Software Engineering project

[PICK ME UP]

Pick me Up

Table of Content

Analysis and Domain Modeling

a.) Conceptual Diagram

- **Concept Definitions**

Use Case 1: addDriver

Responsibility Description	Type	Concept Name
Coordinate actions of concepts associated with this use case and delegate the work to other concepts	D	Controller
Form with driver's details to be saved	K	Driver details
Check that all fields are filled and has correct format	D	Information Checker
Prepare database, to save driver's details	D	Database Connection
Prepare pop up/dialogue box, informing that the driver has been added	D	Pop-up Maker

Use Case 5: viewTaxiInfo

Responsibility Description	Type	Concept Name
Coordinate actions of concepts associated with this use case and delegate the work to other concepts	D	Controller
Container for the driver's licence ID	K	Search Key
Prepare database query for the Admin's request	D	Database Connection
Container for the driver details	K	Driver details

Render the received details into an HTML page, to be passed to the administrator's browsers for display	D	Page Maker

Use Case 6: requestPickup

Responsibility Description	Type	Concept Name
Coordinate actions of concepts associated with this use case and delegate the work to other concepts	D	Controller
Get client's current location	K	Current Location
Get client's destination location	K	Destination Location
Prepare map for client to select destination	D	Map Maker
Prepare database for client request	D	Database Connection
Prepare pop up/dialogue box, informing of driver added	D	Pop-up Maker

Use Case 9: activateTaxi

Responsibility Description	Type	Concept Name
Coordinate actions of concepts associated with this use case and delegate the work to other concepts	D	Controller
Get the driver's current state	K	Store state
Get the driver's new state	K	Store new state
Prepare pop up/dialogue box with the possible states of availability (busy, available, unavailable)	D	Selection Maker
Save state to database	D	Database Connection

Use Case 17: listRequest

Responsibility Description	Type	Concept Name
Coordinate actions of concepts associated with this use case and delegate the work to other concepts	D	Controller
Prepares driver's query for the list of available pickup requests	D	Database Connection
Container for list of available requests	K	Request list
Render the received details into an HTML page, to be passed to the driver's browsers for display	D	Page Maker
Prepare map for route preview	D	Map Maker

Use Case 18: AcceptRequest

Responsibility Description	Type	Concept Name
Coordinate actions of concepts associated with this use case and delegate the work to other concepts	D	Controller
Get identification of the driver accepting	K	Driver tracker
Get the client information of the request accepted	K	Client tracker
Check whether the service request is still available	D	Check availability
Updates the database, removes the request selected	D	Remove Request
Render map route	D	Map Maker
Notify the client that his/her request has been accepted	D	Notifier

- ## Association Definitions

Use Case 1: addDriver

Concept pair	Association description	Association name
Controller ↔ Driver Details	Stores the driver's data entered	Store data
Driver Details ↔ Information Checker	Check that the information in the form is complete and check the format	Check data
Controller ↔ Database Connection	Controller passes the query to the database	queries data
Database Connection ↔ Pop-up Maker	Database Connections passes the message that the data has been saved in the database	Provides response
Pop-up Maker ↔ Page Interface	The pop-up is displayed to the page of the administrator's browser	Displays

Use Case 5: viewTaxiInfo

Concept pair	Association description	Association name
Controller ↔ Search Key	Controller passes/stores the driver's license	Store data
Controller ↔ Database Connection	Controller passes the query to the database to search for the driver	Queries data
Database Connection ↔ Driver Details	Stores the results from the query passed by the database	Query results
Driver Details ↔ Page Maker	Results are passed to the page maker to display the data of the driver searched	Provide results

Use Case 6: requestPickup

Concept pair	Association description	Association name
Controller ↔ Current Location	Gets the current location of the client	Provide Location
Controller ↔ Destination Location	Gets the destination location of the client	Provide Location
Controller ↔ Map Maker	Controller creates a mini map of the client to select his/her destination	Render Map
Controller ↔ Database Connection	Controller passes service query to the database	Queries
Database Connection ↔ Pop-up Maker	Database Connection passes a message to the client that his/her request has been placed	Provides response
Pop-up Maker ↔ Page Interface	The pop-up is displayed to the page of the administrator’s browser	Displays

Use Case 9: activateTaxi

Concept pair	Association description	Association name
Controller ↔ Store state	Stores the current state of the driver	Store data
Store state ↔ Selection Maker	Selection Maker uses the current state to create the pop-up with the states except the current state	Passes data
Selection Maker ↔ Store new state	Saves the selected state	Provide selection
Selection Maker ↔ Database Connection	Saves the new state to the database	Save state
Selection Maker ↔ Page Interface	Displays the pop-up display with the states	Displays

Use Case 17: listRequest

Concept pair	Association description	Association name
Controller ↔ Database Connection	Controller passes the query to the database	Queries
Database Connection ↔ Request list	Request list holds the data passed by the database	Database results
Request list ↔ Page Maker	Request list passes the results to the Page Maker to display it to the driver	Provide results
Request list ↔ Map Maker	Render a map preview of the route	Render Map
Page Interface ↔ Map Maker	Page maker displays the mini map	Displays

Use Case 18: AcceptRequest

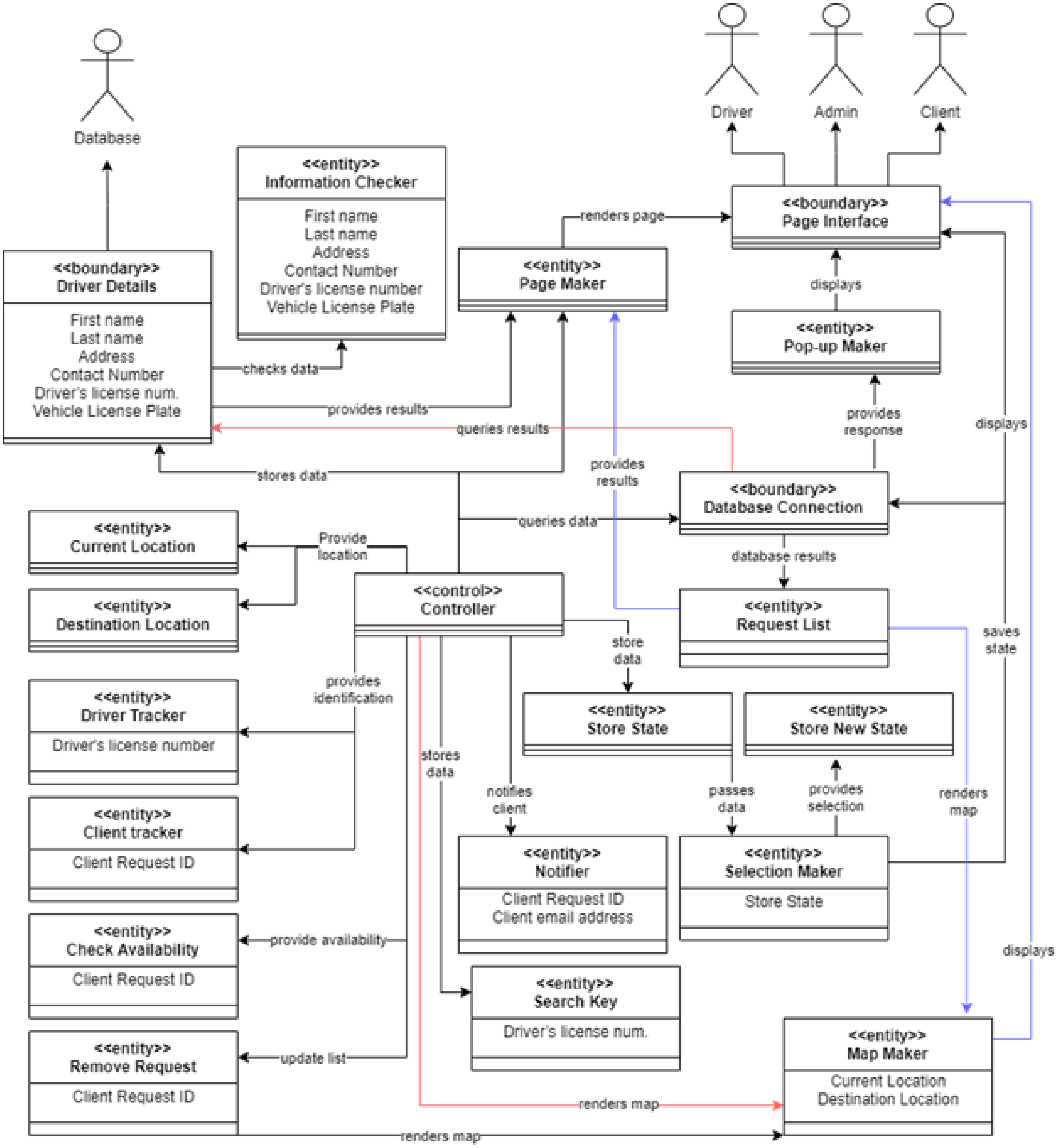
Concept pair	Association description	Association name
Controller ↔ Driver tracker	Controller saves/holds the identification of the driver accepting the request	Provide identification
Controller ↔ Client tracker	Controller save/holds the request selected by the driver	Provide identification
Controller ↔ Check availability	Controller passes identification of the driver and the client to see if the selected request is available, or has been taken	Provide availability
Controller ↔ Remove Request	Controller passes the identification of the driver and the client request to be removed	Update list
Remove Request ↔ Map Maker	Database provides the route to be taken	Render map

Controller ↔ Notifier	Controller calls the notifier of the client, letting him/her of accepted request	Notify client

Attribute Definitions

Concept	Attributes	Attribute Description
Controller	NA.	
Driver details	First name Last name Address Contact Number Driver's license num. Vehicle License Plate	Driver's first name Driver's last name Driver's home address Driver's phone number Driver's driving license number Driver's vehicle license plate number
Information checker	First name Last name Address Contact Number Driver's license num. Vehicle License Plate	Driver's first name Driver's last name Driver's home address Driver's phone number Driver's driving license number Driver's vehicle license plate number
Database connection	NA.	
Pop-up maker	NA.	
Search key	Driver's license num.	Driver's driving license number
Page maker	NA.	
Current Location	NA.	
Destination Location	NA.	

Map Maker	Current Location Destination Location	The current GPS location of the client The destination location of the client
Store state	NA.	
Store new state	NA.	
Selection Maker	Store state	The current state that the driver is in
Driver tracker	Driver's license num.	Driver's driving license number
Client tracker	Client request ID	The identification of the request made by the client
Check availability	Client request ID	The identification of the request made by the client
Remove Request	Client request ID	The identification of the request made by the client
Notifier	Client request ID Client email address	The identification of the request made by the client Client's email address, used to notify client of acceptance


Database
Driver
Admin
Client
<<entity>>
Information Checker
First name
Last name
Address
Contact Number
Driver's license number
Vehicle License Plate
<<boundary>>
Driver Details
First name
Last name
Address
Contact Number
Driver's license num.
Vehicle License Plate
<<entity>>
Page Maker
<<boundary>>
Page Interface
<<entity>>
Pop-up Maker
renders page
displays
checks data
provides results
queries results
provides response
displays
provides results
stores data
<<boundary>>
Database Connection
queries data
database results
<<entity>>
Current Location
<<entity>>
Destination Location
Provide location
<<control>>
Controller
<<entity>>
Request List
store data
saves state
<<entity>>
Driver Tracker
Driver's license number
provides identification
<<entity>>
Store State
<<entity>>
Store New State
stores data
notifies client
passes data
provides selection
renders map
<<entity>>
Client tracker
Client Request ID
<<entity>>
Notifier
Client Request ID
Client email address
<<entity>>
Selection Maker
Store State
<<entity>>
Check Availability
Client Request ID
provide availability
displays
<<entity>>
Search Key
Driver's license num.
<<entity>>
Remove Request
Client Request ID
update list
<<entity>>
Map Maker
Current Location
Destination Location
renders map
renders map

Traceability Matrix

REQ	UC-1	UC-5	UC-6	UC-9	UC-17	UC-18
PW	8	4	4	6	9	9
Controller	X	X	X	X	X	X
Driver details	X	X				
Information checker	X					
Database connection	X	X	X	X	X	
Dialogue maker	X		X			
Search key		X				
Page maker		X			X	
Current Location			X			
Destination Location			X			
Map Maker			X		X	X
Store state				X		
Store new state				X		
Selection Maker				X		
Driver tracker						X
Request list					X	
Client tracker						X
Check availability						X
Remove Request						X
Notifier						X

b.) System Operation Contracts

Contract Name:	**addDriver** (firstName, lastName, address, contractNumber, driverLicenseNum, vehicleLicensePlate)
Responsibilities:	Add a driver to the system
Type:	System
Exceptions:	If there are fields empty, signal/flag the empty field (s) If the format entered is incorrect (2154-5848 instead of 654-7521), signal/flag the field (s)
Preconditions:	Administrator has already logged in and opened up a form
Postconditions:	The data is entered into the database

Contract Name:	**viewTaxiInfo** (firstName, lastName, address, contractNumber, driverLicenseNum, vehicleLicensePlate)
Responsibilities:	Fetch the driver’s information from the database, and display it to the admin
Type:	System
Exceptions:	driverLicenseNum does not exist in the database when searched upon
Preconditions:	Administrator has already logged in
Postconditions:	Display a list of drivers or a single record if searched upon by the administrator

Contract Name:	**requestPickup** (clientEmailAddress, currentAddress, destinationAddress)
Responsibilities:	Place a request (trip service) to the drivers, by the client
Type:	System
Exceptions:	None
Preconditions:	The client has signed in and has his/her GPS turned on
Postconditions:	Place a request to the system, asking for service

Contract Name:	**activateTaxi** (driverLicenseNum, currentState, newState)
Responsibilities:	To update/change the status of the driver (available, unavailable, busy)
Type:	System
Exceptions:	None
Preconditions:	Driver has logged in
Postconditions:	Change his/her status to another status

Contract Name:	**listRequest** (requestID, currentAddress, destinationAddress)
Responsibilities:	To fetch the list of requests (services pending)
Type:	System
Exceptions:	None
Preconditions:	Driver has logged in and has an available status
Postconditions:	Display a list of people that require (trip) services

