Design Document

Software Engineering Course Project

Team Number 3

Team members:
Pratik Mandlecha
Rachit Jain
Raghav Mittal
Runa Chand
Savita Bhat
Shivang Shekhar

Purpose

The on-demand economy has transformed the way businesses cater to their customers all across the globe. On one side, it is extremely convenient and hassle-free for customers as well as businesses to provide for customers' requests. Whereas on the other side, even the smallest calamity like flooding can adversely affect the business and respective supply-demand equation. As a part of the SWE course project, we focus on on-demand business and propose novel usage of external data points to mitigate these disaster scenarios. We choose a cab-hailing service as an example of on-demand business along with the CORONA virus spread as a national calamity to demonstrate how external inputs can ensure uninterrupted services.

We document our architecture, design choices and technology survey in this report. This report is to be used as a guideline in further development and implementation efforts. We describe our architecture design along with a class diagram, detailed design for different components and modules in the following sections. We plan to improve this document to incorporate our learnings and findings during implementation.

Overview

We have decided to bank on the MVCS design pattern for our application, the design document consists MVCS class diagram which specifically differentiates between various services and controllers available.

This Software Design Specification also includes:
A *system architecture description* is a detailed description of the components. (including a template description for each component) and their reuse and relationships with other features. Design decisions and tradeoffs involved with respect to the technologies involved.

We have tried that the design has been made clear, using the use case diagram and class diagram. The structure of our project is highly modularized. We have tried introducing as much functional cohesion as possible. For coupling, we have tried to achieve loose coupling.

