

# **Software Engineering Mini Project**

## **Stage - II**

### **Project Report**

Third Year Computer Engineering  
2021-22

Division: 1 Batch: T5

### **Group Members:**

1. Vishal Anap. (111903015)
2. Muddayya Swami. (111903047)

**Project Topic:** Taxi Management System

**Group No:** T5\_29

## Table of Contents:

Table Content	Page No.
1. Problem Statement .....	3
2. Objectives .....	3
3. Motivation .....	3
4. Summary of SRS .....	3
4.1 Introduction to SRS .....	4
4.2 Overall Description .....	4
4.3 External Interface Requirements .....	4
4.4 System Features .....	4
4.5 Other Non-functional Requirements.....	5
5. Entity Relationship Diagram.....	5
6. Unified Modelling Language(UML) Diagrams.....	6
7. Implementation.....	12
8. Testing.....	21
9. Conclusion.....	21
10. Future Scope.....	21
11. GitHub Repository link.....	22
12. References.....	22

## **PROBLEM STATEMENT:**

For various purposes people need to travel on the daily basis. And there are various mediums used for travelling. But getting the transportation medium at peoples convenience is a difficult thing to achieve. So, our problem statement is to build a web application for Taxi Management System which provide the easy platform for taxi booking.

## **OBJECTIVES:**

1. To create a location based reactive system where customers can book taxis that are in the proximity to their location and can plan a trip to new location.
2. To save time of both the customers and drivers during transit.
3. To provide excellent customer service, with instant response and easy payment gateway.

## **MOTIVATION:**

Transportation is the basic and one of the core needs of working people in our society. There are many public transports available although people avoid to travel from them due to rush, crowd, traffic congestion, unavailability of seats, lack of maintenance, delay, etc. And due to this pandemic situation lot of people used to plan their journey via taxis to avoid contact with infected people. Also, we can find lot of people searching for a taxi in their locality as well as same case with taxi drivers, searching for customers. This motivated us, to develop an efficient and easy to use taxi management system platform which will provide solutions to all problems.

## **SUMMARY OF SRS:**

The SRS stands for Software Requirements Specification. It includes the introduction of system like purpose, product scope, document conventions, references. It also gives overall description, product perspective, operating environment, design and implementation constraints, assumptions, and dependencies. The User Interfaces, hardware interfaces, software interfaces, communication interfaces are included. Also, system features and other non-functional requirements are listed.

- **Introduction to SRS:**

The purpose of SRS is to describe the features of Taxi Management System (TMS), so as to serve as a guide to developers and act as a software validation document for clients. It is intended to completely specify the functionalities which system provides. It describes the product scope in the form of functionalities provided by TMS like, register, sign in, send and receive booking requests, give ratings and feedbacks, easier payment gateway, etc.

- **Overall Description:**

The product perspective is to build a platform where people can book taxi to plan their trip based on their locations and to create responsive, user-friendly and easy to use interface for application which will be available for 24 X 7. The product functions include taxi booking, viewing trip history, making payment, submitting ratings and feedbacks, etc. The user classes for the system are customers and drivers with basic knowledge of the system. The operating environment includes minimum 4 GB RAM, 120 GB free hard drive space and compatible processor.

- **External Interface Requirements:**

The user interfaces encompasses, GUI which is build using React frontend framework and JSX model. This UI will be responsive to size of screen, ensure user get proper messages informing them about the status of task. The hardware interfaces should support TCP/IP protocols. The software interfaces require MySql, MERN stack, compatible processor. The communicating interfaces includes support of HTTP protocol for querying, NodeJS for communication with database.

- **System Features:**

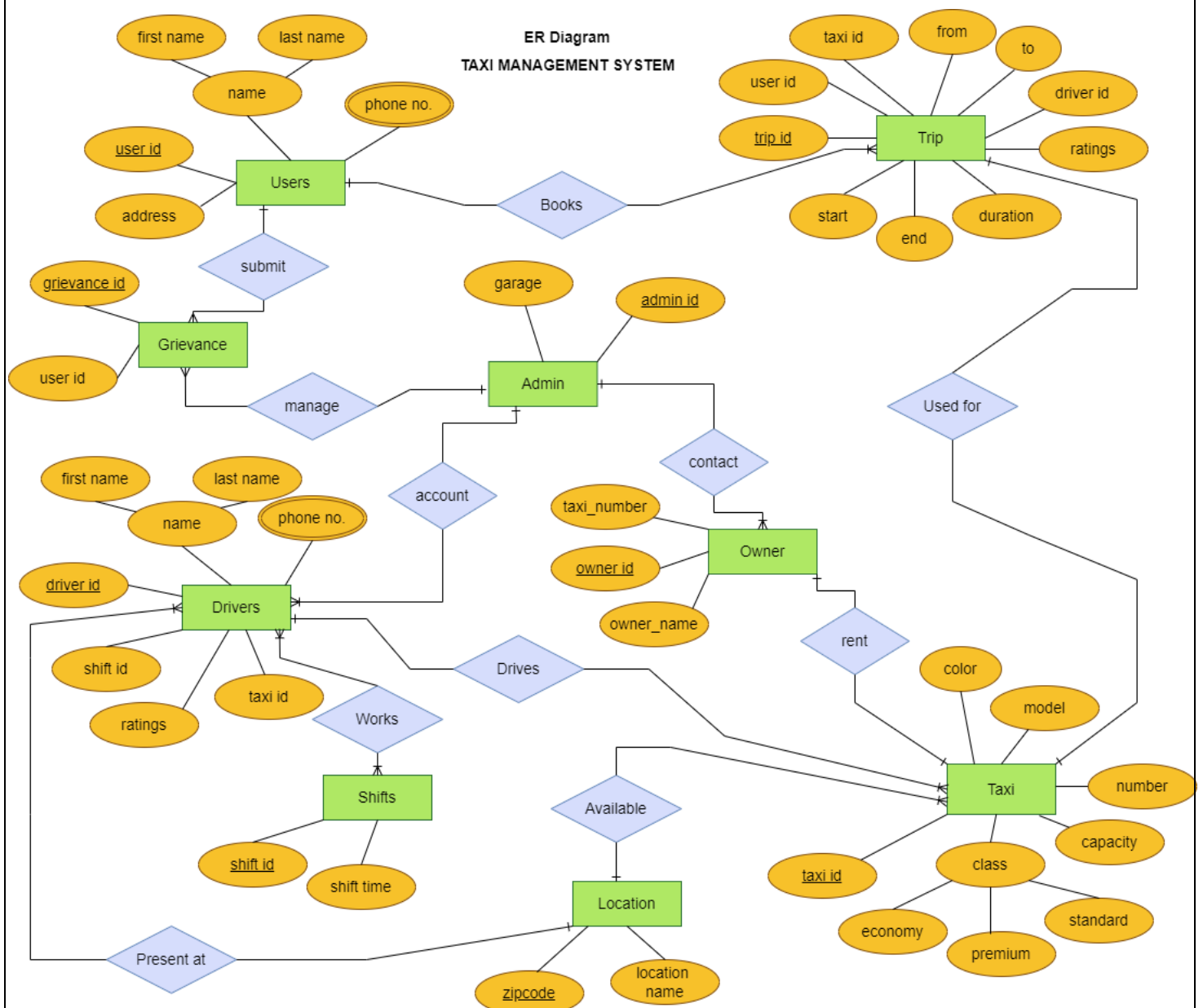
1. Register/ Sign in functionality
2. Sending and Receiving trip requests.
3. View booking status and trip history
4. Payment gateway with ratings and grievances.
5. Checking shifts and taxi details.
6. Updating location and choice to accept trip requests
7. Maintaining taxi's owner information and garage status.
8. Creating driver accounts and viewing feedbacks.

- **Other Non-functional Requirements:**

These requirements include reliability, portability, compatibility, proper and functional website, secure payment gateway, alerts based on activities with respect to system.

## ENTITY RELATIONSHIP DIAGRAM:

ER diagrams are composed of three main elements: entities, attributes, and relationships. Entities - typically displayed in a rectangle. Attributes - displayed in a circle or an oval, the attributes refer to the characteristics of an entity. Relationships - illustrate how two or more entities interact with each other.