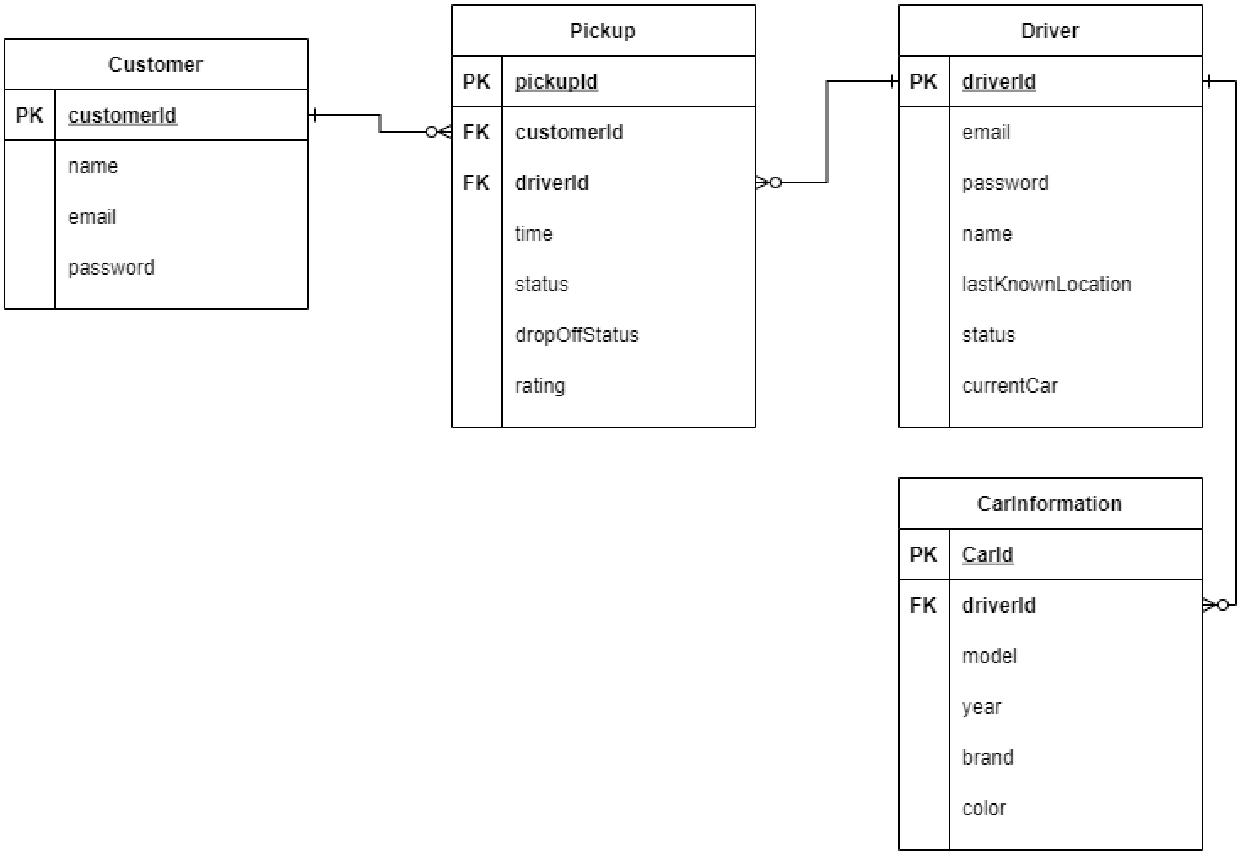
Contract Name:	**AcceptRequest** (requestID, currentAddress, destinationAddress)
Responsibilities:	Allow the driver to accept the client's request (service request)
Type:	System
Exceptions:	The request has already been accepted (seconds ago) by another driver
Preconditions:	Driver has logged in, has an available status and has selected a request
Postconditions:	Allow/grant the driver to accept the request set by the client Notify the client that his/her request has been accepted Provide details of the driver that accepted the request

c.) Persistent Data Storage

Since the System will require presenting and interacting with the locations of many people at the same time, a system of temporarily storing the users last known location as well as persisting information needs to be built. The data will be stored on a Google Firebase Database. This is an external cloud based storage system that meets the requirements of both storage capacity and price range. The database will be responsible for storing driver information and car information, as well as hold basic login information for users and information regarding pickups.

Some of the persisting information that will be stored include:

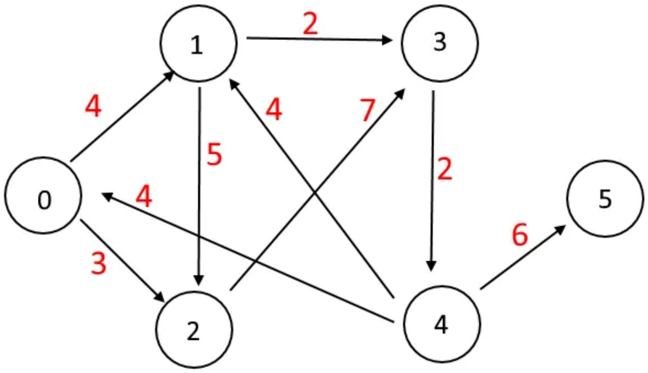
1. Customer: Login information and their name.
2. Driver: Driver login information, name, last known location, address, which car they are currently driving.
3. CarInformation: Information about the car such as model, year, brand, color.
4. Pickup: customerId, driverId, time of request, status, when customer left, and rating.



d.) Mathematical Model

Map

The primary mathematical model used in this map is the concept of 2D Graphs. More specifically, a 2D weighted graph.
An example of such a graph can be seen below.

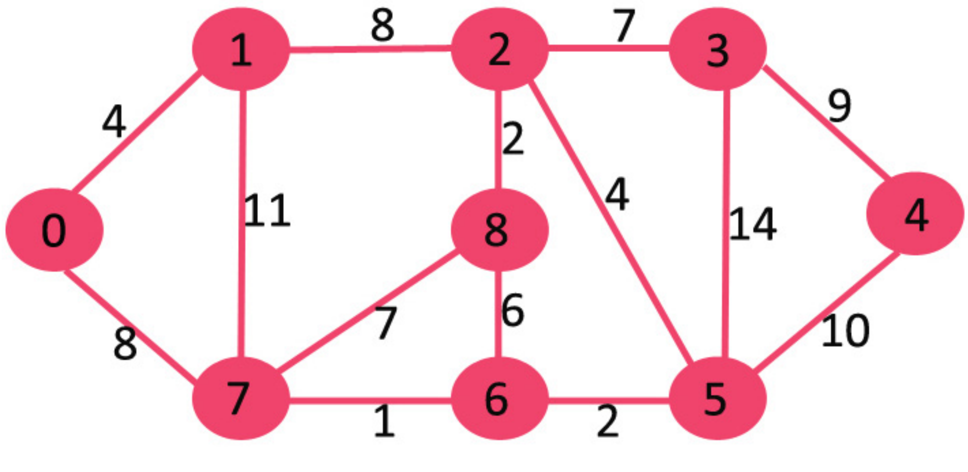


Weighted Graph

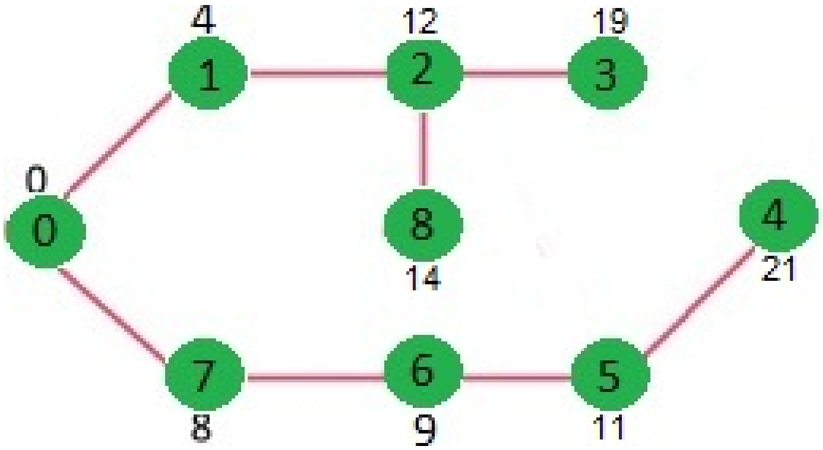
The graph is then displayed on the Google Map Interface at specified coordinates.

Routes

The routes for the graph are made by the Google Directions API using Dijkstra's Algorithm. Dijkstra's Algorithm finds the shortest path to all vertices from a source vertex. An example of a completed shortest path tree can be seen below.



Base Weighted Graph

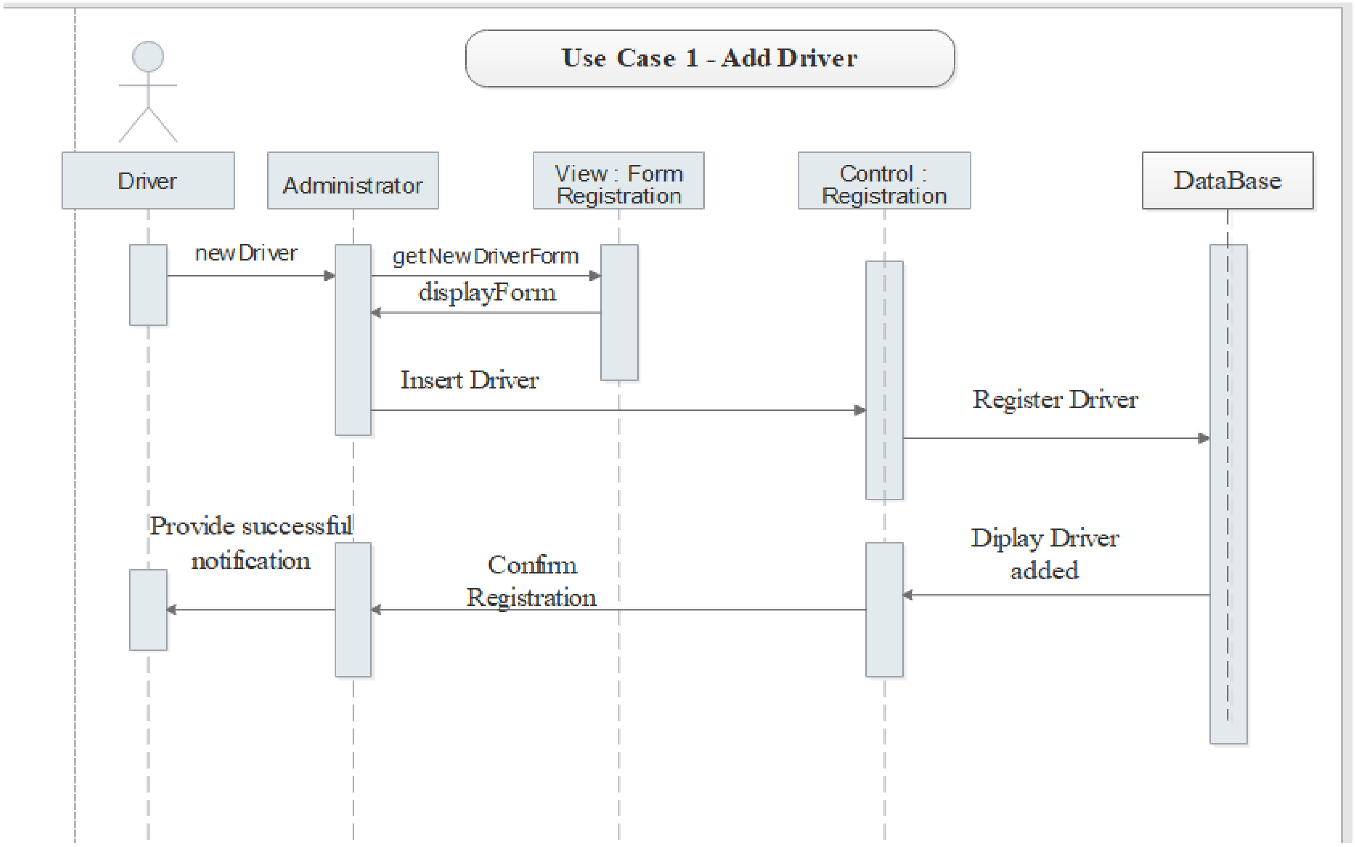


Shortest Tree Path Based On Previous Weighted Graph

This shortest tree path is then used by the Google Directions API to find the shortest path between two locations.