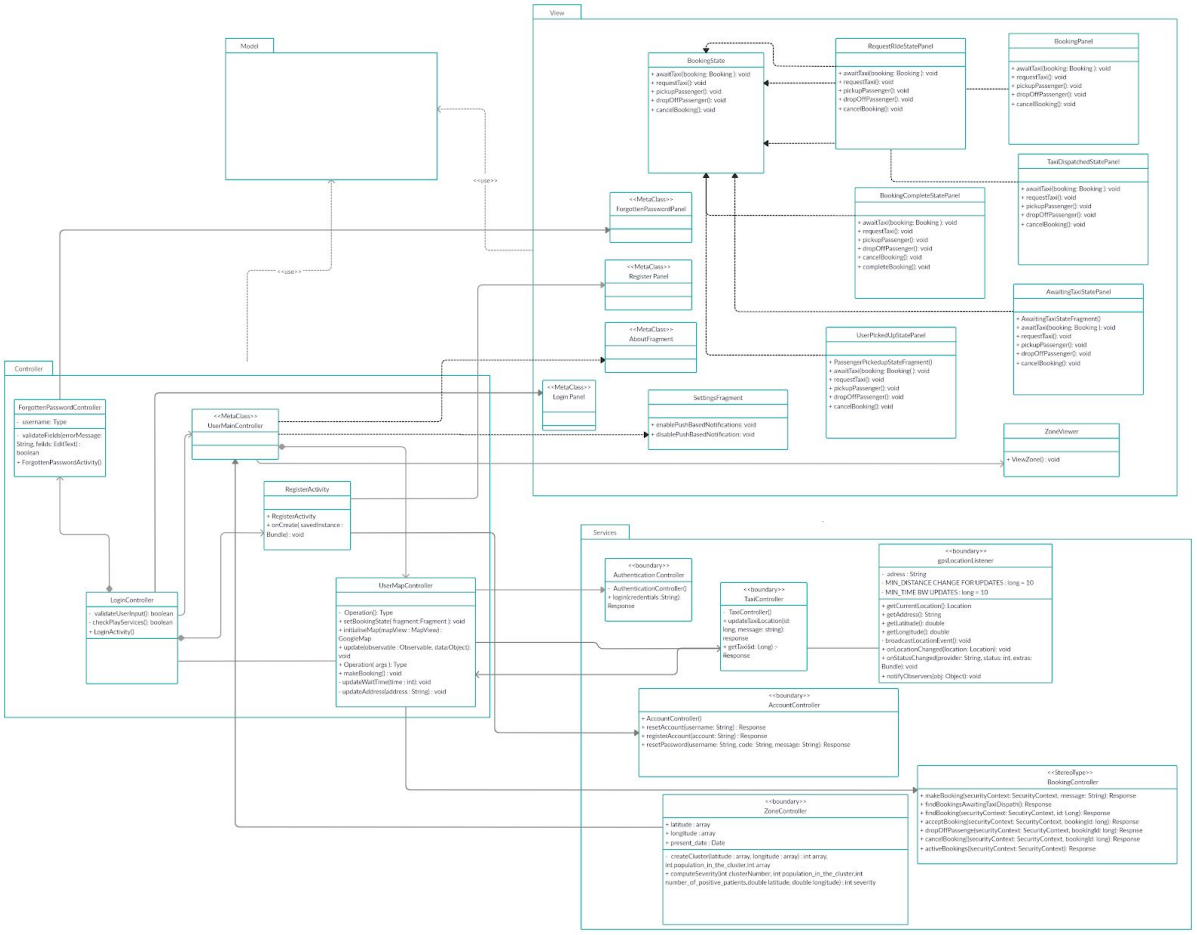
Terminology

1. System: The package of all the components which takes input and gives output to demonstrate the features of the software.
2. User: Any living being who is interacting with the *System*.
3. Driver: A *User* who drives a cab.
4. Passenger: A *User* who is willing to travel between two locations.
5. Zone: A demarcated geographical area to be used to assign threat levels based on the severity of the outbreak.
6. Threat level: A numeric representation attached for each *Zone* to signify the chances of a *User* getting infected while traveling through that *Zone.*

Use-Case Diagram

Link for High-res Diagram: https://tinyurl.com/swe-usecase

Class Diagram



Link for High-res Diagram: **https://tinyurl.com/swe-diagram**

Detailed Design of Components

Class : UserMapController

Method	Input	Output	Description
setBookingState	fragment: Fragment Object *fragment* subclass.	void	Controller for managing booking state.
initialiseMap	mapView: MapView Object of mapView	GoogleMap GoogleMap object	Initialises the google map

	subclass		
makeBooking		void	
updateWaitTime	Integer: time time for which a Passenger has to wait for the cab.	void	Update the time for which a passenger has to wait for the cab to arrive.
updateAddress	String *:* address Address of *User*	void	

Class : TaxiController

Method	**Input**	**Output**	**Description**
updateTaxiLocation	Long: id For identifying the cab String : message	response	Updates the taxi location based on its current location
getTaxi	Long : id	response	

Class : LoginController

Method	**Input**	**Output**	**Description**
validateUserInput		boolean	Validates user details
checkPlayServices		boolean	Checks if plays service are active
LoginActivity		void	Logs user login activity

Class : AccountController

Method	**Input**	**Output**	**Description**
AccountController()		void	Maintains async account activity
resetAccount	String: username	void	reset

registerAccount	String: account	void	register
resetPassword	String: username, String: code, String: message	void	resetPass

Class : BookingController

Method	**Input**	**Output**	**Description**
makeBooking	String : Message	void	Method for booking a cab
findBookingsAwaitingTaxiDispatch		void	Find users who are searching for a cab.
findBooking	Long : id	void	Find all requests for cab booking.
acceptBooking	bookingId: long	void	Accept a booking
dropOffPassenger	Long : bookingId	void	Complete a trip.
cancelBooking	Long : bookingId	void	Cancel an incoming booking

Class : ZoneController

Method	**Input**	**Output**	**Description**

createClusters	Double arrays of latitude and longitude of the locations	Integer array: cluster number assigned for every location, Integer: population_in_the_cluster, int array: number of positive patients in each cluster,	Create clusters based on locations of data points of patients. We will get clusters based on the density of points.
computeSeverity	Integer: clusterNumber, Integer: population_in_the_cluster, Integer: number_of_positive_patients_in_the_cluster, Double: latitude, Double: longitude,	Integer: severity index for a location in the city	This function gives the severity index or threat level for a given location in the city using the density-based clusters and other relevant data.

Constraints

Here is a comprehensive list of constraints that are required to be pro-quo in the functioning of the app/web app.

1. Functional Constraints
    a. The app built in the project will not provide real-time access to disaster situations but uses a pre-trained model that alters the linear decision -making algorithm to make feasible changes to the rider's path.
    b. Network stability is not a factor considered during the project. Offline mode is not offered.
    c. The decision-making model runs on pre-existing data with an acceptable accuracy but does not ensure similar accuracy across all the locations.
    d. The design of the system is such that it can be horizontally scaled but will be tested under constrained conditions only which will be further discussed in the test plan sheet.
    e. Performance issues and constraints are not a part of this document and will be discussed later in the detailed review.

2. Resource Constraints:
    a. A primary resource constraint as discussed above in the functional part is the resource estimation/allocation for the nth user, which can be only discussed in the design document but has been listed here under resource constraint.
    b. An adequate amount of data for a standardized model accuracy is a challenge and some parts of data will be altered as per need of the hour.
    c. Stress testing could become a resource constraint.

3. Quality Constraints:
    a. In agile SDLC, the quality constraint is one of the most talked-about problems but our team will ensure a production-ready code with code quality reports in order to keep a check on any areas where quality standards are not met.
    b. Baseline comparison and standard quality testing measures will be applied to the decision-making model as well to ensure no compromises.