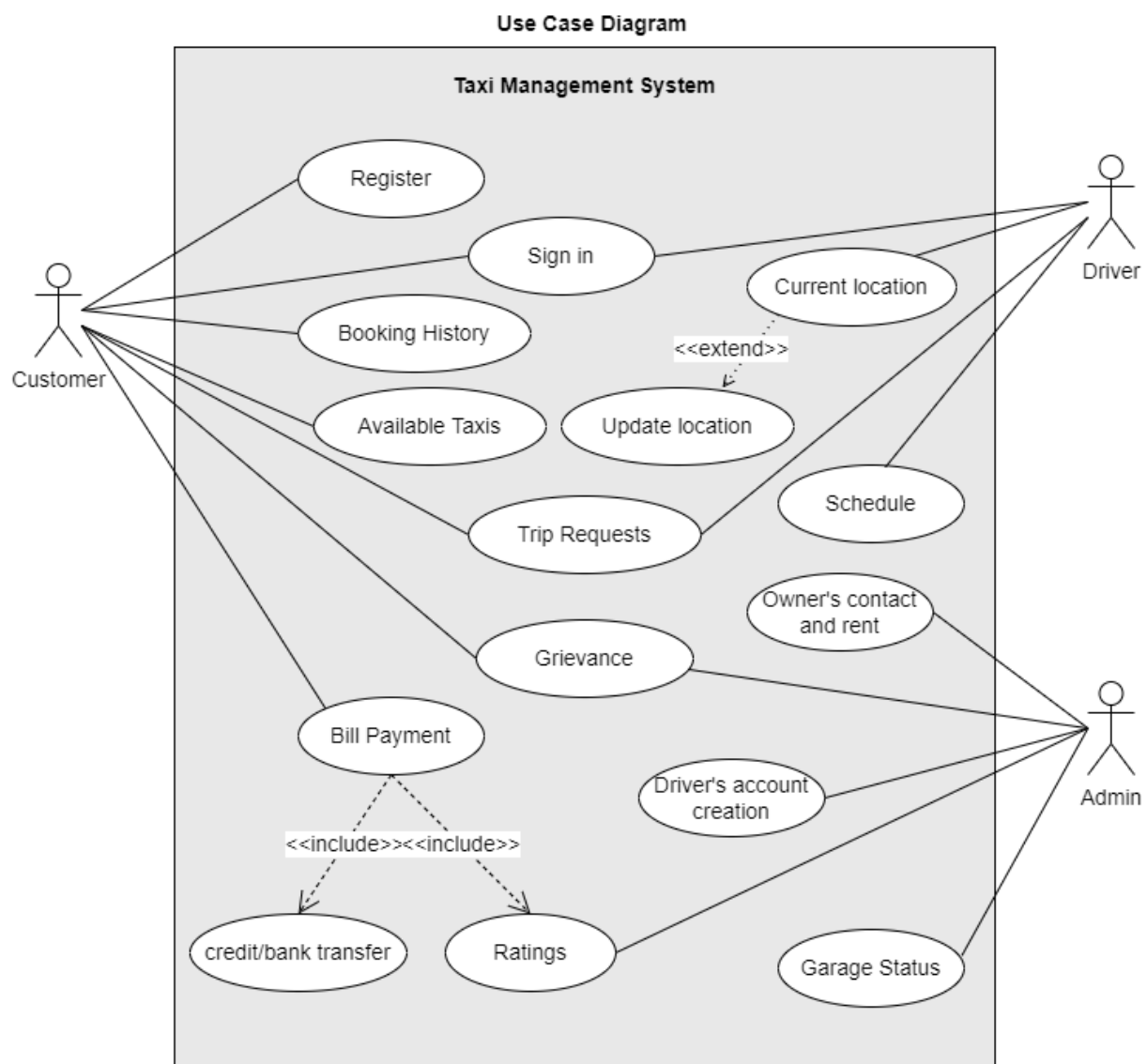


## UML AND EXPLANATION:

### 1. Use Case Diagram:

It is used to represent dynamic behaviour of System. It encapsulates the system's functionality by incorporating use cases, actors, and their relationship. It depicts external view of the system and gathers the system needs.

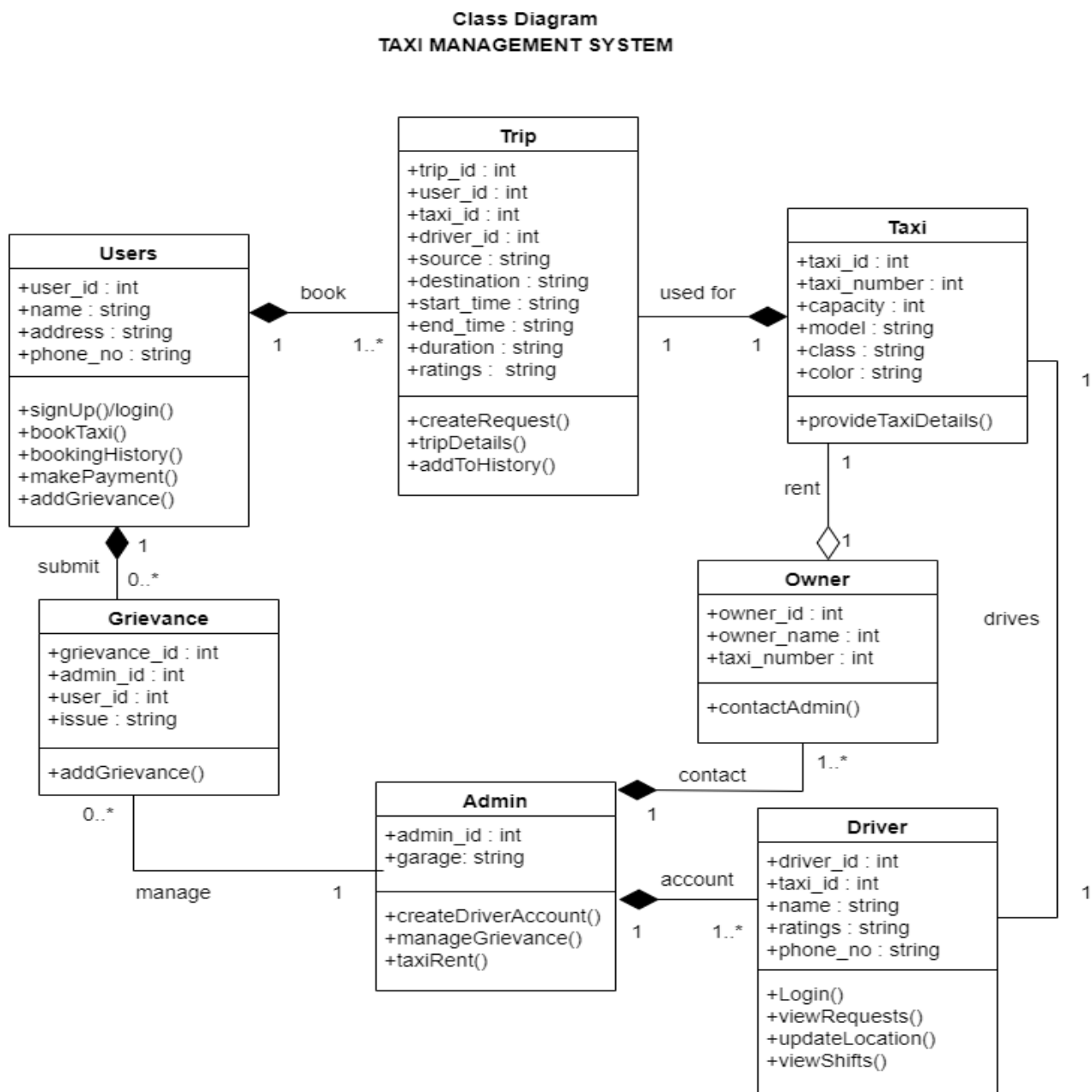
System's boundaries represented using a rectangle, use cases using ovals. Actors are the users of a system. An "include" relationship indicates that one use case is needed by another in order to perform a task. An "extend" relationship indicates alternative options under a certain use case.



## 2. Class Diagram:

The class diagram depicts a static view of an application. A class consist of its objects and also may inherit from other classes. It Analyse and design static view of an application and also describes the major responsibilities of system.

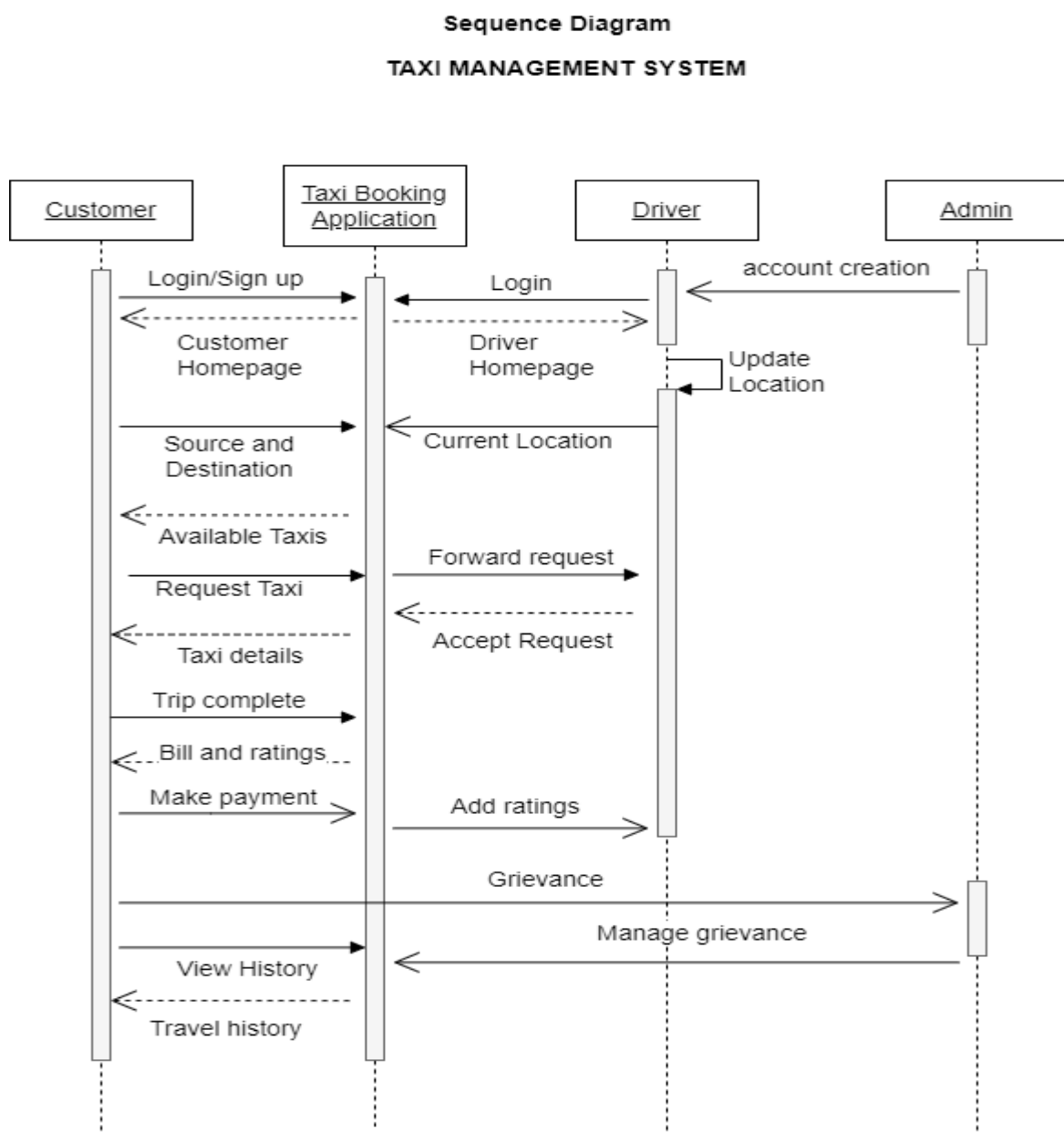
A class represent a concept which encapsulates state(attributes) and behavior (operations). Each attribute has a type. Each operation has a signature. The class name is the only mandatory information.