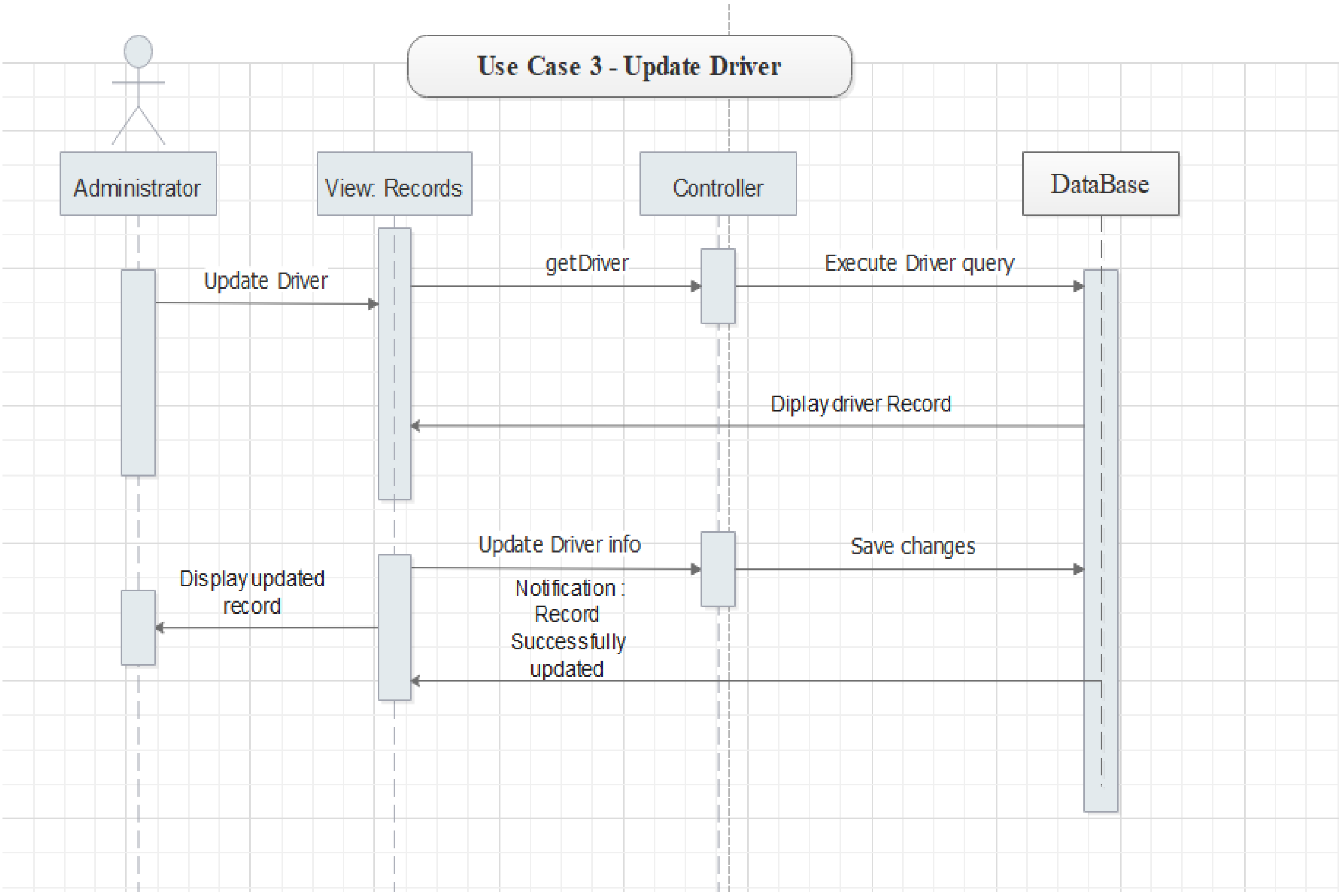
Interaction Diagrams

1 . Add Driver



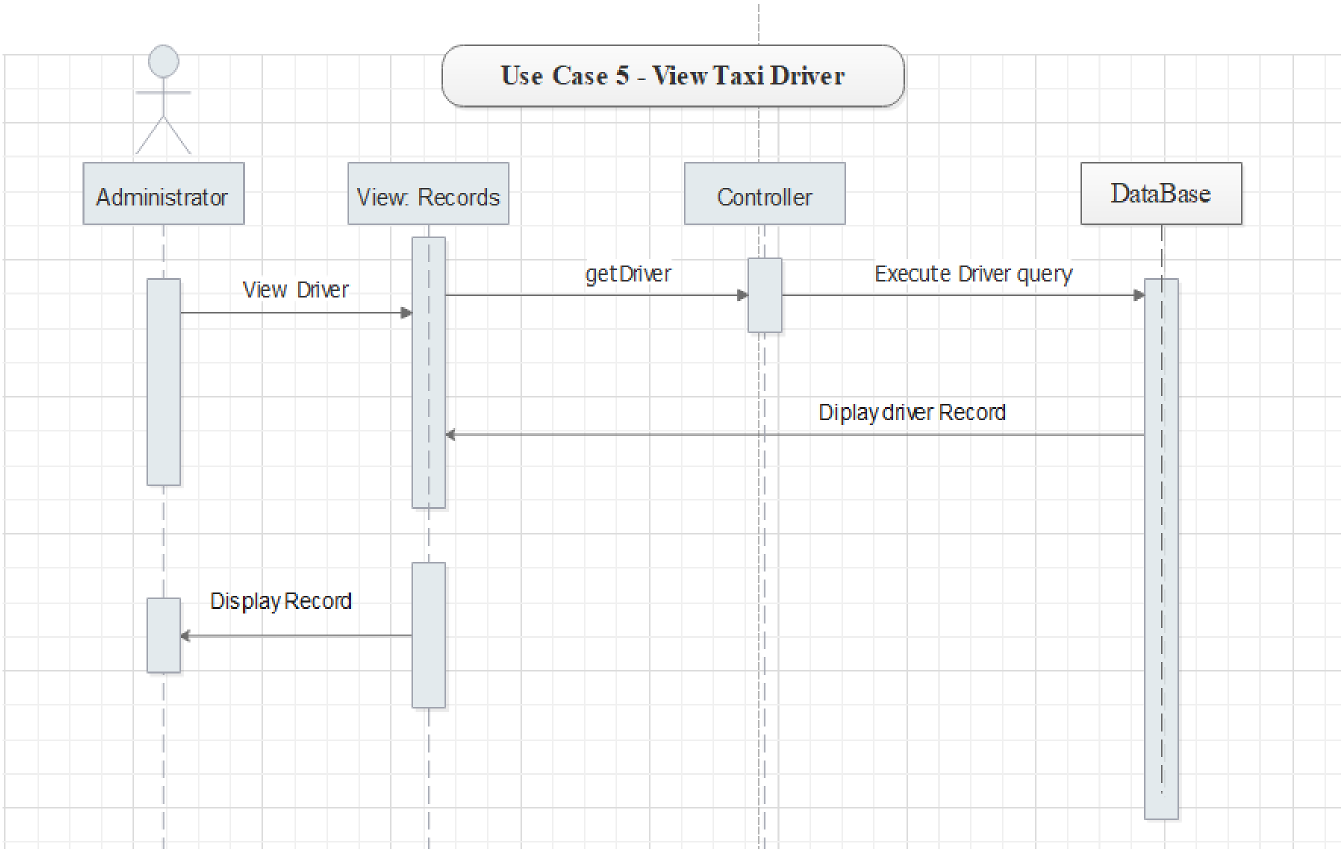
- Once a taxi driver wants to register to the “Pick me up App” the admin first proceeds to open the new driver form from the registration section, once the form meets all the requirements it is then sent to the controller which verifies the input and passes the register driver command to the database.
- Once the database registers the new driver it then sends a confirmation to the admin of the successful registration and how a new driver has been added to the Pick me up taxi group.

2. Update Driver



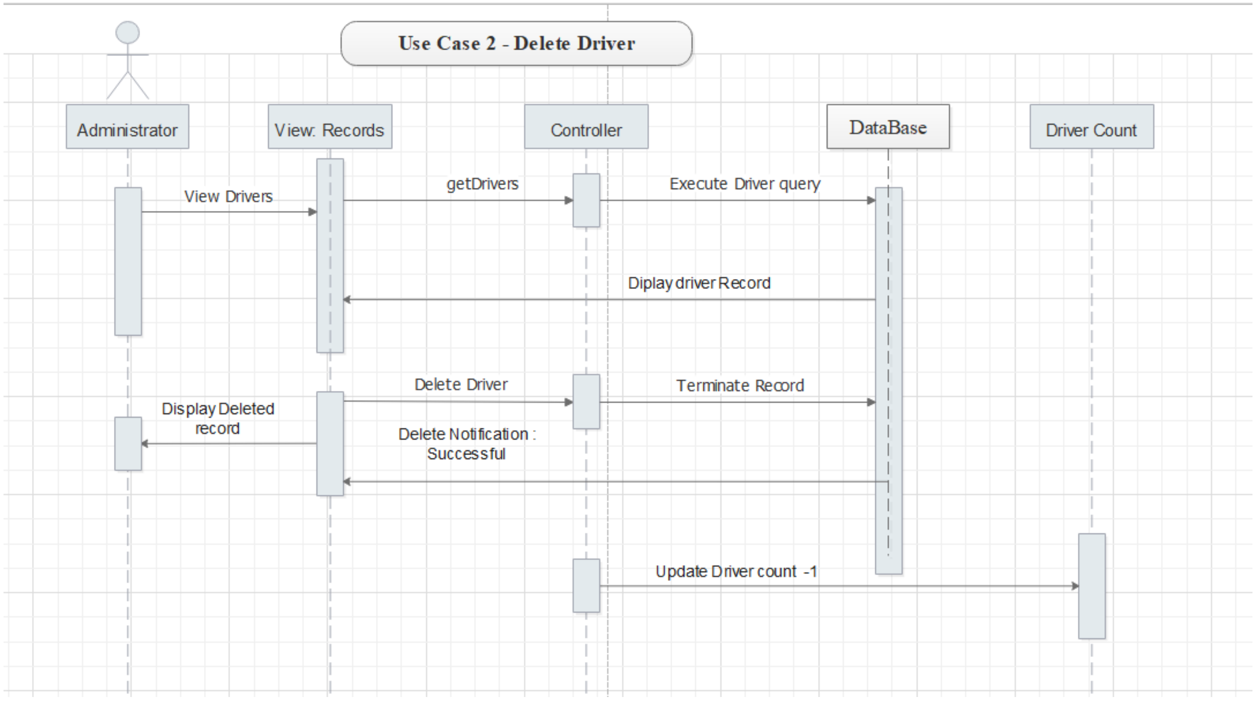
- After a taxi driver either changes vehicle or licence type he then asks the administrator to make the necessary changes to reflect his record. The administrator then proceeds to call the drivers record from the database, once the update has been done the controller then issues the save changes command to the database which then notifies the admin about the record being updated successfully.
-

3. View Taxi Driver



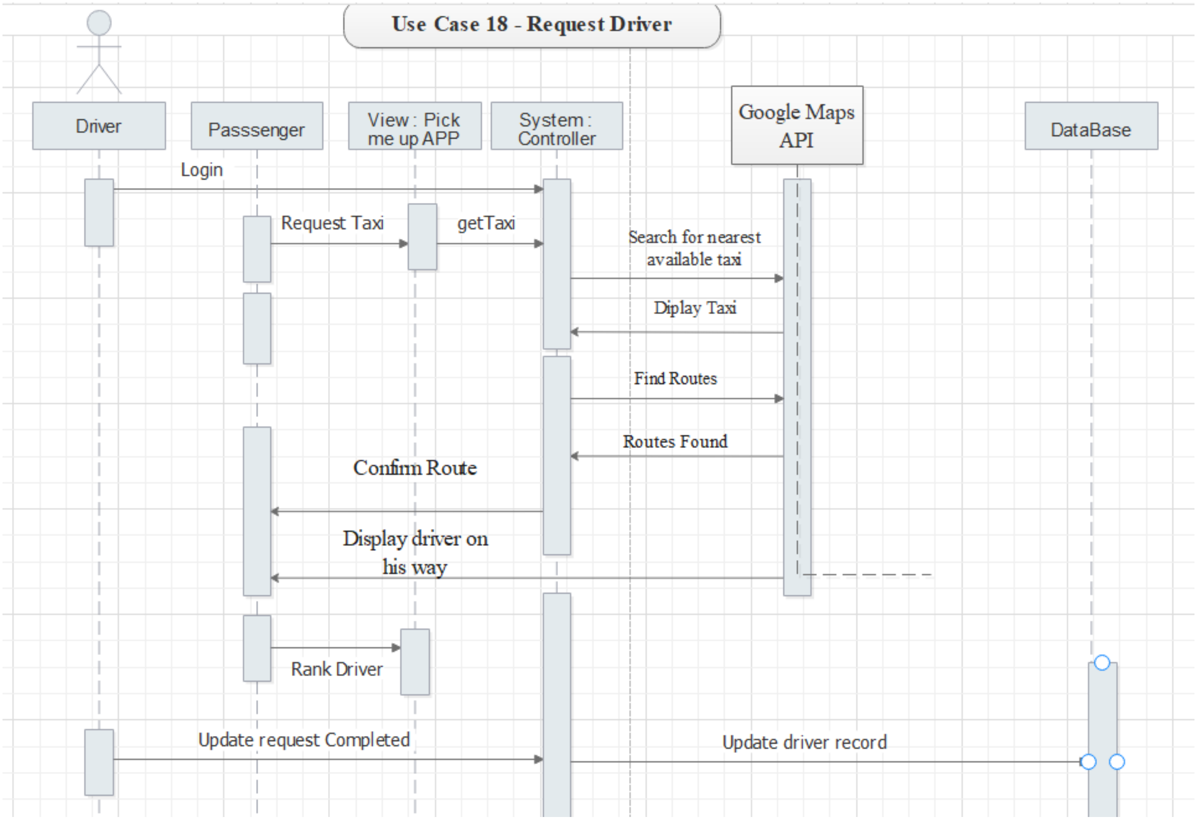
- Once the administrator wants to view a driver he first checks the records list, then provides a command to the controller to getDrivers list. Once the database gets the execute driver query from the database the database displays the driver record to the administrator.

4 . Delete Driver



- When deleting a taxi driver, the admin first clicks the view record profile, once the record is clicked the command goes to the controller to initiate the database. The database then provides the list of drivers, the admin selects which record he/she wants to delete and passess the message to the controller to delete record X. Once record X has been deleted the database notifies the admin with a delete notification followed by the controller which removed 1 from total driver count.

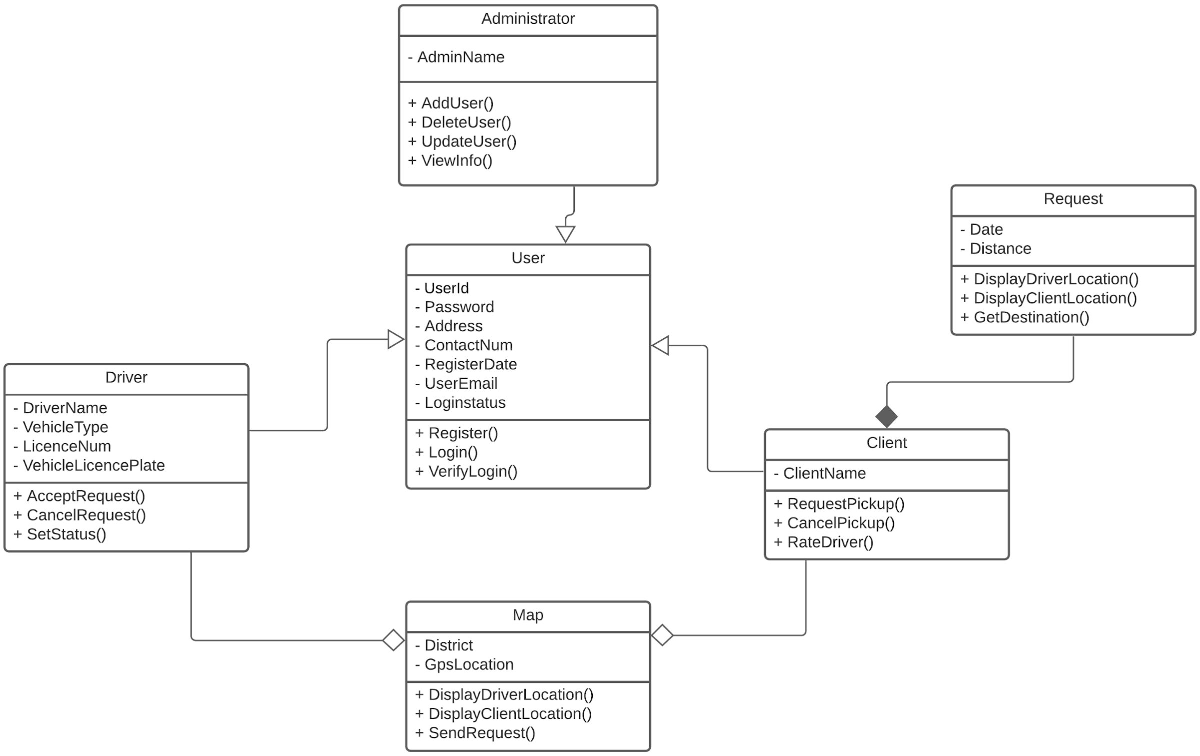
5. Request Driver



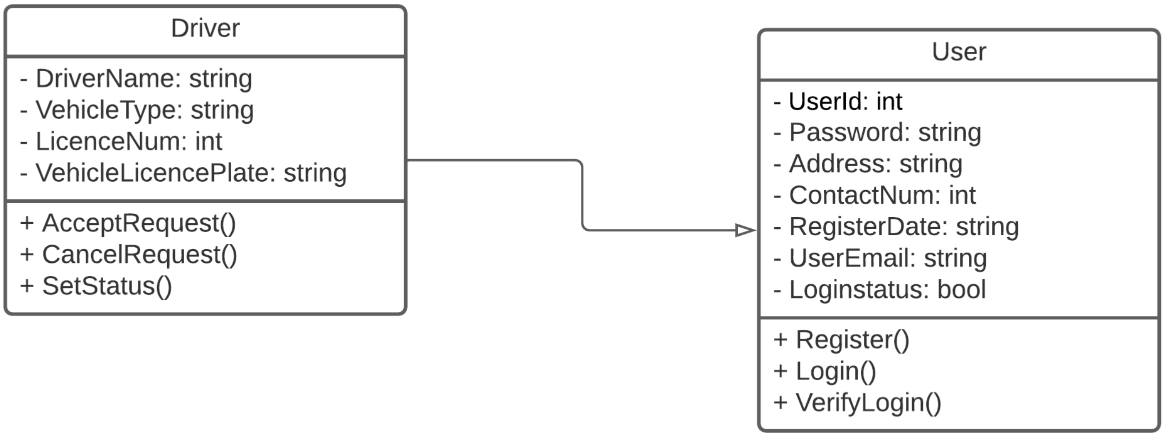
- Once the driver is logged in to the pick me up app, he can then set his status to : available, busy or offline. Once the passenger clicks on the pick me up app they can then proceed to request for a taxi, the app then notifies the system controller to get an available taxi. The controller proceeds to search for the nearest available taxi in google map. Once the nearest available taxi is identified then the controller finds and identifies the nearest route to the passenger, once the passenger confirms the route google maps displays that the driver is on his way. After the passenger has been dropped off, the passenger proceeds to rank the driver where the controller then accepts the ranking information and updates the database to show the quality of service that was provided by the driver.

