Apart from the major objective, the application serves to provide relief to the drivers and customers in a plethora of other forms. The application aims to provide drivers an incentive to drive more by providing them with 100% of the receipts from each ride after a certain threshold of rides per day has been attained. This was brought up by drivers who disliked the fact that they had to pay a cut to the company for every ride that they completed. This also brought a culture of drivers canceling rides and completing rides unofficially.

This culture brought a problem of security for customers as they could not be tracked by the ride hailing companies and in turn could not complain if something went wrong. To bring end to end security, we introduced a feature of QR codes. The rider must scan the QR code on the driver’s phone to start a ride that would act as an acknowledgement to start a ride and would provide a timestamp for accurate fare calculations.

In conclusion, our application aims to overcome the problems faced by drivers and customers who use other ride hailing services and we believe that this will provide a more satisfactory service.

2. System Architecture

2.1 Architecture Diagram



2.2 Architecture Description

<Give description of each subsystem in the architecture diagram above. Moreover, give description of how subsystems interact with each other>
The architecture is based on Model View Controller architecture as well as Pipe and Filter architecture. The three unique actors in the system will be interacting with the view module to view the current state of the application.

For customers, this will be the position of drivers on the map, driver's details and other information related to their ride to a specific destination. For drivers, the information can consist of details of the customer that they will be riding with. Both the drivers and users will be able to view the ratings associated with the actors they will be traveling with. The admin will be able to browse through complaints, location of drivers and requests for new accounts.

When any actor chooses to make any changes to the state of the application, the view module will reflect this interaction to the controller module. As a result, the controller is able to inform the View module and the Model module to change what they are displaying and change their state respectively.

The Model module is where the state of the application is updated. It also receives requests from the View module to render data and the Model responds with data that can be rendered.

Some specific actions of the actors are associated with the Pipe and Filter architecture. This architecture is used for a sequences of actions where the input of each action depends on the output of the subsequent action.

For the customer, the entire process of riding is broken down into further subsystems. The customer must first book a ride and then start the ride by acknowledging the driver's request to start a ride. By the end of the ride, the ride must be added to the ride history of the customer and the customer will only be able to rate the driver at the end of the ride.

For the driver, they must accept a ride and request to start a ride that is acknowledged by the customer. By the end of the ride, the driver must input the amount collected into the system. The driver may only be able to rate a customer by the end of the ride which completes the process of taking a ride.

For the admin, they will either receive a complaint or the request to make a new account. In both the scenarios, they must verify the authenticity of the new account or the complaint and then proceed accordingly.

2.3 Justification of the Architecture

<List down pros and cons of the architecture you have defined in the context of your system. Moreover, give a justification of why this architecture is appropriate for your system. Make sure that you also discuss how this architecture helps in implementation of non-functional requirements. >

Our application uses two architecture patterns namely Model View Controller and Pipe and Filter architecture. The advantages and disadvantages of the above mentioned architectures are listed below

Model View Controller
This architecture separates modules into view, model and controller. This allows developers to distribute their resources to focus on each resource. Due to this architecture, a user's rendered data does not clash with the data updates in the database. The asynchronous model allows data to render quickly as views are independent of changes being made to the database. The controller acts as a low level module that updates the state of the application and updates the view of the user. Test-driven development becomes much simpler as the application is divided into segments that would render data, respond to user interaction and change the state of the application. Making changes to the data rendered becomes simplified as only one module handles changes to data rendering. Overall, the architecture provides a modular approach to development.

While this architecture has its advantages, some developers do not prefer it. Independent testing of components may have been an advantage, the collective testing of components becomes difficult in MVC. Scalability may also become an issue as the entire application must be scaled and it is not possible to scale specific components.

We believe that this architecture is appropriate for our development process as it will speed up development and it will make debugging simpler and limited to a certain component of the application. Using MVC would also make our code reusable and easier to understand for other developers in the future as future edits in rendering would not affect the logic of the application. Provided that our application has a complex backend logic and frontend rendering using APIs of maps, it is optimal to work on both components independently. Rendering times for the application will significantly decrease with an independent view module. Response times for the users would significantly decrease as an independent module for reading and understanding user interactions would result in a faster fetch or put operation in terms of the database. It will help user interaction to avoid getting stuck in a pipeline of operations. Such a problem could occur if user interactions, updating the database and rendering views would be the job of a single module.

Pipe and Filter

This architecture sets up a linear route for data to flow that would be filtered along the path and would produce a final output in the end called the sink. This makes it extremely easy to add processes in the future. This makes debugging simpler as developers can check which filter produces inaccurate data and that can be fixed. It treats subsystems as functions where inputs and outputs play a crucial role.

This architecture does make the development process simpler and easy to understand as compared to the Model View Controller approach but it has a downside. It can create performance issues if enough filters are added to the pipeline. That would require more processing hence more time. This can affect the usability of the application and users would face delays.

This architecture helps us breakdown some processes in the application into subsystems as they depend on each other. For example, the process of banning a user depends on receiving a complaint that would be verified for authenticity. Users cannot rate other users unless they have ridden together and that is only possible at the end of the ride. Such subsystems become simpler to implement and understand with this architecture. We have made sure that any complex rendering or calculation is not a part of the pipeline or it would severely slow down the application. Our architecture will make sure that processes related to each other are executed in a certain order in order to avoid wrong processing or rendering bad information.