### 3. Sequence Diagram:

The sequence diagram represents the flow of messages in the system. The purpose of Sequence diagram is to model high level interaction among active objects within a system and to model interaction among objects inside a collaboration realizing the use case.

A sequence diagram is structured in such a way that it represents a timeline which begins at the top and descends gradually to mark the sequence of interactions. Each object has a column and the messages exchanged between them are represented by arrows.

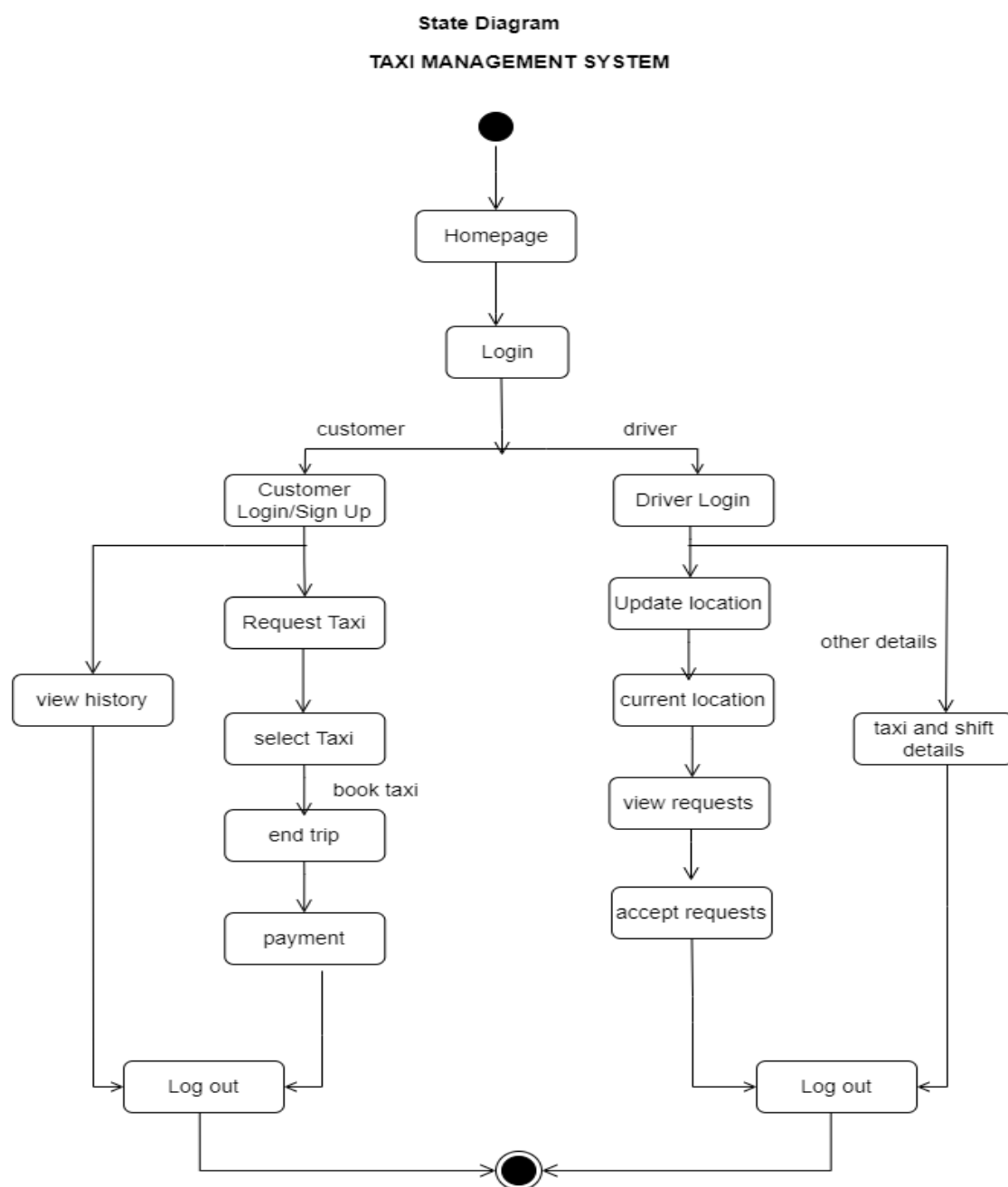




#### 4. State Diagram:

A state diagram is used to represent the condition of the system or part of the system at finite instances of time. It captures the software system's behaviour. It models the behaviour of a class, a subsystem, a package, and a complete system.

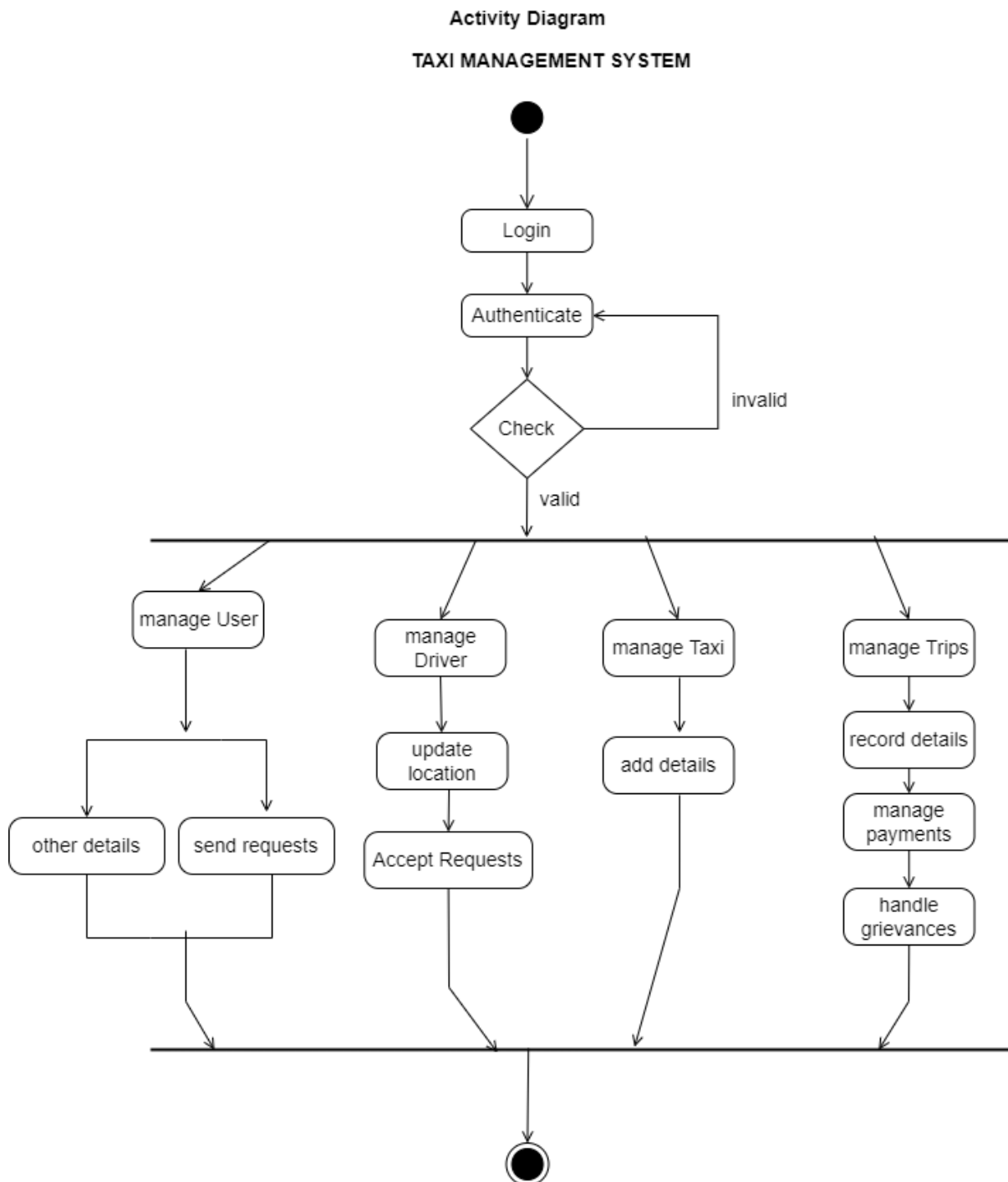
Its initial state usually marked by a solid black circle, middle states are portrayed as boxes with rounded corners, transitions are external straight lines each with an arrow at one end connect various pairs of boxes and final state is portrayed as a large black dot with a circle around it.



## 5. Activity Diagram:

The Activity diagram is used to demonstrate the flow of control within system. It models concurrent and sequential activities. It helps in envisioning the workflow from one activity to another.