Class Diagram and Interface Specification

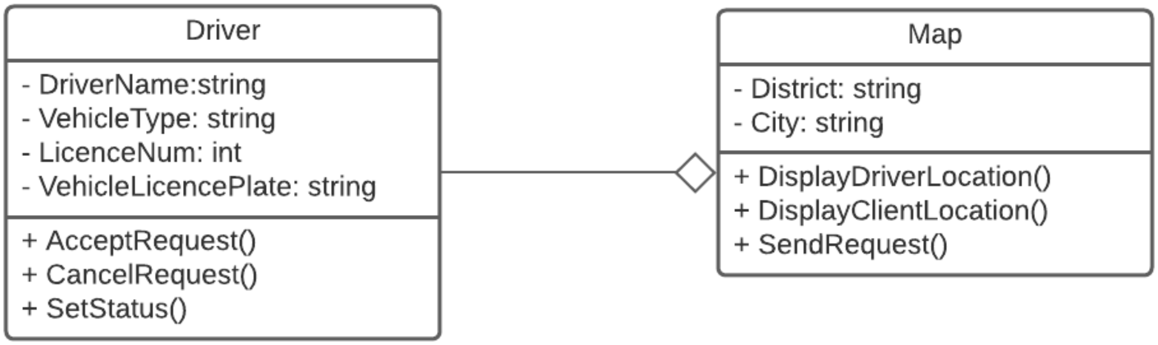
1. Class Diagram



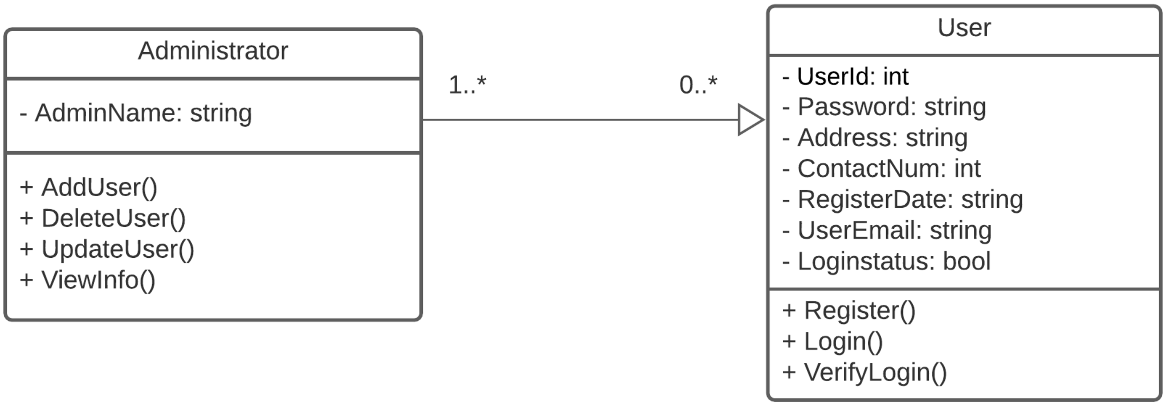
Data Types and Operation Signatures



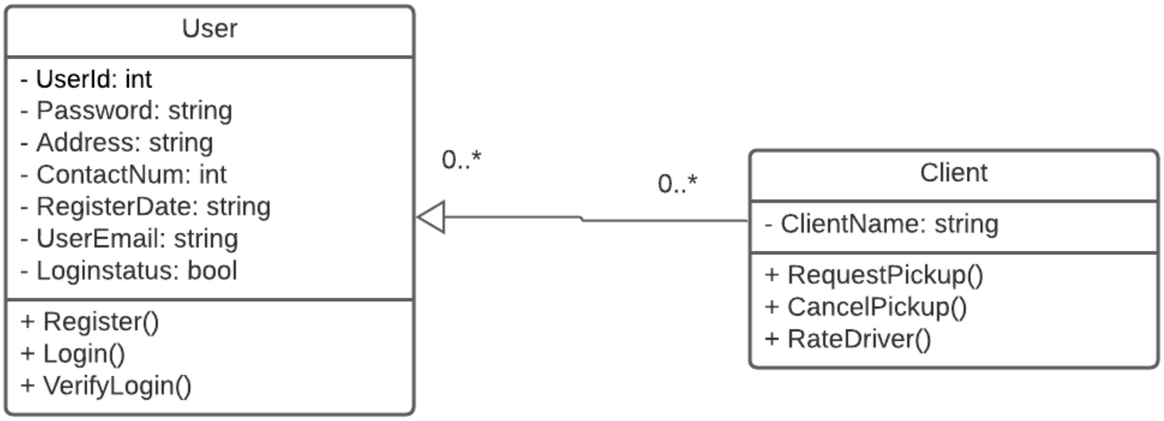
- Driver inherited from User while containing specific attributes for driver users alone. All Users may create an account and login.
- Drivers have the option to set their status to



- Driver users have access to the map and may accept/cancel requests they've received. In addition they may also set their current status to "busy" "active" or "inactive"
- A Driver's location must be seen on the map by clients, however will not be shown if they are set to status "inactive"



- Administrators inherit from the User class, and they may modify user information that includes both driver and clients.
- The Administrator is responsible for adding, updating and removing users, using the similarly named operations/methods; and users being both drivers AND clients. ViewInfo() is for admins to view information of users as



- Clients inherited the user class and are able to request and cancel a request, as well as give the option to rate the driver once a drive is over.
- Clients can request a pickup, thus notifying the driver. They may also cancel this request. At the end of a drive the client has the option to rate the driver.

Traceability Matrix Evolved

(a check/X signifies the concept(left) as part of the class (above)

REQ	Driver	Admin	User	Map	Client	Requests
PW	8	4	4	6	9	9
Controller	X	X	X	X	X	X
Driver details	X	X	X			
Information checker	X	X	X		X	
Database connection	X	X	X	X	X	X
Dialogue maker		X				
Search key	X	X				
Page maker		X				
Current Location	X				X	
Destination Location				X	X	
Map Maker			X		X	X
Store state	X					
Store new state	X			X		
Selection Maker				X	X	
Driver tracker				X		
Request list						X
Client tracker						
Check availability				X		X
Remove Request					X	X
Notifier				X	X	X

- It was decided to combine related concepts into a single class, concepts such as RemoveRequest, requestList, and so on.
- Some concepts are in multiple classes such as DestinationLocation is in both Map and Client. This is because both classes will be utilizing this concept, in the example's case; the **Map** class will mark the DestinationLocation on the display map, and the **ClientClass** is given the option to create the DestinationLocation

Algorithms

The system primarily consists of a pathing algorithm in order to provide the shortest path between two locations on the map. Namely, location of the driver and location of the client. The algorithm will be responsible for identifying this shortest path in order to ensure efficiency of travel. This algorithm will take place using an external API, namely, Google Directions. The system will be in charge of receiving both locations from the two independent parties, then sending it to the API. When the system retrieves the API response, the system will then show the Driver the shortest route between the two locations.

Algorithm:

Driver requests shortest route to client:

```
While “requesting_shortest_route” is true
        Set “Driver Location” to Current Location
        If cannot get Current Location
                Inform user that Locations services is disabled or Internet is not available
                continue
        End if
        Send Current Location and Client Location to Directions API
        Fetch Shortest Route from API
        Display Shortest Route on map view
End While
```

According to Crovari in an article from 2019, Google Directions uses the Dijkstra's Algorithm to find the shortest route.

```
Algorithm Dijkstra(W[1..n, 1..n])
// Shows Dijkstra's Algorithm to find the shortest route in a weighted graph. According to GeeksforGeeks
// Input: A weighted graph W
// Output: Shortest Path Tree
Create empty set S that will hold shortest path tree
Assign all vertices distance values of INF
Assign source vertex distance value of 0
While S does not have all vertices
        a.) Pick unrouted vertex U
        b.) Include U in S
        c.) Update shortest path tree with shortest path to vertex U
End While
Return S
```

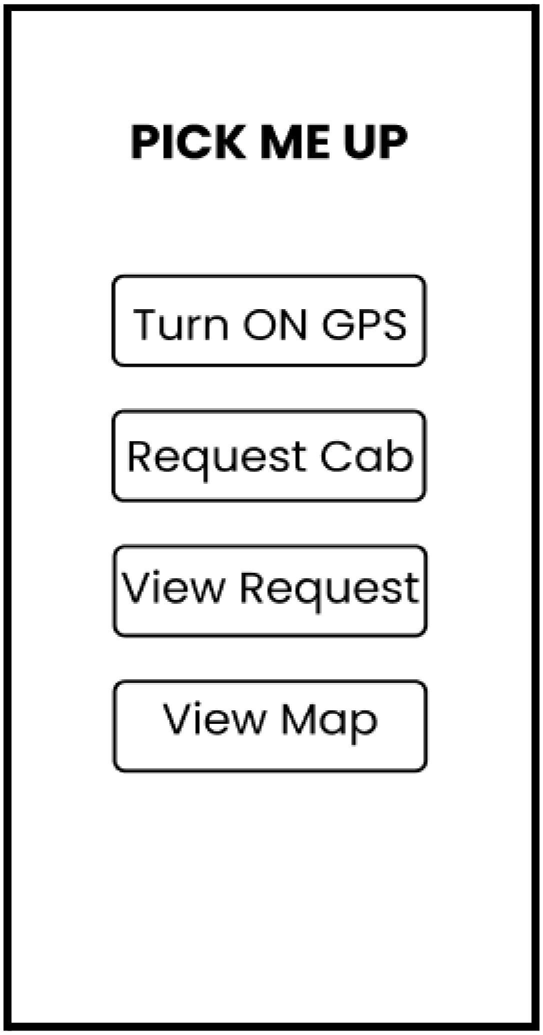
Using this shortest path tree, the Directions API returns the shortest path between two locations.

Concurrency

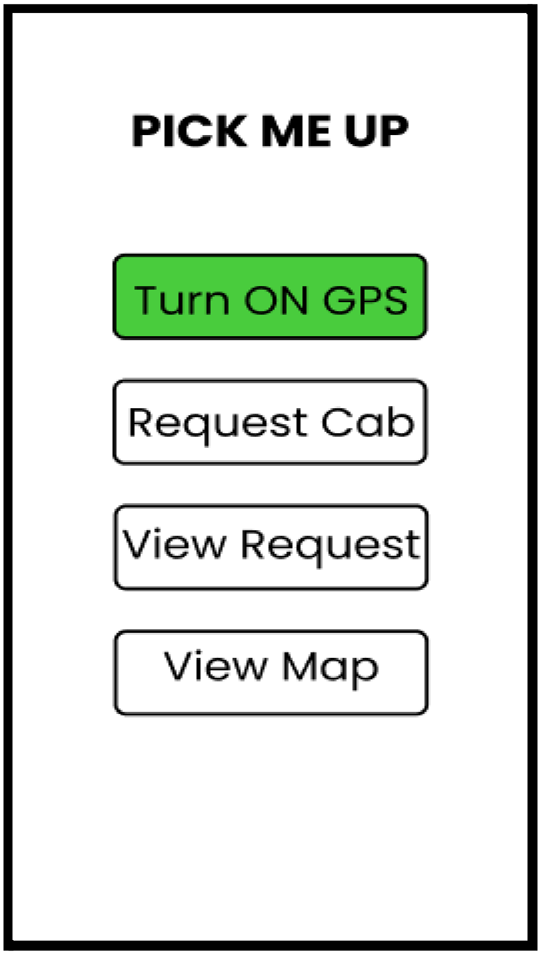
This system will allow for multiple users to be using it at the same time. Since Users will be able to affect data on the system, an exception needs to be put in place that will allow for data to remain consistent throughout the system lifetime. This case specifically applies to different drivers attempting to accept the same pickup request in a short amount of time. Especially with poor internet connectivity, like what is normally experienced by users of 3G in Belize. In order to synchronize the data, and allow for the information to remain consistent, as well as to not double book any pickups, a check within the System will ensure that no other driver has already accepted the pickup.

User Interface Design and Implementation

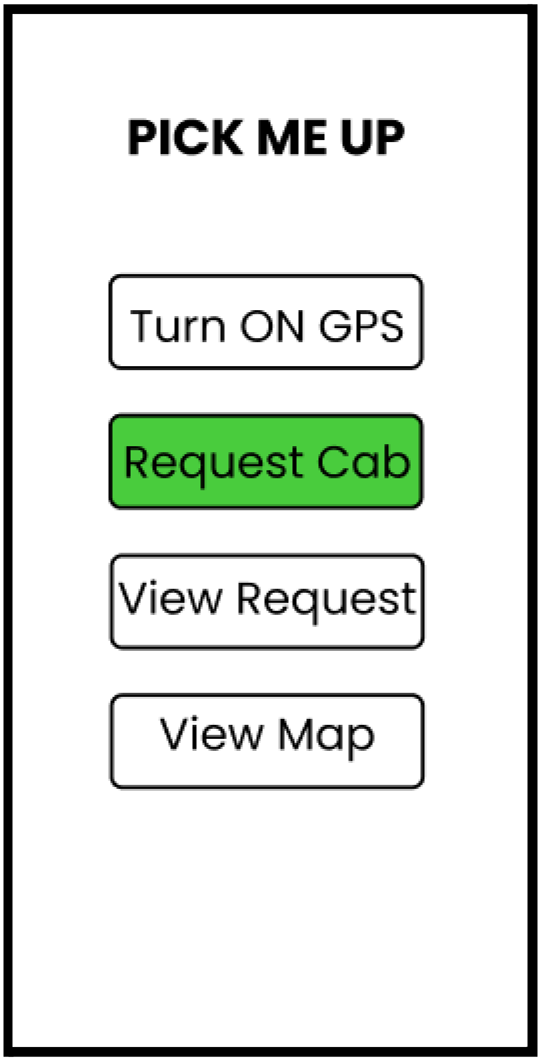
After extensive review of the first report, it is decided that the interface will not receive any changes in it's design and user effort. The interface will stay the same because it is easy to learn and understand with its simplistic design.