3. Risk Management

3.1 Potential Risks and Mitigation Strategies

Sr.	Risk Description	Mitigation Strategy
1.	The size of the software and time required to finish is underestimated and now there is a time crunch to finish it.	Document requirements properly to have an idea of the size of the project and also at the beginning overestimate the time required to finish the project so that you have wiggle room at the end. Create a Gantt Chart so that people know about dependencies between different sections and know about any delays and also have a clear view of the timeline.
2.	There is a lack of clarity among members of the group.	Check and recheck your requirements before beginning the development and make sure everyone is on the same page before the development phase.
3.	Group members can get ill and the development can come to a stop.	Make sure that everyone is on the same page so that if one person cannot work on their part, another group member can take over their work until they get better.
4.	The server used for the deployment of the product may be unavailable due to unforeseen circumstances.	Deploy the application on a backup server from another provider so that your application is never down.
5.	Unresolved conflicts among different sections of the application not resolved in a timely manner.	Hold regular meetings and look for such conflicts and resolve them as soon as you get aware of them.

6.	Inadequate testing leads to the deployment of an application riddled with errors.	Ensure that proper test cases are used and each and every possible scenario is tested before deployment.
7.	Resource unavailability (e.g internet) due to political unrest in the country.	No ability to mitigate this risk.
8.	The skill required to finish a particular task is insufficient.	Outsource that particular task or use premade code/product in place of that specific task.
9.	Hardware requirements needed for proper deployment of the application are coming up short and are risking a timely deployment.	Try to invest in a better server from a better provider and ensure they meet your application's minimum required specifications to operate properly.
10.	Requirements change as the product is developed and needs are identified. Changes are larger than anticipated.	Patches and fixes can be made to cater to smaller issues. Larger changes must be dealt with by initially creating code that is easy to maintain and change.

4. Tools and Technologies

The project will consist of two applications. One will be a mobile application for the customers and the drivers and the other would be a web application suitable for the administrators. The following tools will be used for development purposes.

For the mobile application we would be using flutter for front-end and firebase for backend.

Flutter:

- Version 3.3.4
- Flutter is a free and open-source mobile UI framework created by google for mobile app development. Since it is easy to learn and use, we will be using it for the front end.

Firebase:

- Latest release
- Fire base is a google backed application development software which will be used as the back-end for the application. It boasts a real-time database along with multi-platform authentication.

For the web application we would be using MongoDB for the back and React JS, Express JS and Node JS for the front end.

Express JS:

- Version 4.18.2
- Express is a node js web application framework that provides broad features for building web and mobile applications

React JS:

- Version 18.2.0
- The React. js framework is an open-source JavaScript framework. It's used for building interactive user interfaces and web applications quickly and efficiently with significantly less code than you would with vanilla JavaScript

Node JS:

- Version 16.9.0
- Node. js lets developers use JavaScript to write command line tools and for server-side scripting

MongoDB:

- Version 6.0
- MongoDB is a scalable, distributed database which stores data in flexible, JSON-like documents.

For both web and mobile applications we would be using Selenium for testing purposes.

Selenium:

- Version 4.5.0

- Selenium is a free (open-source) automated testing framework used to validate web applications across different browsers and platforms.

For deployment purposes we will be using AWS.

AWS:
- Latest Version
- AWS CodeDeploy is a fully managed deployment service that automates software deployments and will be used for the deployment of both web and mobile applications
-

Throughout the project a project management tool, Trello will be used.

Trello:
- Latest version
- Helps manage the project, workflow and with task management among the team.

5. Hardware Requirements

Deployment Server:

Firebase:

Our flutter app must target these versions;
- IOS 11 or later
- MacOS 10.13 or later
- Android 4.4 or later

AWS:
- 16 GB RAM minimum, 32 GB RAM recommended

- 400 GB available disk space minimum, 500 GB recommended
- CPU cores 6 minimum, 32 recommended
- Processor speed: 2.3 GHz or more minimum and recommended

Development Machines:

Flutter:
- Microsoft® Windows® 7/8/10 (64-bit), Mac® OS X® 10.10 (Yosemite) or higher
- 4 GB RAM minimum, 8 GB RAM recommended
- 2 GB of available disk space minimum,
- 4 GB Recommended (500 MB for IDE + 1.5 GB for Android SDK and emulator system image)
- 1280 x 800 minimum screen resolution

ReactJS:
- Windows XP, Windows 7 (32/64 bit) or higher, Mac® OS X® 10.10 (Yosemite) or higher
- 4 GB RAM minimum
- 10 GB available disk space
- At least one Internet Browser e.g. Chrome, Firefox, Microsoft Edge etc.
- Node.js
- Active internet connection minimum speed 512kbps and above.
- At least one installed code Editor to test and debug your code (eg. VScode, Sublime)

6. Who Did What?

Name of the Team Member	Tasks done
Irtza Tariq	Architecture Justification, Architecture Diagram
Ebad ur Rehman	Architecture Description,Introduction

M Muneeb Arshad	Risk Management
Mustafa Afzal	Risk Management, Hardware Requirements
Nasrah Shaukat	Tools and Technologies

7. Review checklist

Before submission of this deliverable, the team must perform an internal review. Each team member will review one or more sections of the deliverable.

Section Title	**Reviewer Name(s)**
3.Risk Management	Nashrah Shaukat
4. Tools and Technologies	M Muneeb Arshad
2. System Architecture	M Muneeb Arshad
5. Hardware Requirements	Ebad ur Rehman
1.Introduction	Irtza Tariq