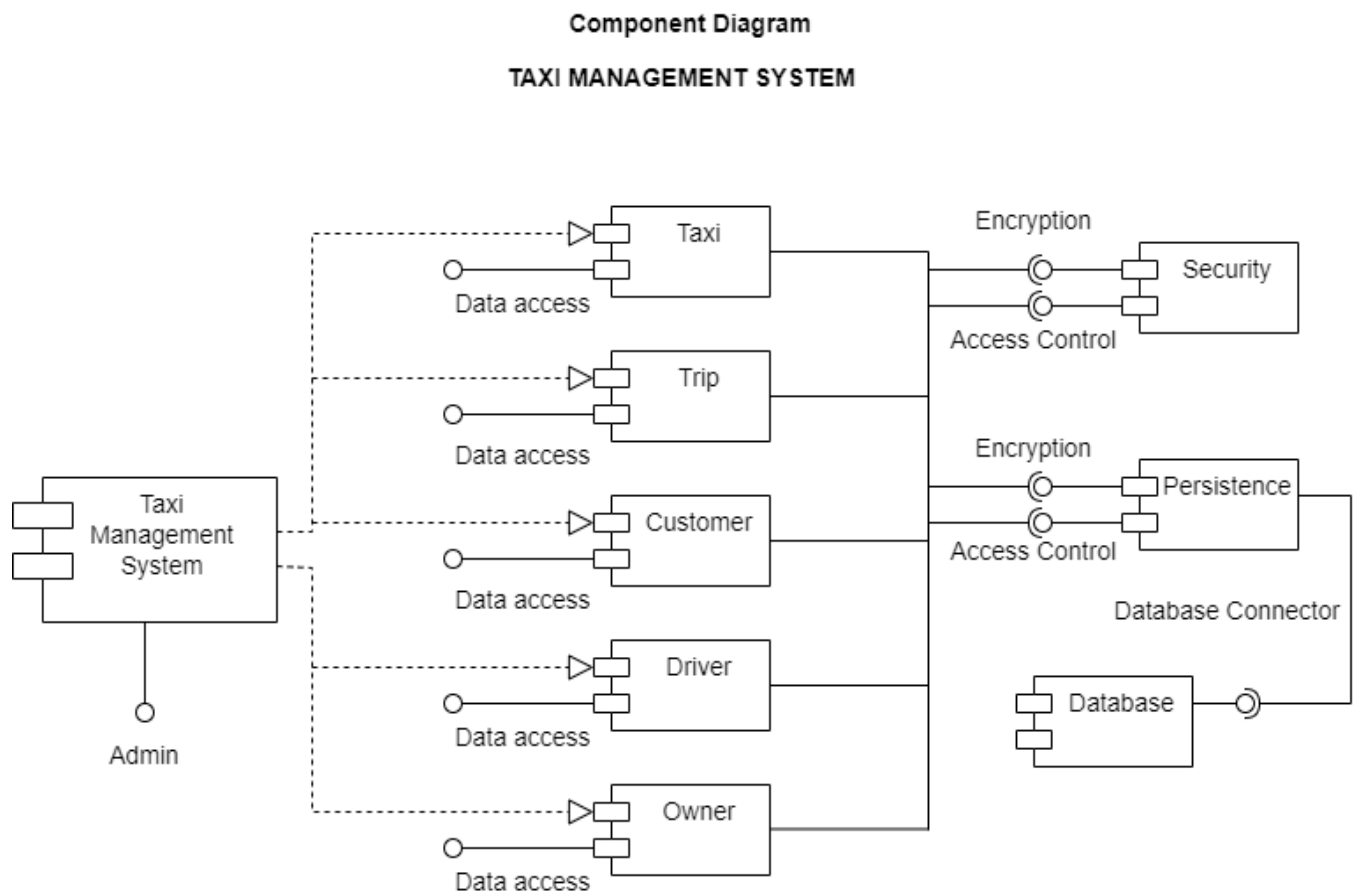
It consists of action, decision node, control flows, start and end node as a components.



## 6. Component Diagram:

The component diagram is used to break down a large object-oriented system into the smaller components. It models the physical view of system. The purpose of Component diagram is to explore each component of system and to depicts the relationship and organization of components.

Its component symbol represent an entity required to execute a function. Node symbol represents hardware or software objects. Port symbol specifies a separate interaction point between the component and the environment. Dependency symbol shows that one part of your system depends on another.



# IMPLEMENTATION:

## 1. Home Page:

Activities Visual Studio Code Apr 8 02:00

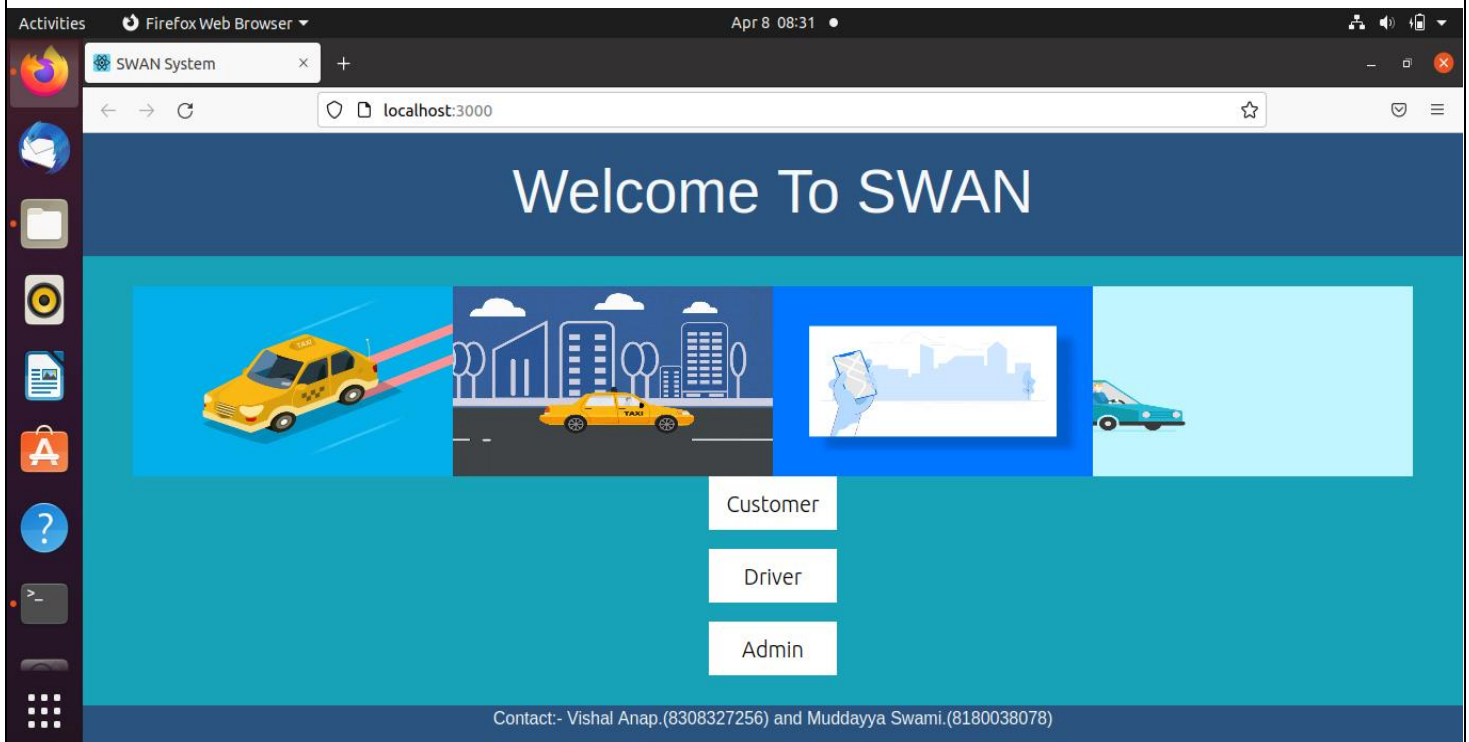
landingPage.js - swan - Visual Studio Code

```
File Edit Selection View Go Run Terminal Help

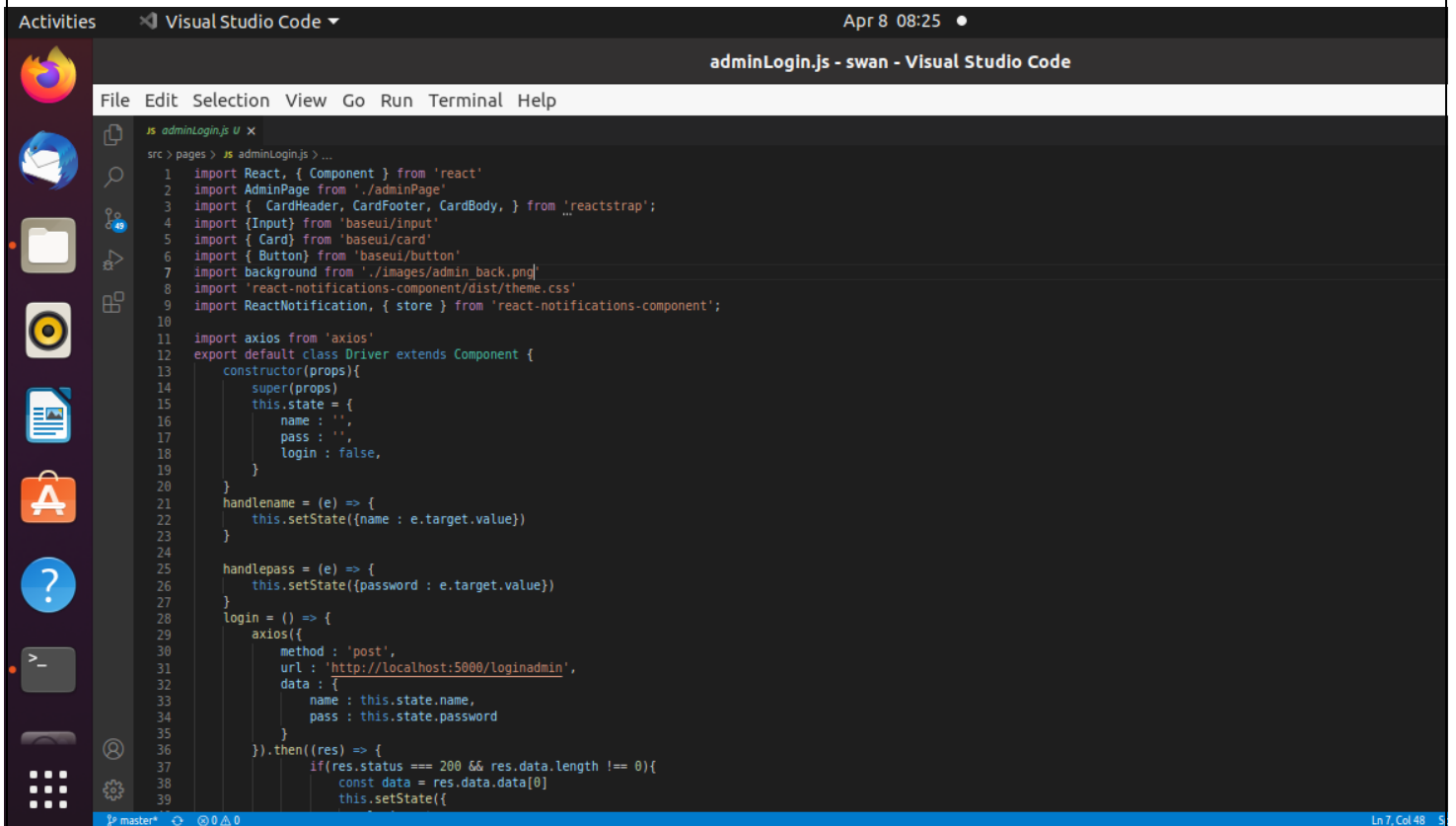
src > pages > landingPage.js > Landing > render

14
15 handledriver = () => {
16   this.setState({type: "driver"})
17 }
18 handleadmin = () => {
19   this.setState({type: "admin"})
20 }
21 render() {
22   console.log(this.state)
23   if (this.state.type === "customer") {
24     return <Redirect to="/customer" push></Redirect>
25   }
26   else if (this.state.type === "driver") {
27     return <Redirect to="/driver" push></Redirect>
28   }
29   else if (this.state.type === "admin") {
30     return <Redirect to="/admin" push></Redirect>
31   }
32   // [this.state.type === "driver" && <Redirect to="/driver"></Redirect>]
33   return
34 }
35 <div>
36   <div style={{ display: 'flex', flexDirection: 'row', justifyContent: 'center', alignItems: 'center', height: '28vh', backgroundColor: '#2A547F' }} >
37     <h1 className="display-4 text-light">Welcome To SWAN</h1>
38   </div>
39   <div style={{ display: 'flex', flexDirection: 'row', justifyContent: 'center', alignItems: 'center', height: '73vh' }} className="bg-info">
40     <div style={{ display: 'flex', flexDirection: 'row', justifyContent: 'center', alignItems: 'center' }}>
41       <img alt="landing" src=(require("./images/ing 1.gif")) height="170px" width="300px" ></img>
42       <img alt="landing" src=(require("./images/ing 2.gif")) height="170px" width="300px" ></img>
43       <img alt="landing" src=(require("./images/ing 3.gif")) height="170px" width="300px" ></img>
44       <img alt="landing" src=(require("./images/ing 4.gif")) height="170px" width="300px" ></img>
45     </div>
46     <div style={{ display: 'flex', flexDirection: 'column' }} >
47       <Button kind="primary" onClick={this.handlecust} className="lead" margin="20px">Customer</Button>
48       <hr className="my-2" />
49       <Button kind="primary" onClick={this.handledriver} className="lead">Driver</Button>
50       <hr className="my-2" />
51       <Button kind="primary" onClick={this.handleadmin} className="lead">Admin</Button>
52     </div>
53   </div>
54   <div style={{ display: 'flex', flexDirection: 'row', justifyContent: 'center', alignItems: 'center', height: '7vh', backgroundColor: '#2A547F' }} >
55     <p className="display-5 text-light">Contact:- Vishal Anap.(8308327256) and Muddayya Swami.(8180038078)</p>
56   </div>
57 </div>
```