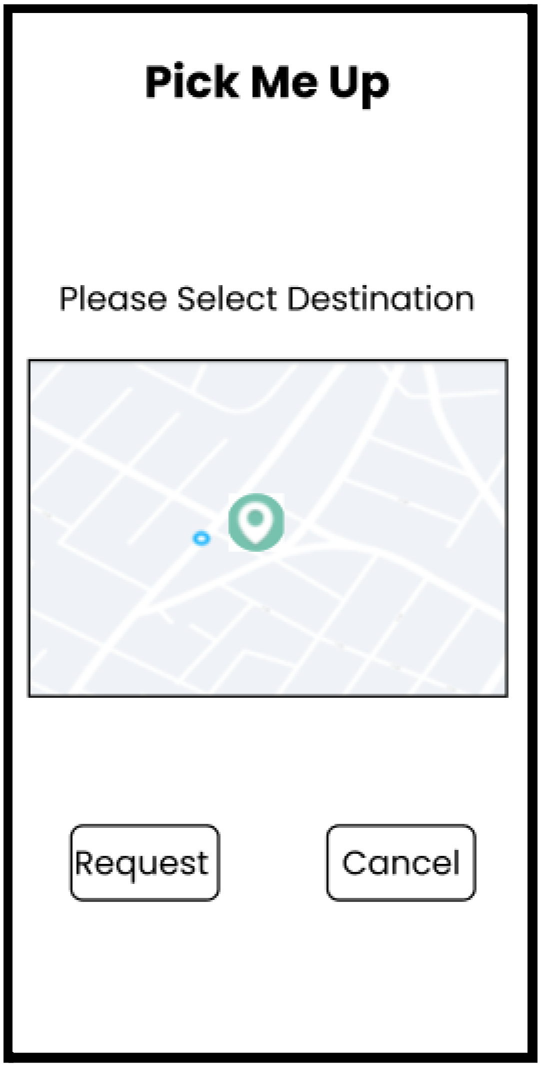
This diagram shows the client homepage after login to the system. The System will be interactive through touch.



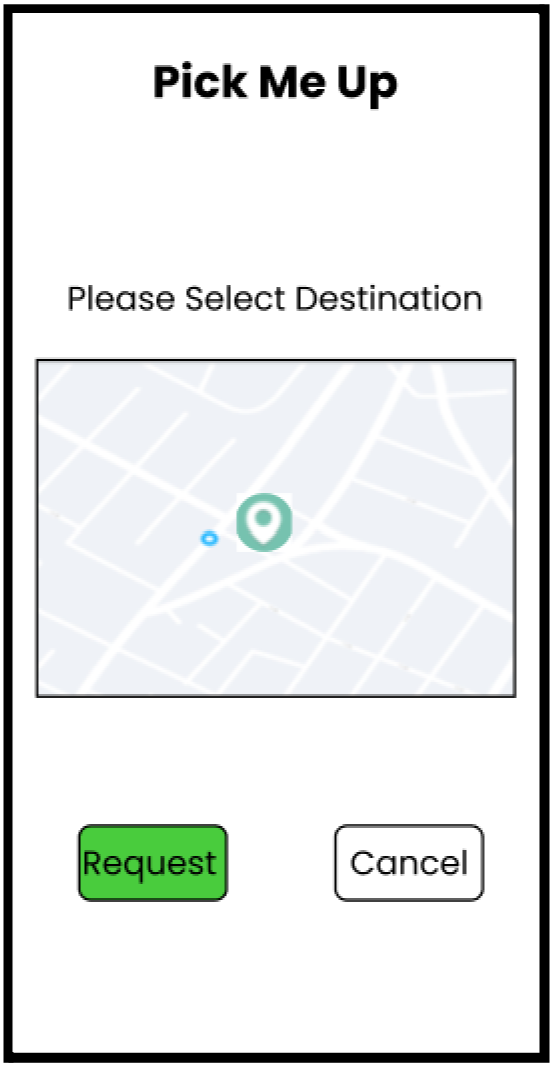
The Green highlights in the diagram will be used to show selection of different options available. The current option highlighted shows that the user GPS is currently on.



The diagram is showing that you can click on the “Request Cab” option, which will bring up another page to confirm the service.



The Diagram shows an interactive map that contains your location and allows you to choose your destination.The client can Cancel or confirm the Pickup service on this page.

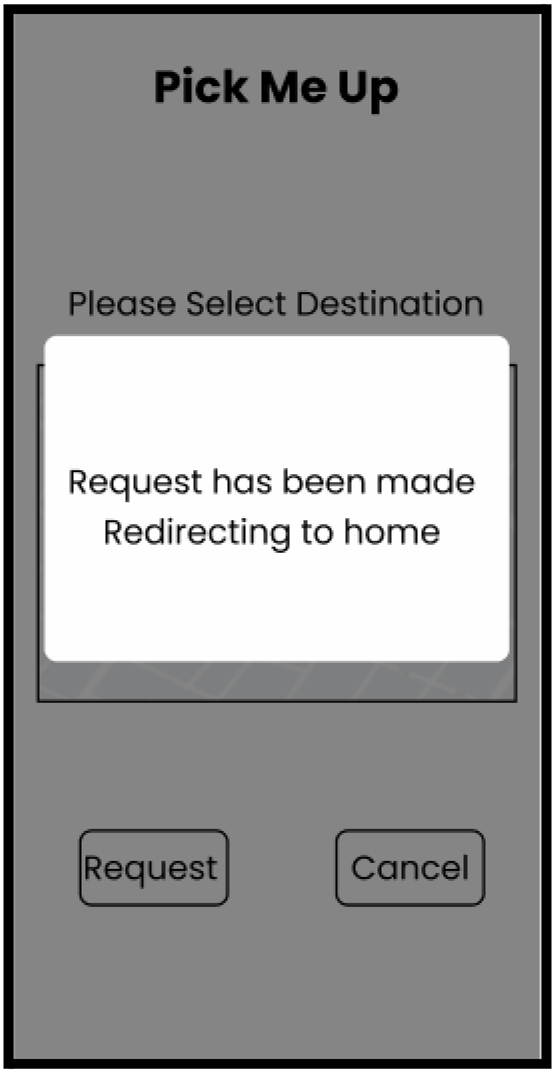


Pick Me Up

Please Select Destination

The Diagram shows you that to finish the confirmation, the client must click the request option.

This will bring up a message showing you that your Pickup Request has been confirmed.

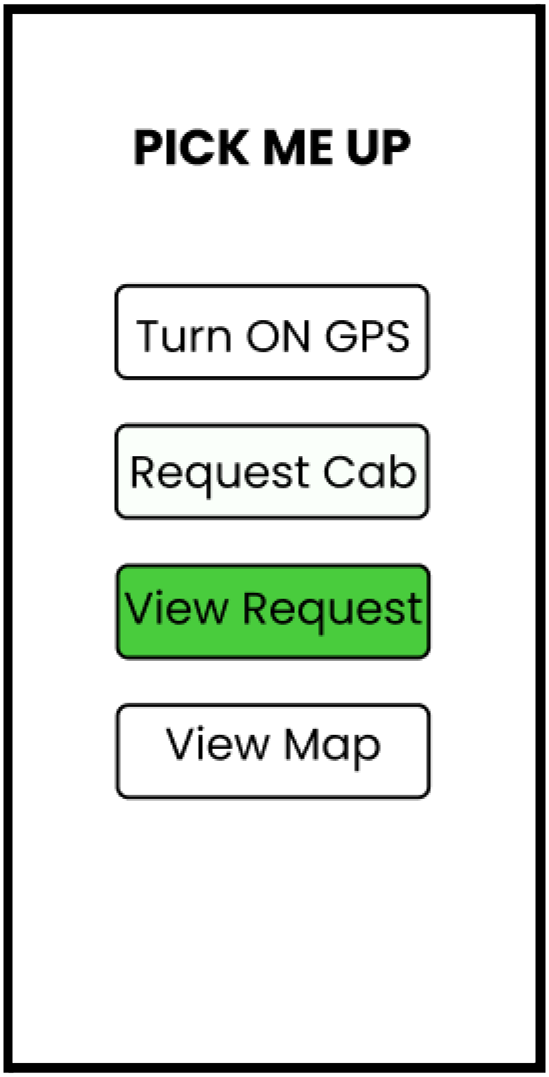


Pick Me Up

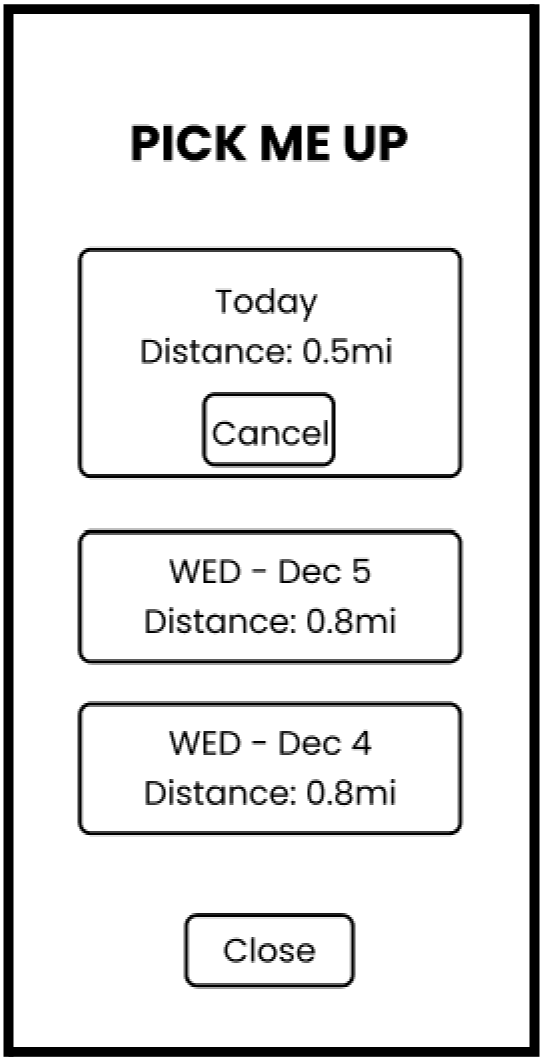
Please Select Destination

Request has been made
Redirecting to home

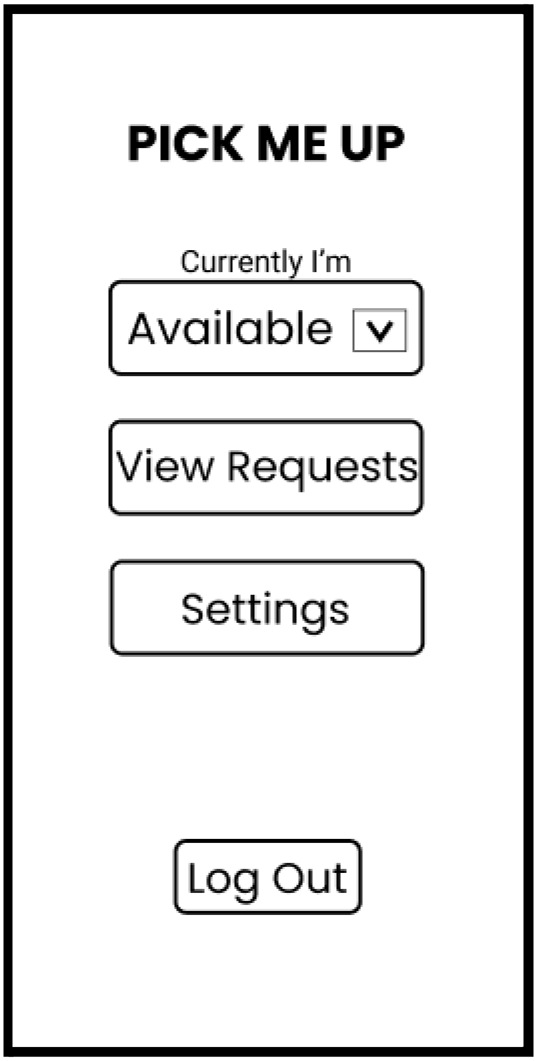
This diagram shows the successful Request Pick up notification message to the customer after choosing a destination and clicking on the Request option. After this notification, the system redirects the user to the homepage.



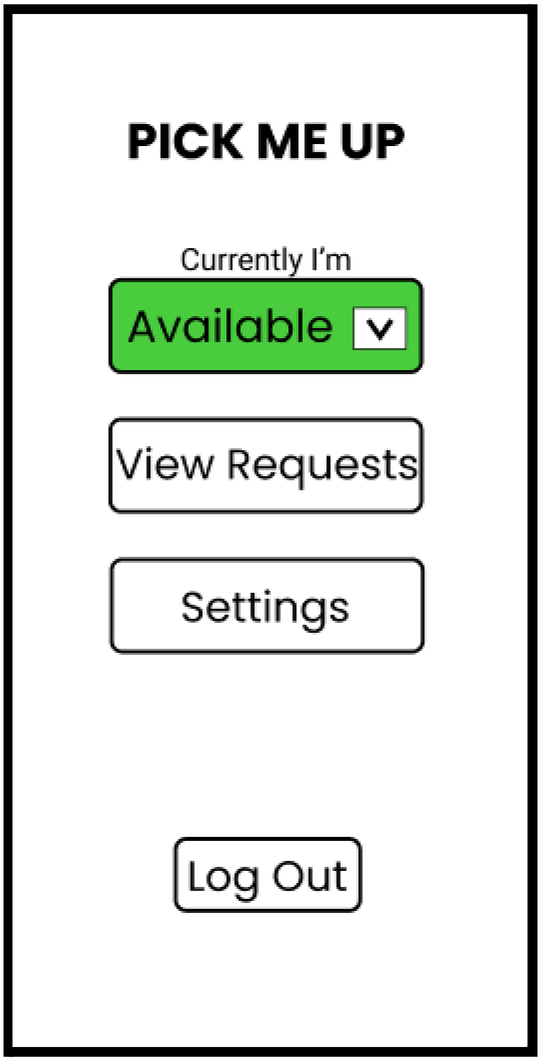
The Diagram shows that the View Request option is being selected, which will bring up another page with the User Pick up Requests.



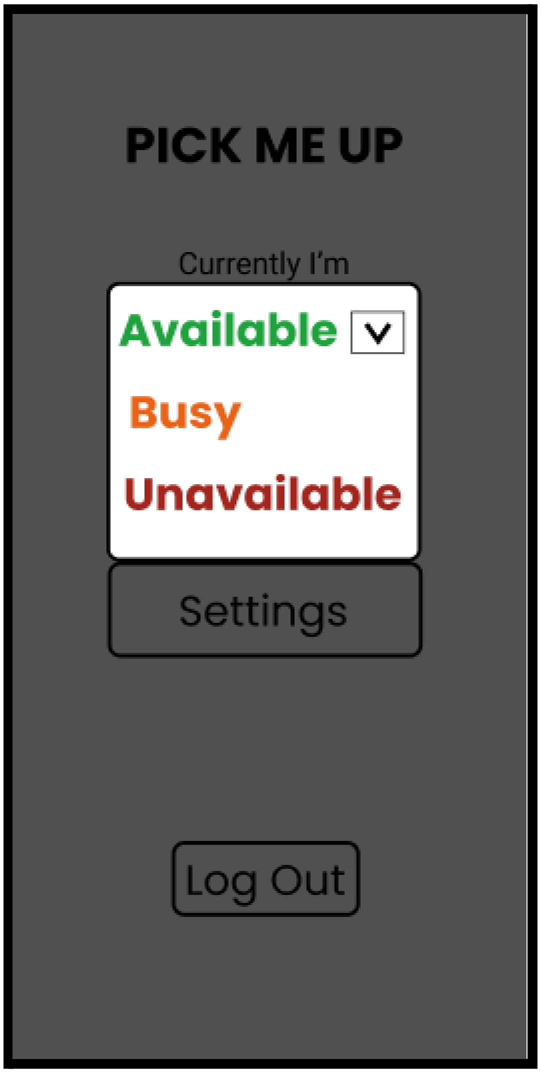
This diagram shows the user all their past Requests and also the user's ongoing request. There are 2 options given, one is the close option which will return the user to the homepage and the other option is to cancel the ongoing Pickup request.



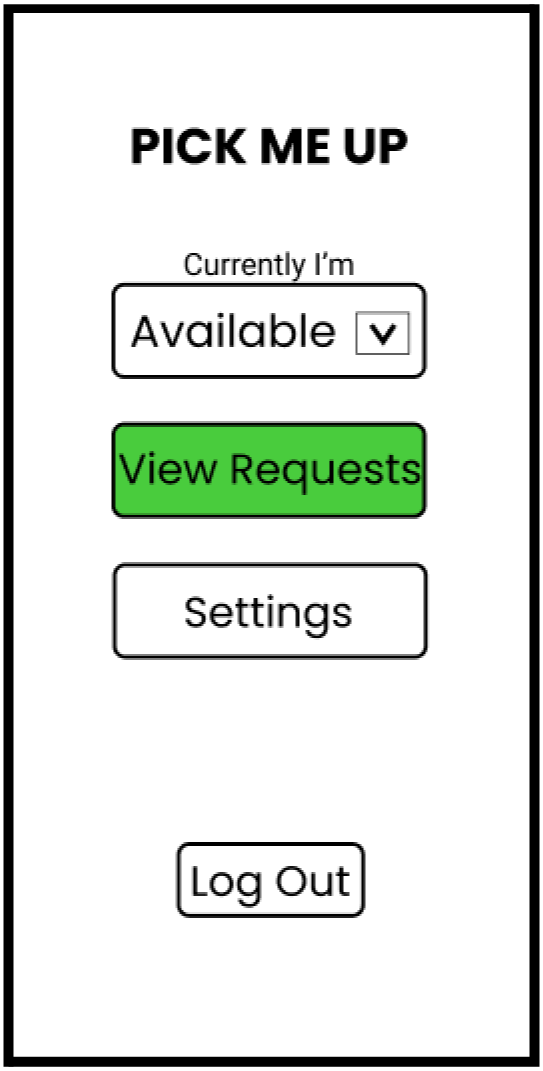
This Diagram shows the homepage for the Driver. There are 4 options available for the driver to carry out his task.



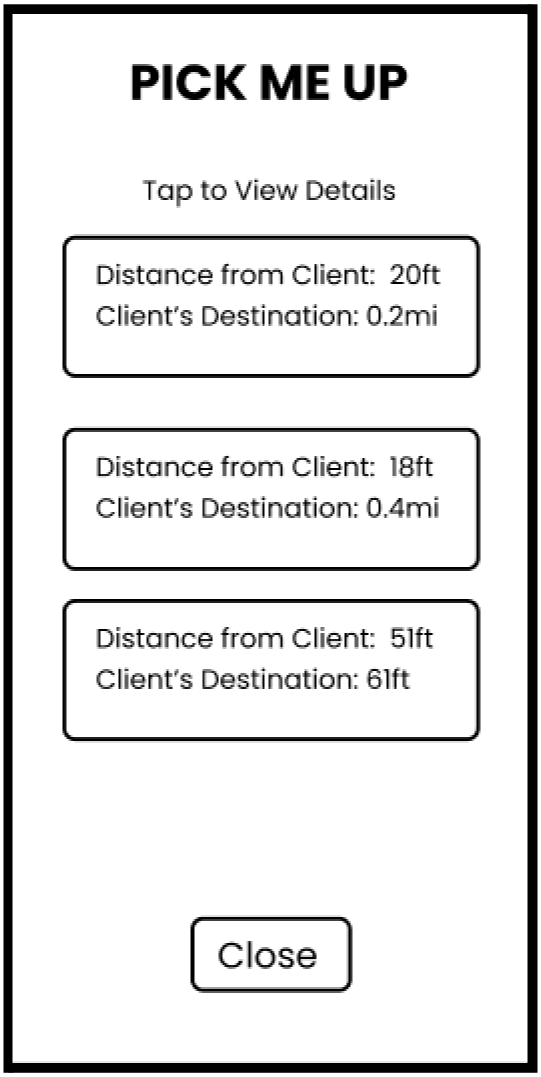
This diagram shows how the user can click on the status options and this will allow the user to set if they're busy,available or not.



This diagram shows the 3 options available when the user clicks on their status and decides if they are available to work or not.



This diagram shows you that the option view Request can be clicked on. This will carry the user to another page showing all of the available requests.



PICK ME UP

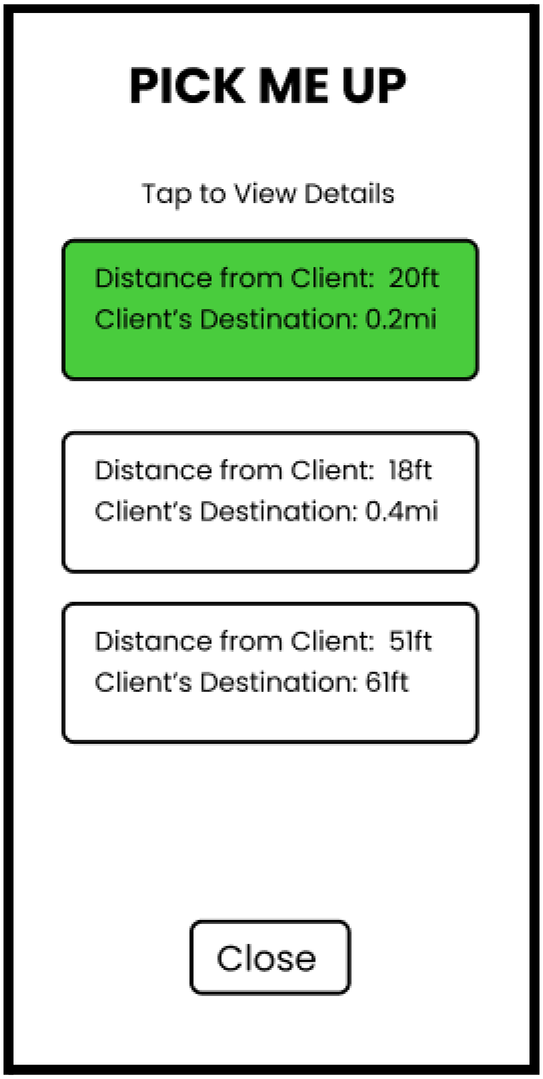
Tap to View Details

Distance from Client: 20ft
Client's Destination: 0.2mi

Distance from Client: 18ft
Client's Destination: 0.4mi

Distance from Client: 51ft
Client's Destination: 61ft

This diagram shows the user all of the available Pick up requests. The user can then choose a request and get a more detailed view of the request or go back to the home page.



PICK ME UP

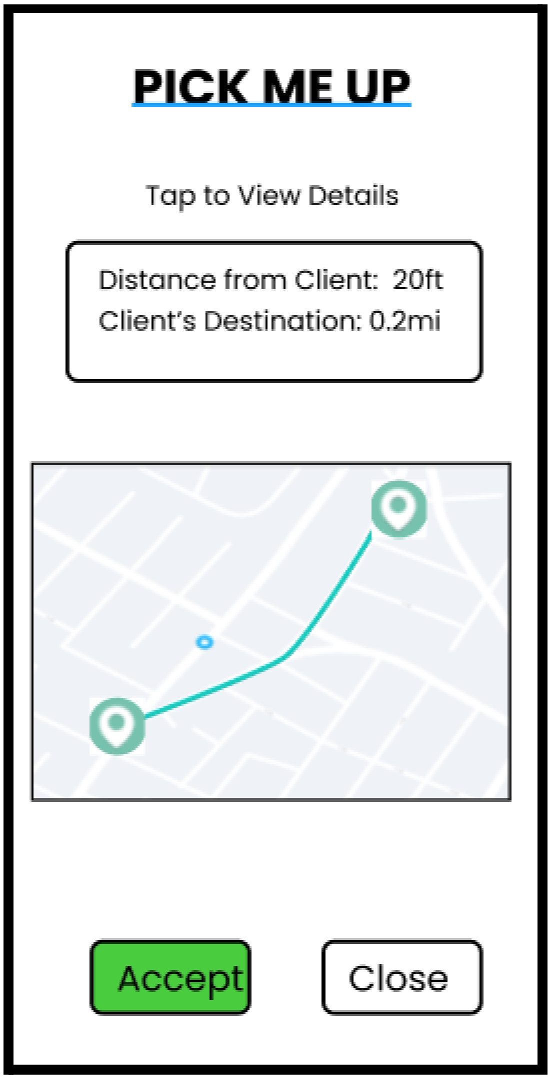
Tap to View Details

Distance from Client: 20ft
Client's Destination: 0.2mi

Distance from Client: 18ft
Client's Destination: 0.4mi

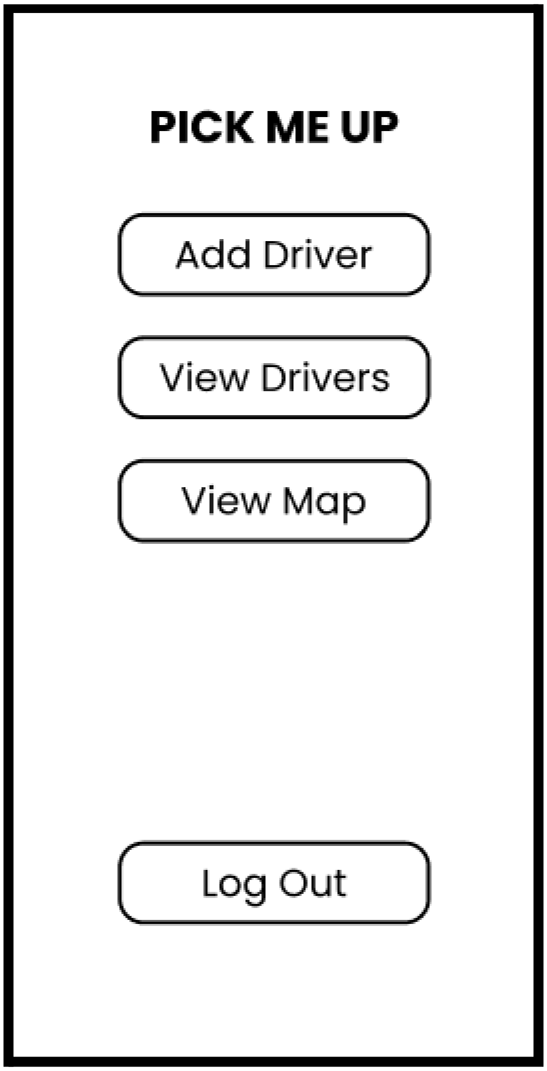
Distance from Client: 51ft
Client's Destination: 61ft

The diagram shows that a Pick up request can be clicked on to see more information about the Request. Clicking on a Pick up Request option will show another page with more information and a map.

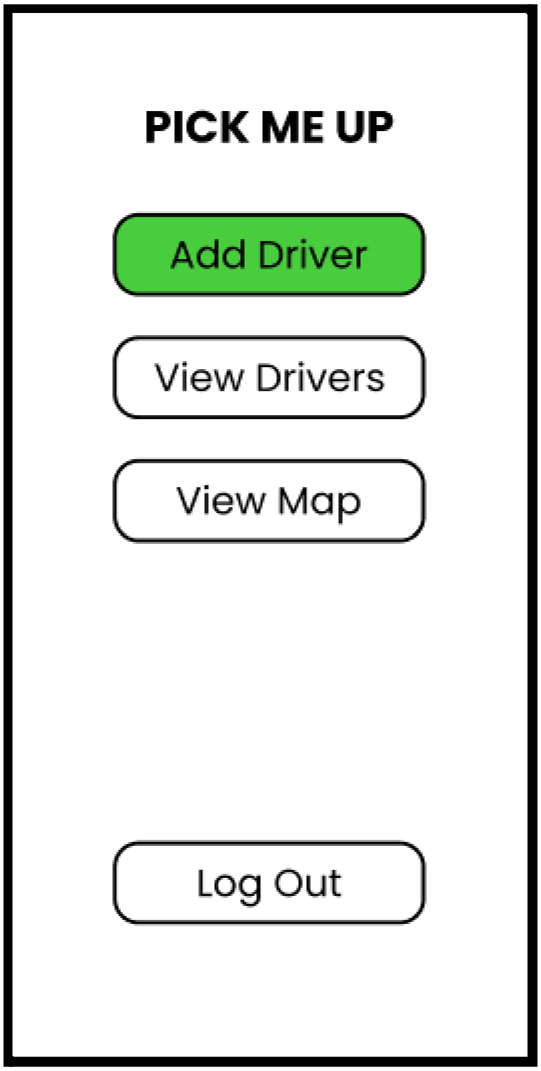


Distance from Client: 20ft
Client's Destination: 0.2mi

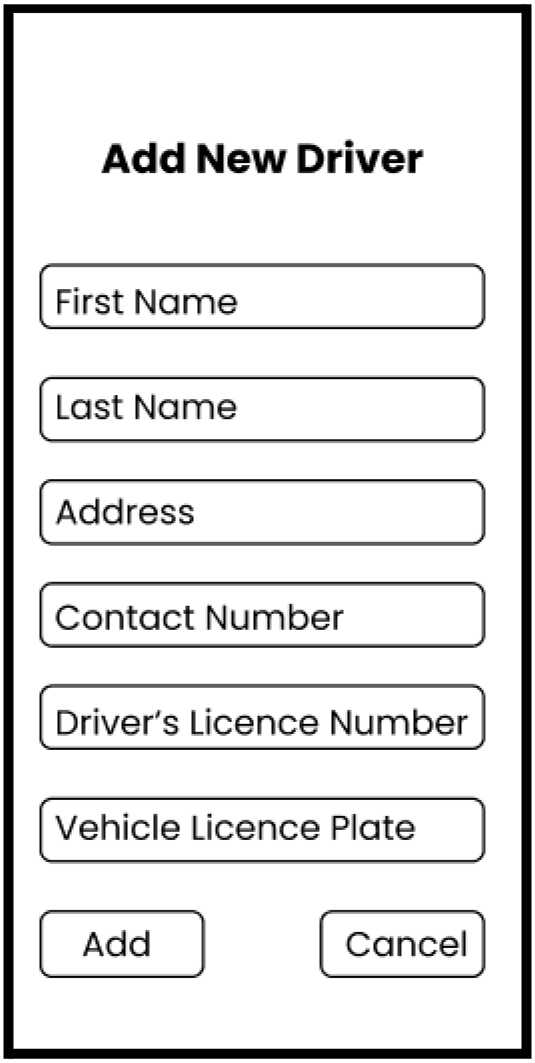
This Diagram shows the page after a user decides to view the Request list and choses a request.This map will show the client distance from the current user location and there is an option for the user to click to accept this request.