Ln 48, Col 25 Spaces: 4

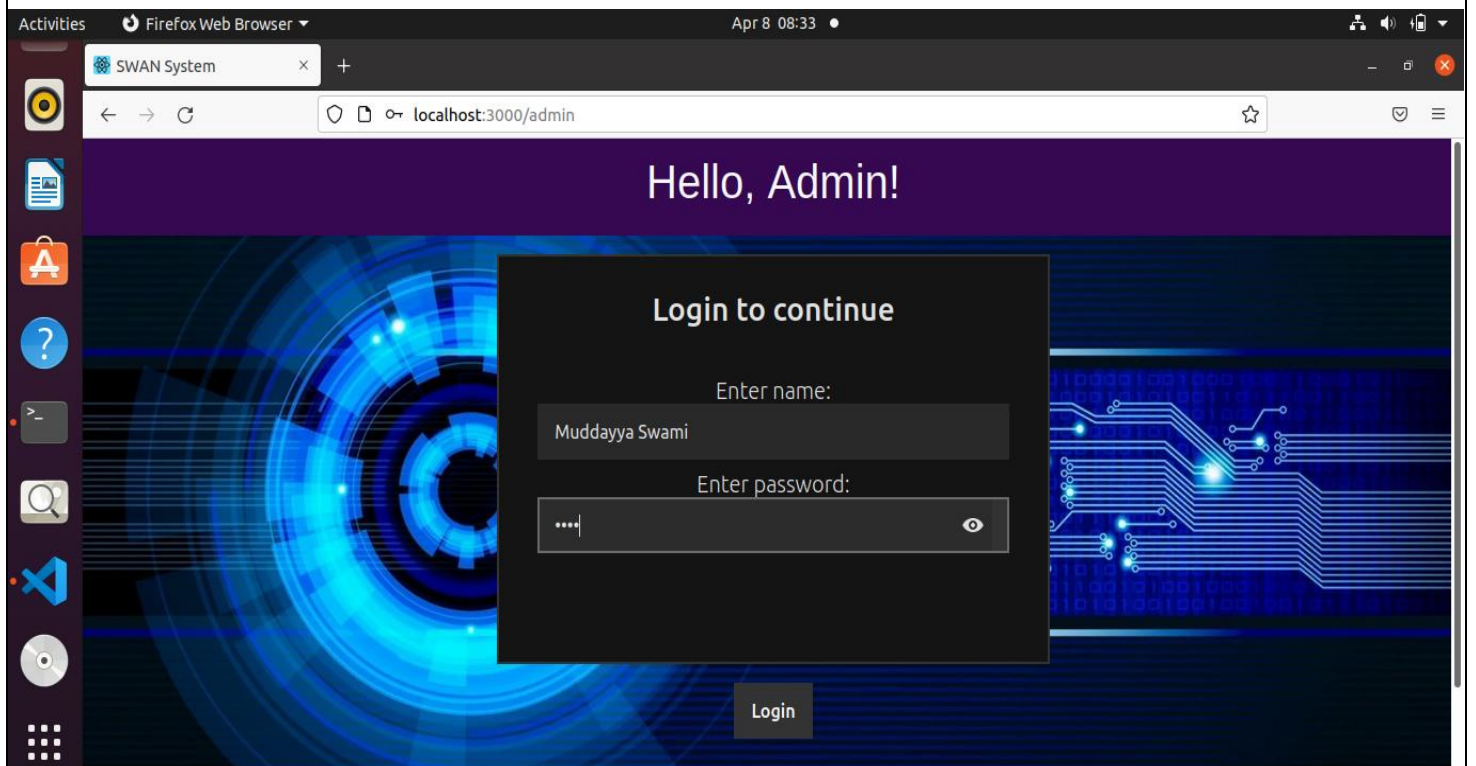


## 2. Admin Login:



The screenshot shows the Visual Studio Code editor with the file `adminLogin.js` open. The code is a React component named `Driver` that handles admin login. It uses `React`, `AdminPage`, `reactstrap` components, `baseui` components, `axios` for API calls, and `react-notifications-component` for notifications. The component has a state with `name`, `pass`, and `login` properties. It includes handlers for `handleName`, `handlePass`, and `login`. The `login` function sends a POST request to `http://localhost:5000/loginadmin` with the user's name and password. If the response status is 200 and the data length is not zero, it sets the state to `login: true` and displays a notification.

```
1 import React, { Component } from 'react'
2 import AdminPage from './adminPage'
3 import { CardHeader, CardFooter, CardBody, } from 'reactstrap';
4 import { Input } from 'baseui/input'
5 import { Card } from 'baseui/card'
6 import { Button } from 'baseui/button'
7 import background from './images/admin back.png'
8 import 'react-notifications-component/dist/theme.css'
9 import ReactNotification, { store } from 'react-notifications-component';
10
11 import axios from 'axios'
12 export default class Driver extends Component {
13   constructor(props){
14     super(props)
15     this.state = {
16       name: '',
17       pass: '',
18       login: false,
19     }
20   }
21   handleName = (e) => {
22     this.setState({name: e.target.value})
23   }
24
25   handlePass = (e) => {
26     this.setState({password: e.target.value})
27   }
28   login = () => {
29     axios({
30       method: 'post',
31       url: 'http://localhost:5000/loginadmin',
32       data: {
33         name: this.state.name,
34         pass: this.state.password
35       }
36     }).then((res) => {
37       if(res.status === 200 && res.data.length !== 0){
38         const data = res.data.data[0]
39         this.setState({
```