This Diagram shows the interface of the homepage of the administrator. The administrator main job is to manage the Drivers information on the database.

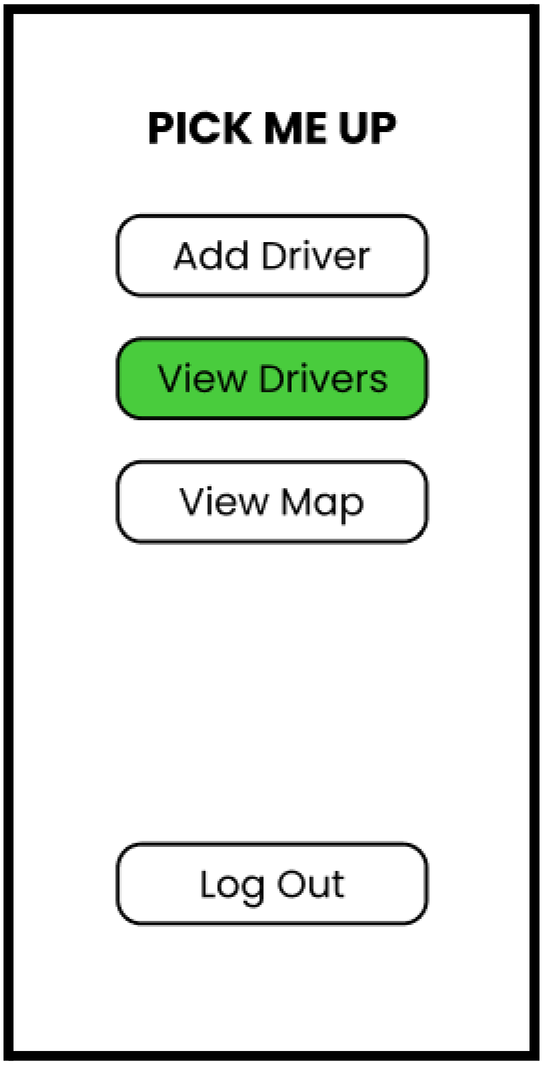


This Diagram shows the selection of the option add drivers, where will allow the administrator to enter another driver into the database. Choosing this option will carry the user to another page.

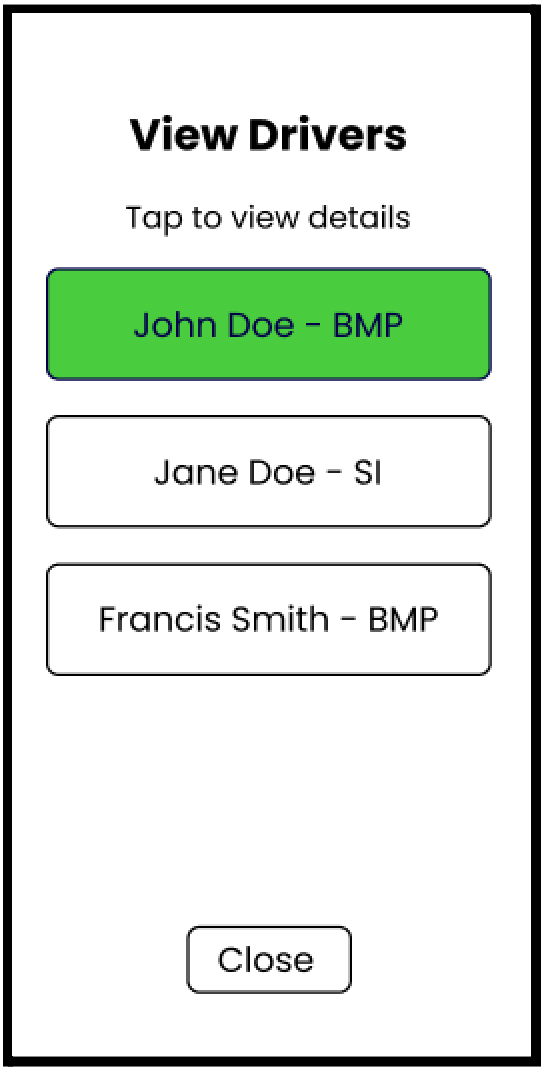


This diagram shows what information is needed to enter a new driver into the database.

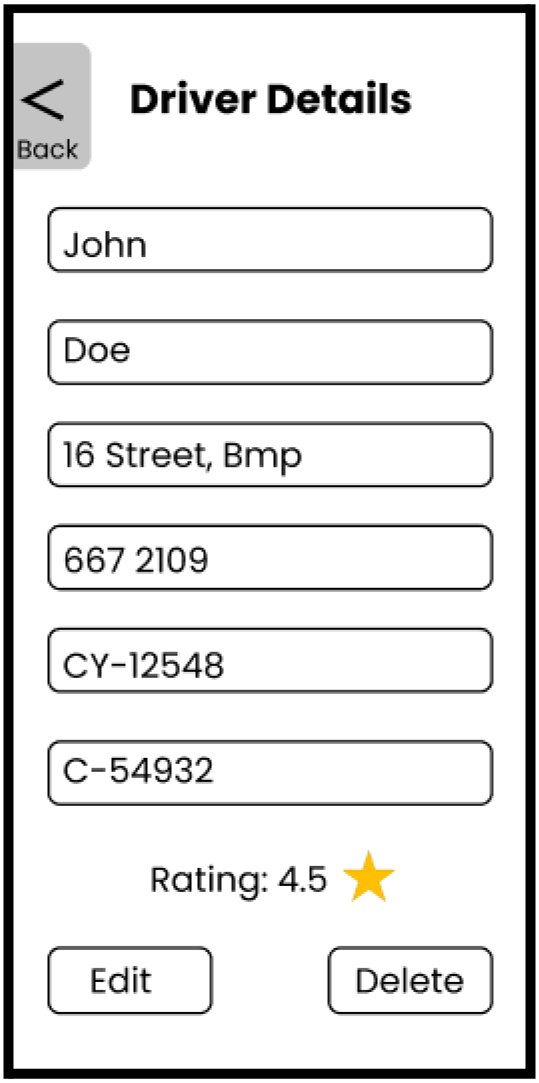
Clicking on the Add button will complete this action, you can also click cancel and it will return the user to the homepage.



This Diagram shows the option View Drivers being selected to view all the Drivers in the database. Clicking on this option will bring up another page to view all drivers in the database.



This Diagram shows the page after clicking the view driver option on the homepage. All drivers in the database are shown onto this list. You can click on a driver option to view their information on another page.



Driver Details

John

Doe

16 Street, Bmp

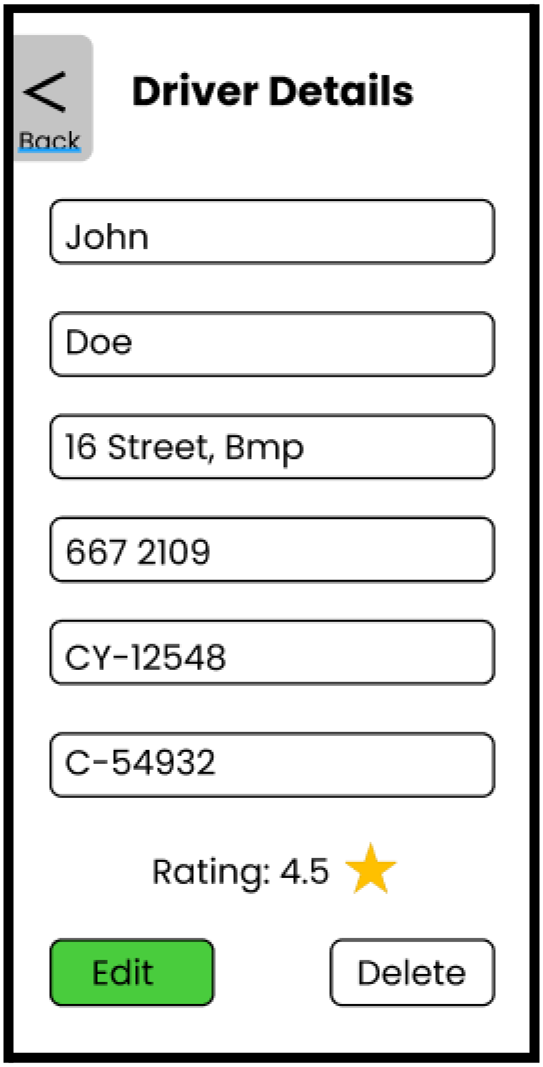
667 2109

CY-12548

C-54932

Rating: 4.5 ★

This Diagram shows a specific driver information after clicking on them in the View Driver list page. You are given 3 options to either edit the driver, delete the driver or go back to the homepage. The Driver rating is also shown on this page.



John

Doe

16 Street, Bmp

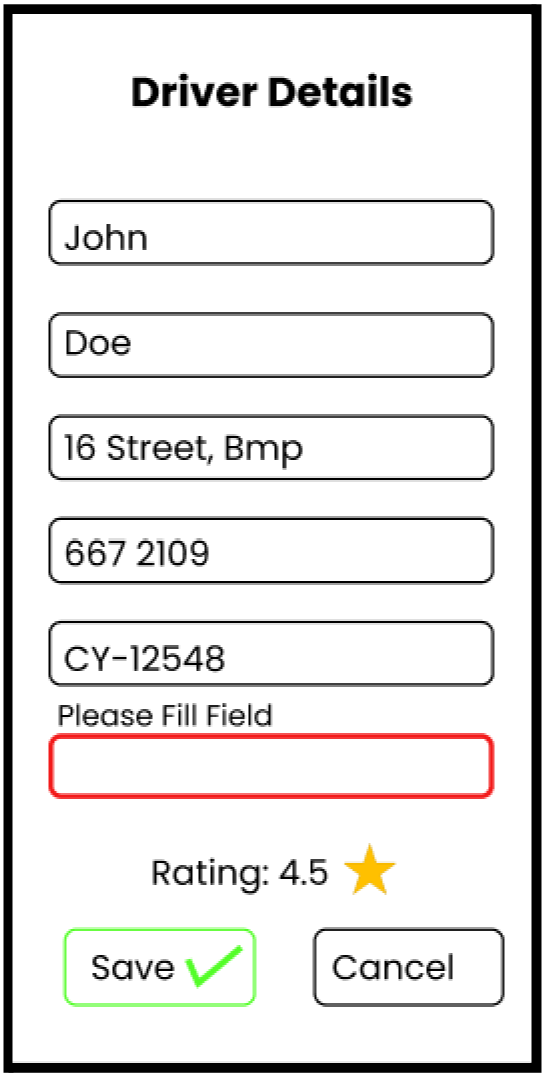
667 2109

CY-12548

C-54932

Rating: 4.5 ★

This diagram shows that the edit option is being clicked. This will bring up another page to edit the driver information.



Driver Details

John

Doe

16 Street, Bmp

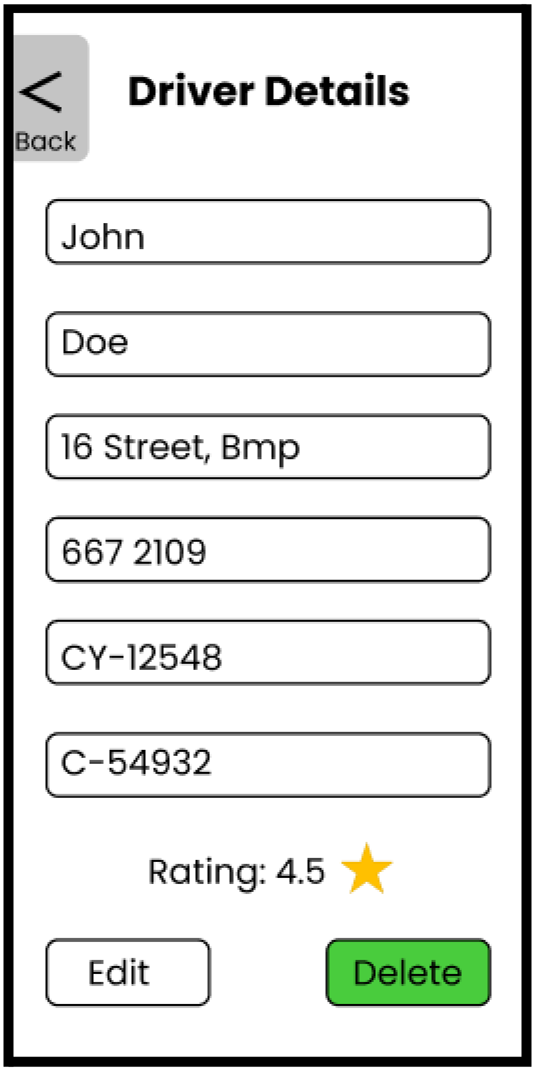
667 2109

CY-12548

Please Fill Field

Rating: 4.5 ★

This diagram shows the interface of how a driver information is edited. Every field must be filled out with valid data or the system will prompt an error message. You can then save the newly updated information or just cancel the editing.