### 3. Customer Login:

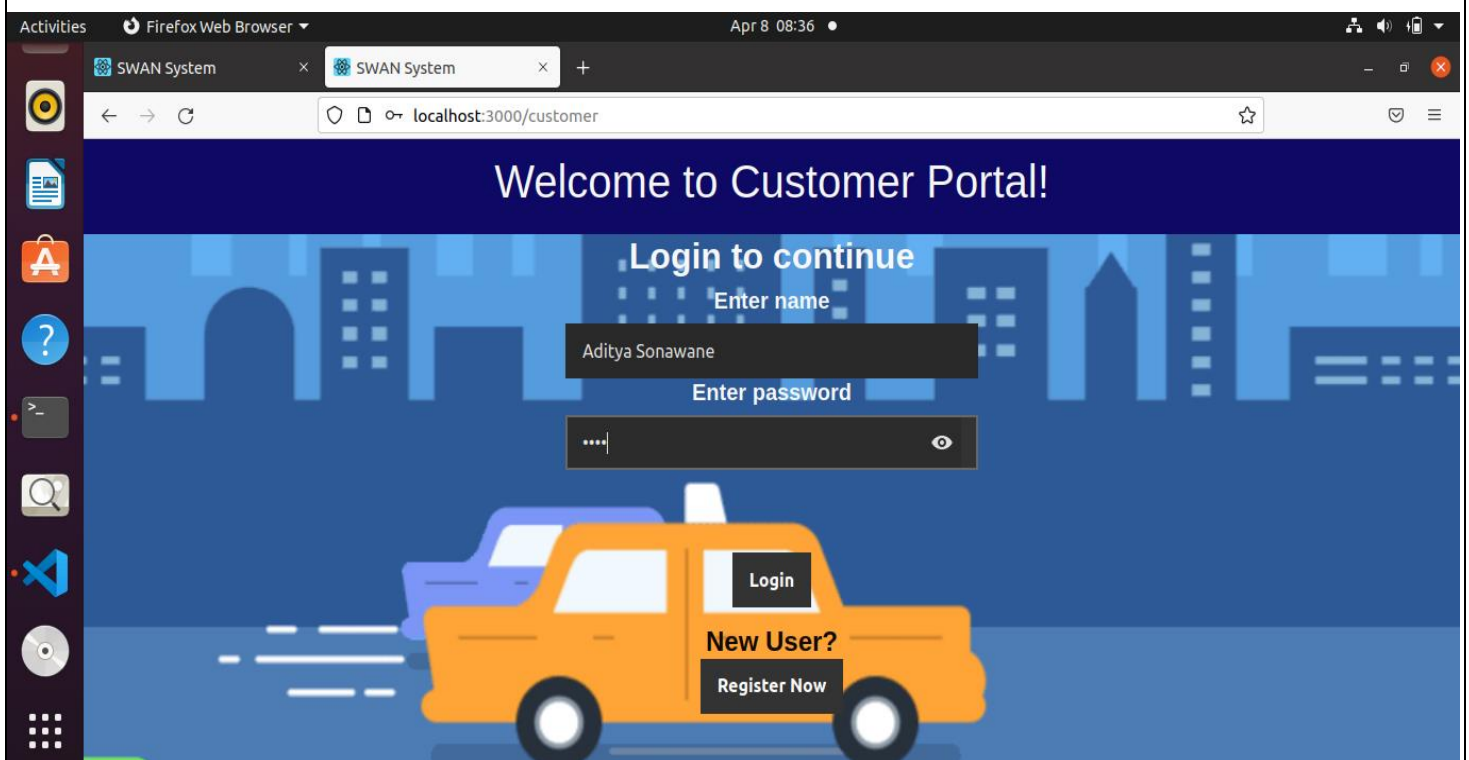
Activities Visual Studio Code Apr 8 02:03

customerLogin.js - swan - Visual Studio Code

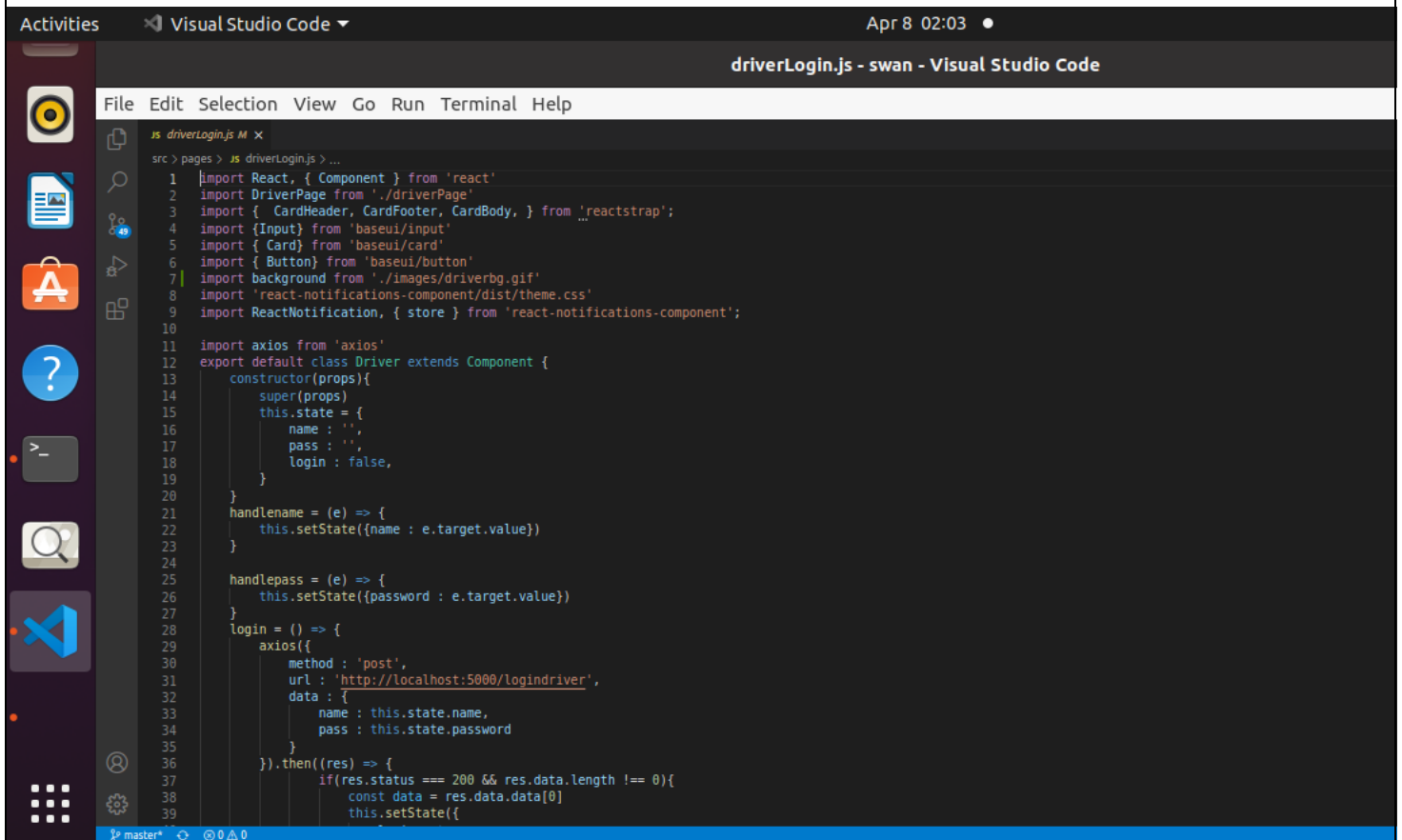
File Edit Selection View Go Run Terminal Help

```
JS customerLogin.js M x
src > pages > JS customerLogin.js > Customer
1 import React, { Component } from 'react'
2 import axios from 'axios'
3 import User from './customerPage'
4 import { Modal, ModalBody, ModalFooter, ModalHeader } from 'reactstrap'
5 import ReactNotification, { store } from 'react-notifications-component';
6 import { Input } from 'baseui/input'
7 import { Button } from 'baseui/button'
8 import background from './images/customerbg.gif';
9 import 'react-notifications-component/dist/theme.css'
10 export default class Customer extends Component {
11   constructor(props){
12     super(props)
13     this.state = {
14       name: '',
15       password: '',
16       phno: '',
17       add: '',
18       login: false,
19       register: false,
20       modal: false
21     }
22   }
23   handleName = (e) => {
24     this.setState({name: e.target.value})
25   }
26   handlePass = (e) => {
27     this.setState({password: e.target.value})
28   }
29   handleAdd = (e) => {
30     this.setState({add: e.target.value})
31   }
32   handlePh = (e) => {
33     this.setState({phno: e.target.value})
34   }
35   toggle = () => {
36     this.setState({
37       login: !this.state.login,
38       register: !this.state.register,
39     })
40   }
41 }
```