Driver Details

John

Doe

16 Street, Bmp

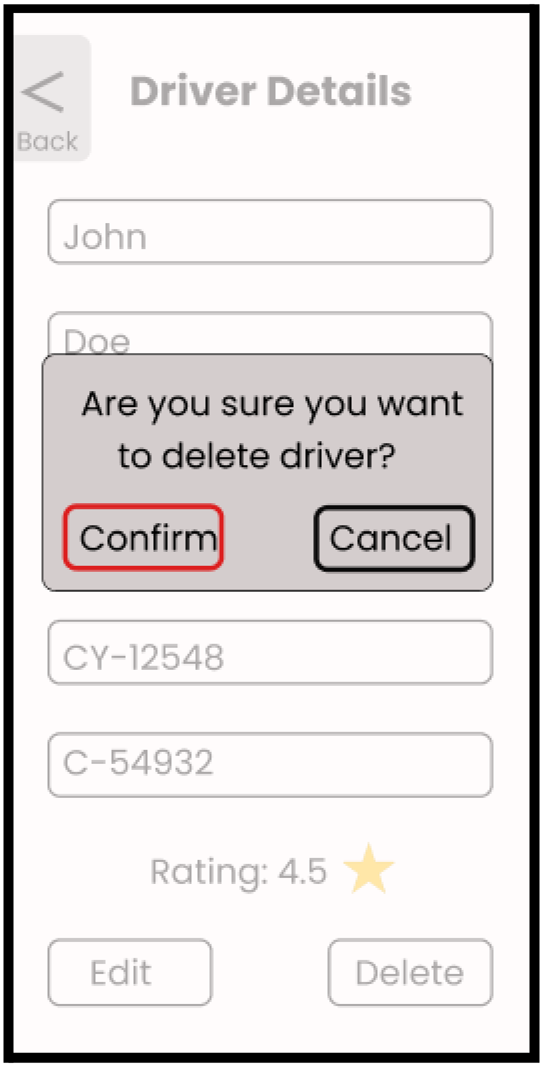
667 2109

CY-12548

C-54932

Rating: 4.5 ★

This diagram shows the delete option being selected. This will bring up a message box to confirm if the user wants to delete the driver information from the database.



Driver Details

John

Doe

Are you sure you want to delete driver?

CY-12548

C-54932

Rating: 4.5 ★

This Diagram shows the message box prompting the user after an attempt to delete the driver information from the database. The user can click confirm to complete this action or just press cancel to abort it.

When designing the system, we focus mostly on the ease of use aspect of a system. We incorporated touch interaction, where navigating the system is so straight forward because the system only has big clickable buttons with very descriptive labels on them. For example, to order a Pickup, it only takes a click of the Request button and a click on the map to decide your destination. We decided that signing up is a big issue in privacy and is also time consuming, so the minimum requirement to use our system is to leave on your GPS. The other main users we focus on are the drivers, the drivers can pick up any client they choose to pick up from a list or requests and we also provide the driver with basic information like the pickup spot, destination and a shortest path similar to services of google maps.

Design Testing

Test Cases 1 , 2 , 3 and 4 will be responsible to test the ability of the administrator to add drivers (UC-1),update drivers information (UC-3), view Taxi information (UC-5) and delete drivers information (UC-2) from the database.

Test Case 1	**TC-1**
Use case being used:	UC-1- Add Driver
Criteria for Success/Fail:	Test is a success if the administrator can add a new driver with all required information.
Input Data:	Text input, Date, integer(valid) | Char/double(invalid)
Test Procedure:	Expected Result:
Create a connection with database Step 1: Call function "add_driver(fullname, dateofbirth, socialsecurity, licenseplate)" with invaild fullname data. Step 2: Call function "add_driver(fullname, dateofbirth, socialsecurity, licenseplate)" with invaild dateofbirth data. Step 3: Call function "add_driver(fullname, dateofbirth, socialsecurity, licenseplate)" with invaild socialsecurity data. Step 4: Call function "add_driver(fullname, dateofbirth, socialsecurity, licenseplate)" with invaild licenseplate data.	**Success** **Fail** - Display an error message for invalid data: Prompts the user for another input **Fail** - Display an error message for invalid data: Prompts the user for another input. **Fail** - Display an error message for invalid data: Prompts the user for another input. **Fail** - Display an error message for invalid data: Prompts the user for another input.

Step 5: Call function “add_driver(fullname, dateofbirth, socialsecurity, licenseplate)” with valid data for all parameters.	**Success-** A new driver is added to the database.

Test Case 2	**TC-2**
Use case being used:	UC-2-Delete Driver
Criteria for Success/Fail:	The test is a success if the administrator can remove driver information from the database.
Input Data:	none
Test Procedure:	Expected Result:
Create a connection with database Step 1: Call function “deleteDriver()”	**Success** **Success-** Selected driver is deleted from the database

Test Case 3	**TC-3**
Use case being used:	UC-3- Update Driver
Criteria for Success/Fail:	The test is a success if the administrator can update the driver information from the database.
Input Data:	Text input, Date, integer(valid) , Char/double(invalid)

Test Procedure:	Expected Result:
Create a connection with database	**Success**
Step 1: Call function “updateDriver(fullname, dateofbirth, socialsecurity, licenseplate)” with invaild fullname data.	**Fail** - Display an error message for invalid data: Prompts the user for another input
Step 2: Call function “updateDriver(fullname, dateofbirth, socialsecurity, licenseplate)” with invaild dateofbirth data.	**Fail** - Display an error message for invalid data: Prompts the user for valid input data.
Step 3: Call function “updateDriver(fullname, dateofbirth, socialsecurity, licenseplate)” with invaild socialsecurity data.	**Fail** - Display an error message for invalid data: Prompts the user for valid input data.
Step 4: Call function “updateDriver(fullname, dateofbirth, socialsecurity, licenseplate)” with invaild licenseplate data.	**Fail** - Display an error message for invalid data: Prompts the user for valid input data.
Step 5: Call function “updateDriver(fullname, dateofbirth, socialsecurity, licenseplate)” with valid data in all parameters.	**Success-** Driver information is updated in the database.

Test Case 4	**TC-4**
Use case being used:	UC-5 view Taxi
Criteria for Success/Fail:	The test is a success if the administrator can view the taxi driver information.

Input Data:	Text input (valid), char/integer (invalid)
Test Procedure:	Expected Result:
Create a connection with database Step 1: Call function “viewTaxi (drivername)” with empty parameters. Step 2: Call function “viewTaxi(drivername)” with invalid drivername data. Step 3: Call function “viewTaxi(drivername)” with valid drivername data.	**Success** **Success-** Displays a list of all Driver information. **Fail-** Display an empty list. Prompts the user to input valid data. **Success**- Displays the Driver information

Test case 5 will test the precision of location tracking of the client and the availability of service when the client requests a “Pickup”. (UC-6) Request Pick-up does this with the help of (UC-13) viewPickup

Test Case 5	**TC-5**
Use case being used:	UC-6 & UC-13 Request Pick-up & viewPickup
Criteria for Success/Fail:	The test is a success if the user can request a service.
Input Data:	double(valid), integer/char (invalid)
Test Procedure:	Expected Result:
Step 1: Call function “RequestPickup()” Step 2: Call function “viewPickup()”.	**Success-** A pickup order is created for the client. **Success**- The client pickup location has been set to the order.

Test Case 6 will test the functionality of a driver selecting and accepting to provide services to a client. (UC-18) accept Request does this with the help of (UC-17) List Request and (UC-13) View Pick Up.

Test Case 6	**TC-6**
Use case being used:	UC-18 & UC-17& UC-13-Accept Request & ListRequest & viewPickUp
Criteria for Success/Fail:	The test is a success if the driver can accept a Pickup Request.
Input Data:	Touch, Integer(valid), Char(invalid)
Test Procedure:	Expected Result:
Step 1: Call function “ListRequest()”. Step 2: Call function “viewPickUp()”. Step 3: ”Call function “AcceptRequest()”.	**Success-** A list of Pickup Requests is shown. **Success-** Displays the location of the selected client. **Success-** The selected Request is marked as taken.

Test Case 7 will test the functionality of the user to sign into the system and Display an interface depending on the authority of the account. (UC-4) Login is incharge of this Test case.

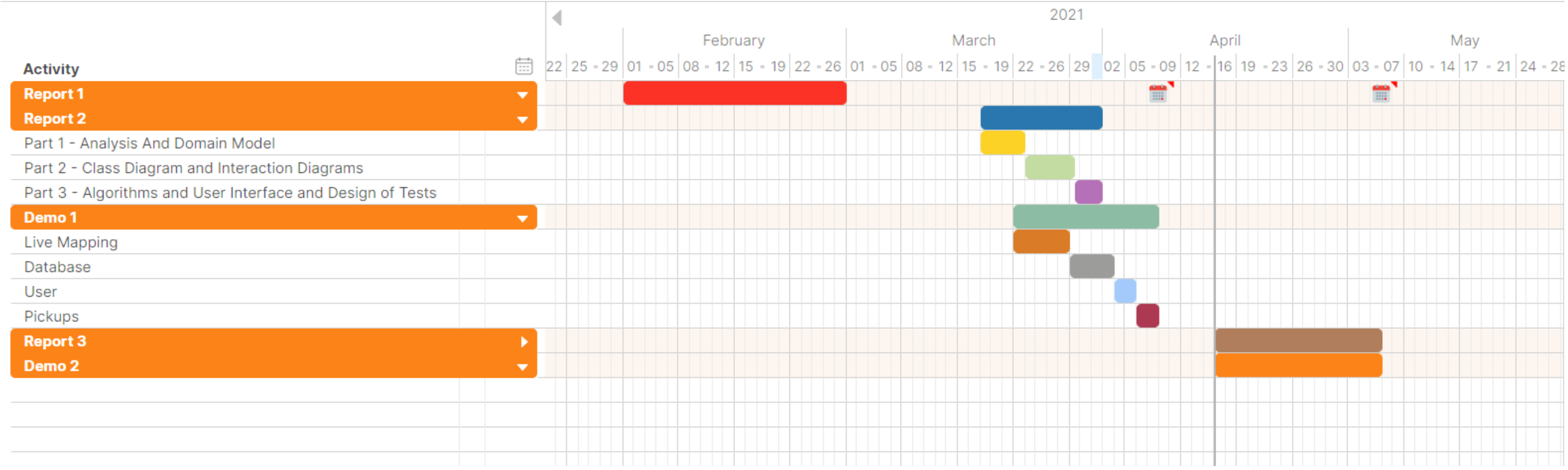
Test Case 7	**TC-7**
Use case being used:	UC-4 Login
Criteria for Success/Fail:	The test is a success if the user can login to their respective home page.
Input Data:	Text Input (valid)
Test Procedure:	Expected Result:
Step 1: Call function “login(Username, password)” with invalid username data.	**Fail-** Display an error message for invalid input data. Prompts the user to input valid

Step 2: Call function “login(Username, password)” with invalid password data. Step 3: Call function “login(username, password)” with valid data for all parameters.	data. **Fail-** Display an error message for invalid input data. Prompts the user to input valid data. **Success-** Display Homepage based on credentials.

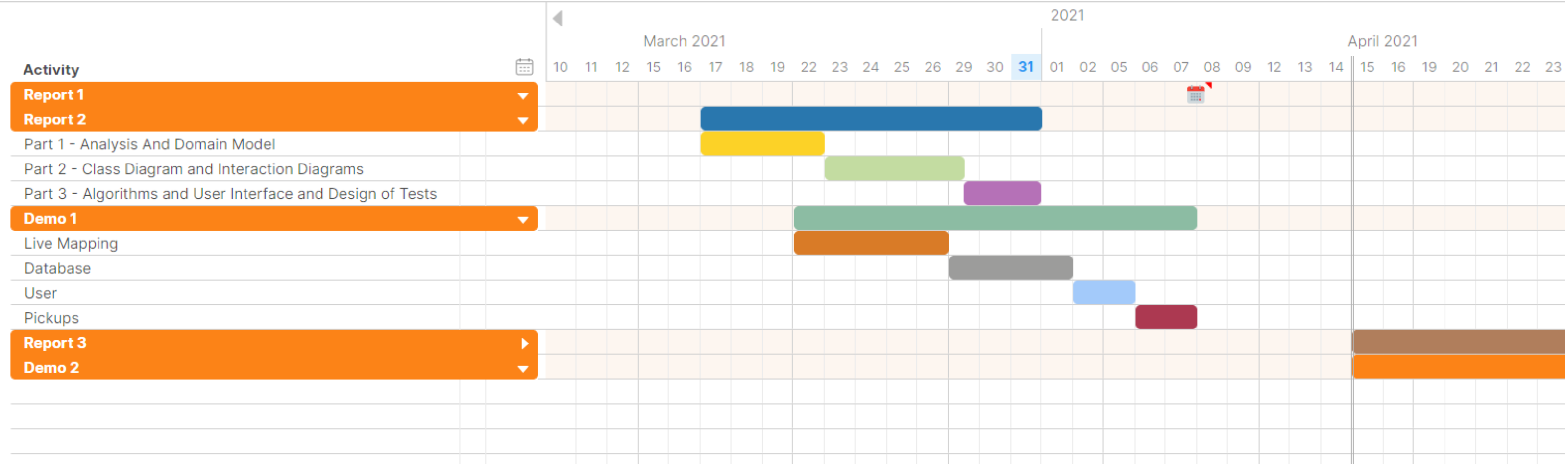
Progress Report

Use Case	Untouched	Developed	Coded	Implemented	Tested
UC-1	X				
UC-2	X				
UC-3	X				
UC-4		X			
UC-5		X			
UC-6		X			
UC-7	X				
UC-8	X				
UC-9			X		
UC-10			X		
UC-11			X		
UC-12		X			
UC-13		X			
UC-14	X				
UC-15	X				
UC-16			X		
UC-17	X				
UC-18	X				
UC-19			X		

Plan of Work



Showing Weekly Plan of Work



Showing Daily Plan of Work between 10 March and April 23.
Can be seen dynamically at
https://plan.tomsplanner.com/public/se-pick-me-up-group-one-plan-of-word.

Breakdown of Responsibilities

Front End

Concept	Programmer
Map Maker	Osmer Escarraga
Page Maker	Osmer Escarraga
Popup Maker	Osmer Escarraga
Responsiveness	Osmer Escarraga
Design	Osmer Escarraga

Back End

Concept	Programmer
Controller	Pablo Cawich
Database	Pablo Cawich
Users	Pablo Cawich
Pickup Requests	Pablo Cawich
Routes	Pablo Cawich

Integration

Pablo Cawich will be responsible for integrating the front end designs and themes to work correctly with the backend services. Pablo will be responsible for ensuring that all services work correctly in conjunction with each other as well as testing that integration has been done successfully.

Effort Breakdown Table

Task	Team Member Names				
	Pablo Cawich	Hector Castellanos	Osmer Escarraga	Austin Shaw	Michael Sanchez
Domain Analysis	0%	0%	100%	0%	0%
Interaction Diagrams	0%	0%	0%	0%	100%
Class Diagram & Interface Specification	0%	0%	0%	100%	0%
Algorithm and Data Structures	100%	0%	0%	0%	0%
User Interface Design and Implementation	0%	100%	0%	0%	0%
Test Case Design	0%	100%	0%	0%	0%
Project Management and Plan of Work	100%	0%	0%	0%	0%

References

https://magazine.impactscool.com/en/speciali/google-maps-e-la-teoria-dei-grafi/
https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/
https://plan.tomsplanner.com/public/se-pick-me-up-group-one-plan-of-word.