master\*

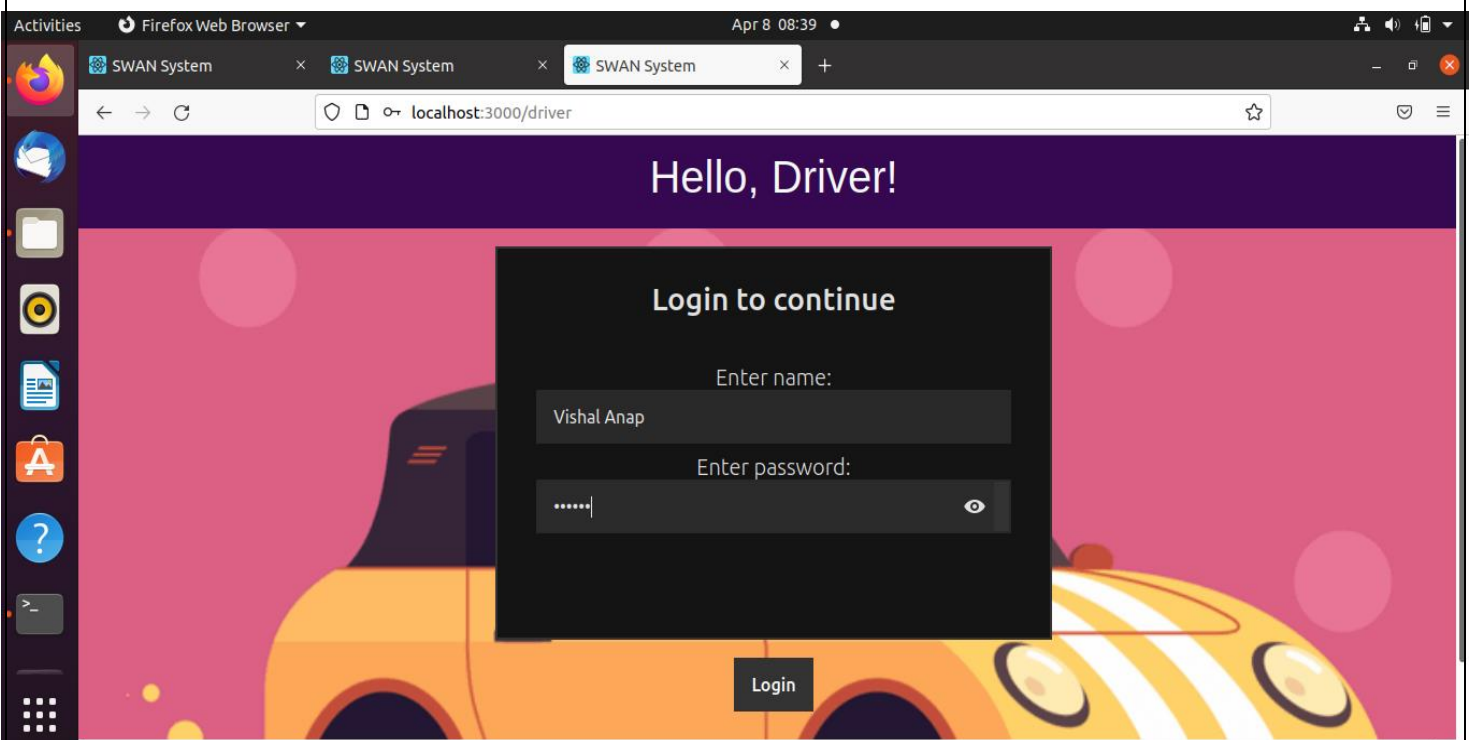


#### 4. Driver Login:



The screenshot shows the Visual Studio Code editor with the file `driverLogin.js` open. The code is a React component for driver login. It imports React, DriverPage, CardHeader, CardFooter, CardBody, Input, Card, Button, background, and ReactNotification. The component has a state with name, pass, and login. It has handlers for name, password, and login. The login handler uses axios to post to `http://localhost:5000/logindriver` with the name and password. It then checks the response status and data.

```
1 import React, { Component } from 'react'
2 import DriverPage from './driverPage'
3 import { CardHeader, CardFooter, CardBody, } from 'reactstrap';
4 import {Input} from 'baseui/input'
5 import { Card } from 'baseui/card'
6 import { Button} from 'baseui/button'
7 import background from './images/driverbg.gif'
8 import 'react-notifications-component/dist/theme.css'
9 import ReactNotification, { store } from 'react-notifications-component';
10
11 import axios from 'axios'
12 export default class Driver extends Component {
13   constructor(props){
14     super(props)
15     this.state = {
16       name: '',
17       pass: '',
18       login: false,
19     }
20   }
21   handlename = (e) => {
22     this.setState({name: e.target.value})
23   }
24
25   handlepass = (e) => {
26     this.setState({password: e.target.value})
27   }
28   login = () => {
29     axios({
30       method: 'post',
31       url: 'http://localhost:5000/logindriver',
32       data: {
33         name: this.state.name,
34         pass: this.state.password
35       }
36     }).then((res) => {
37       if(res.status === 200 && res.data.length !== 0){
38         const data = res.data.data[0]
39         this.setState({
```



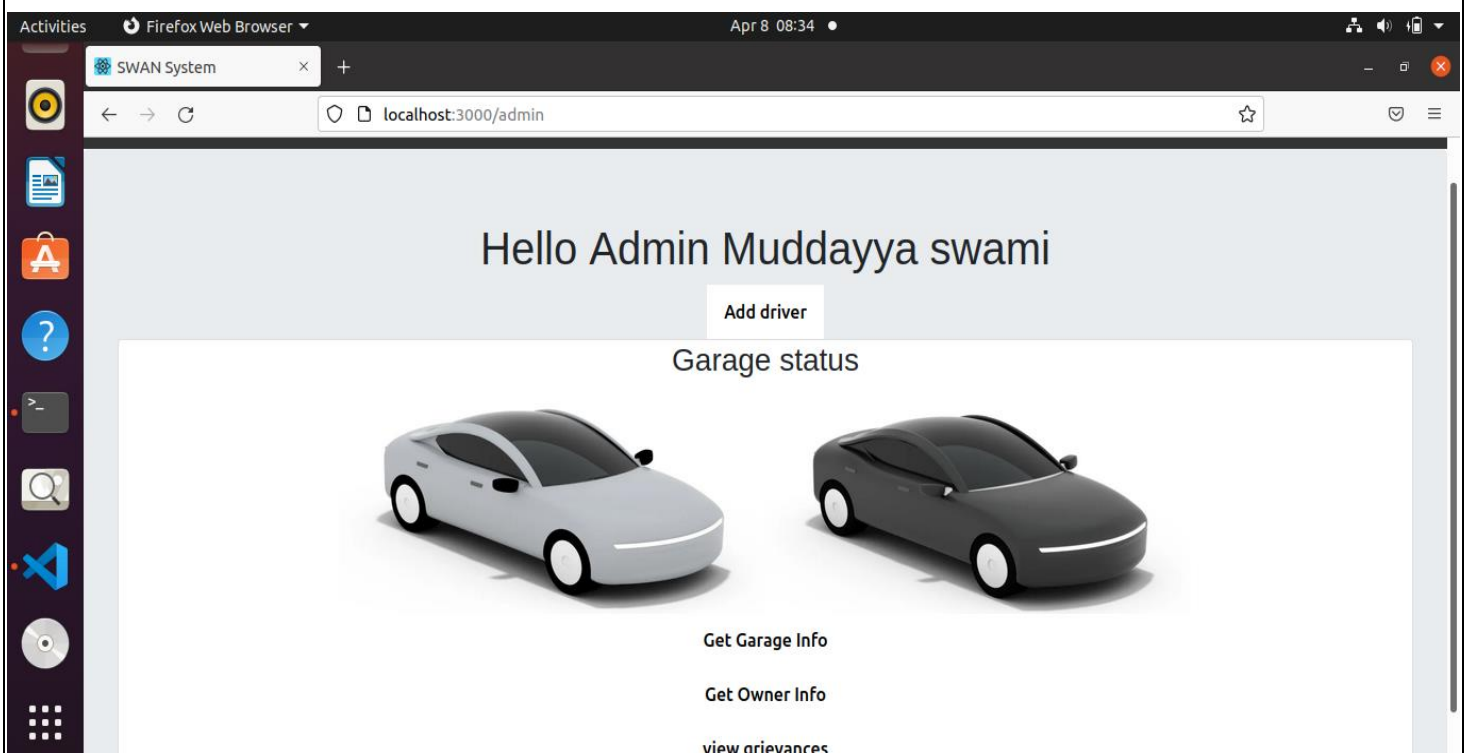
## 5. Admin Dashboard:

Activities Visual Studio Code Apr 8 08:25

adminPage.js - swan - Visual Studio Code

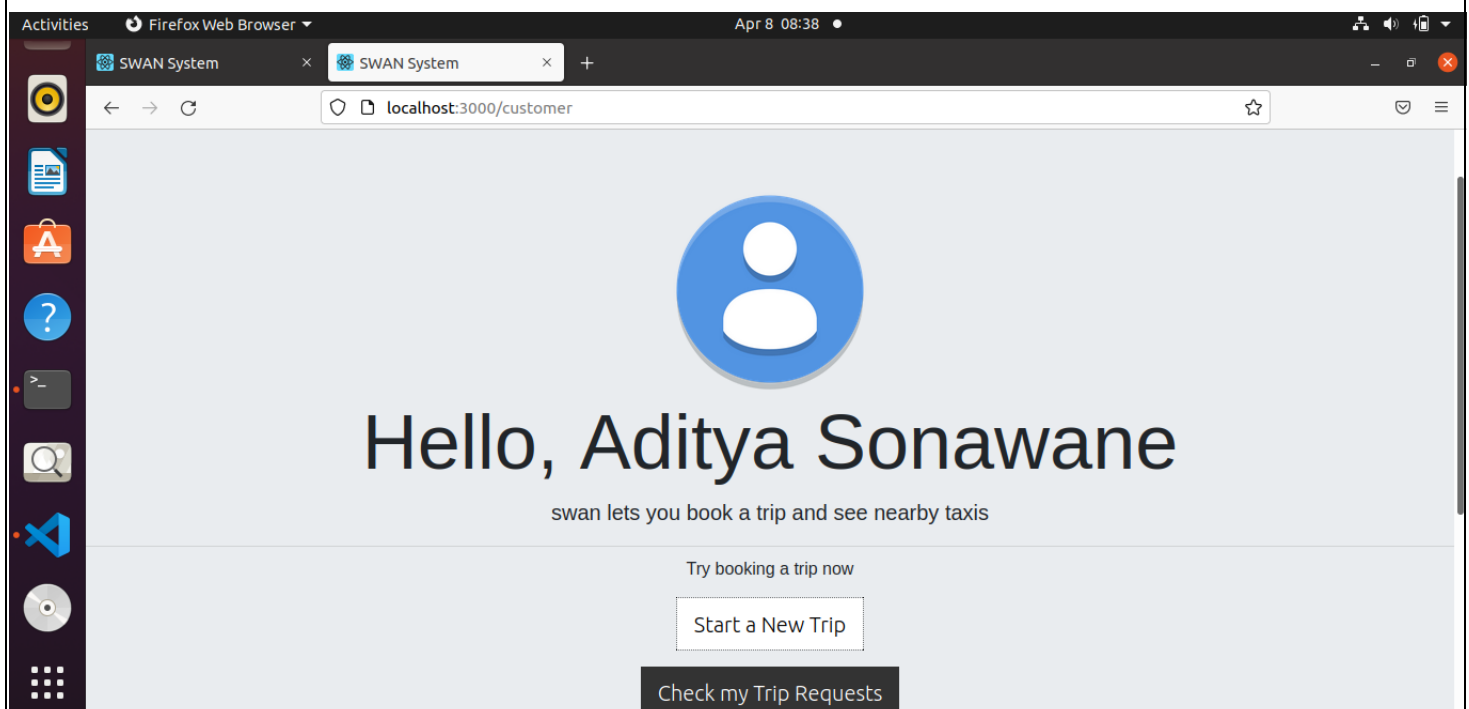
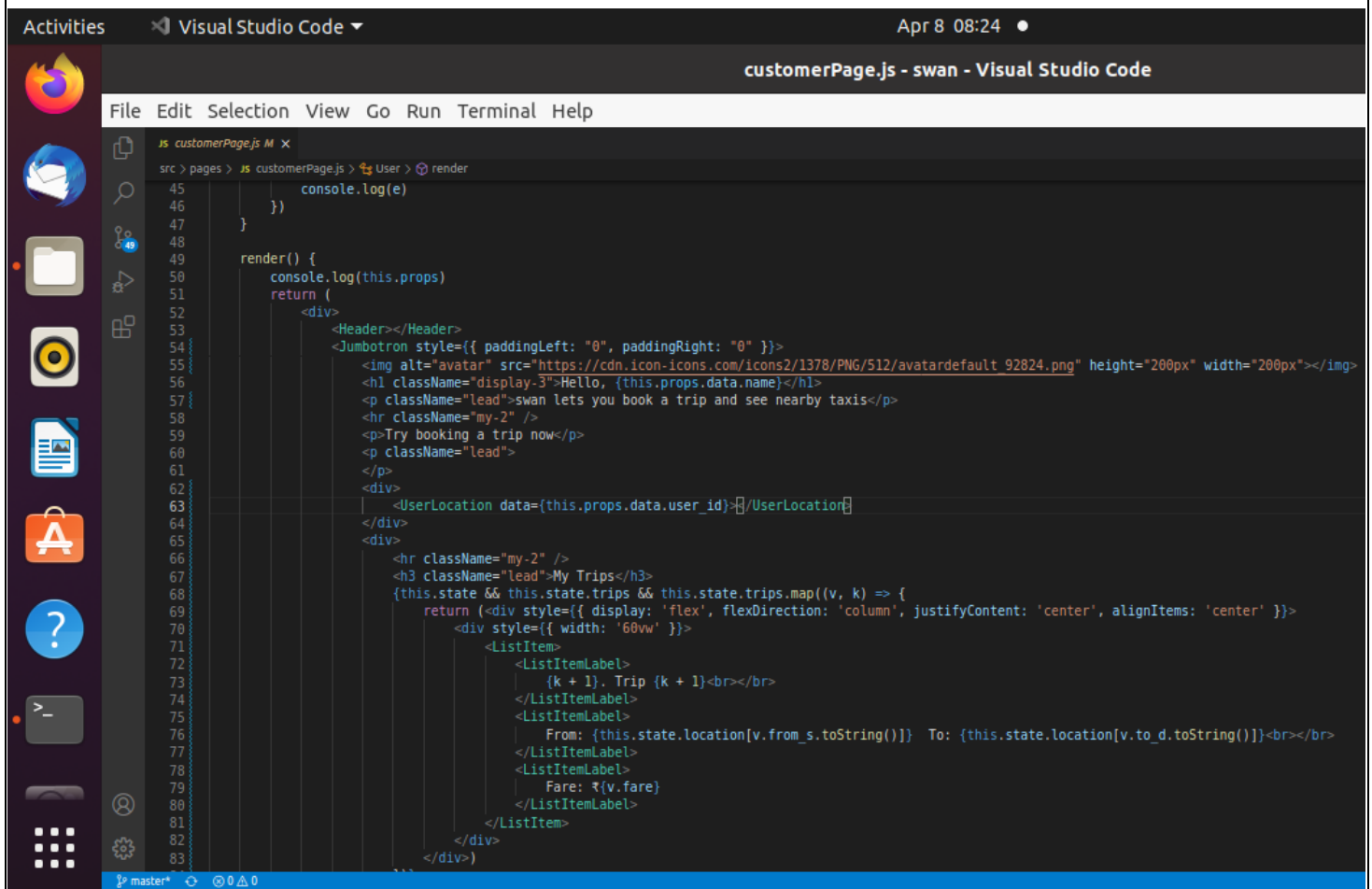
File Edit Selection View Go Run Terminal Help

```
JS adminPage.js M X
src > pages > JS adminPage.js > Admin > getgrievance > then() callback
1 import React, { Component } from 'react'
2 import { Card, Jumbotron, Spinner } from 'reactstrap'
3 import { Button } from 'baseui/button'
4 import { Input } from 'baseui/input'
5 import { Modal, ModalBody, ModalFooter } from 'baseui/modal'
6 import Header from './headerFile'
7 import axios from 'axios'
8 import ReactNotification, { store } from 'react-notifications-component';
9 import 'react-notifications-component/dist/theme.css'
10 import { ListItem, ListItemLabel } from 'baseui/list'
11
12 export default class Admin extends Component {
13   constructor(props) {
14     super(props)
15     this.state = {
16       open : false,
17       loading : false
18     }
19   }
20   getgarage = () => {
21     axios({
22       method : 'get',
23       url : 'http://localhost:5000/api/getgarage'
24     }).then(res => {
25       console.log(res.data)
26       this.setState({
27         garage : res.data.garage
28       })
29     }).catch(e => {
30       console.log(e)
31     })
32   }
33
34   getgrievance = () => {
35     axios({
36       method : 'get',
37       url : 'http://localhost:5000/api/getgrievance'
38     }).then(res => {
39       console.log(res.data)
```

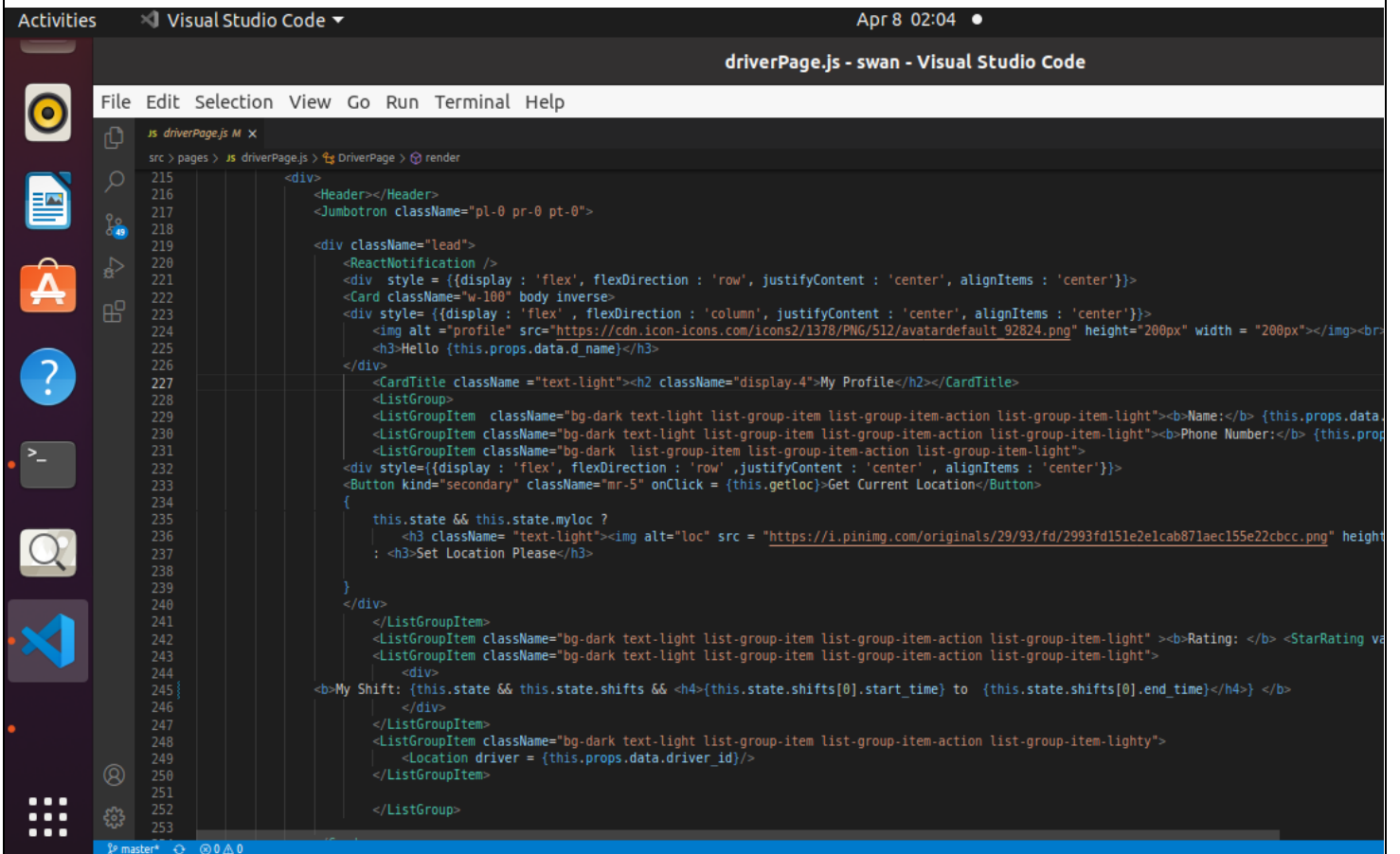




## 6. Customer Dashboard:

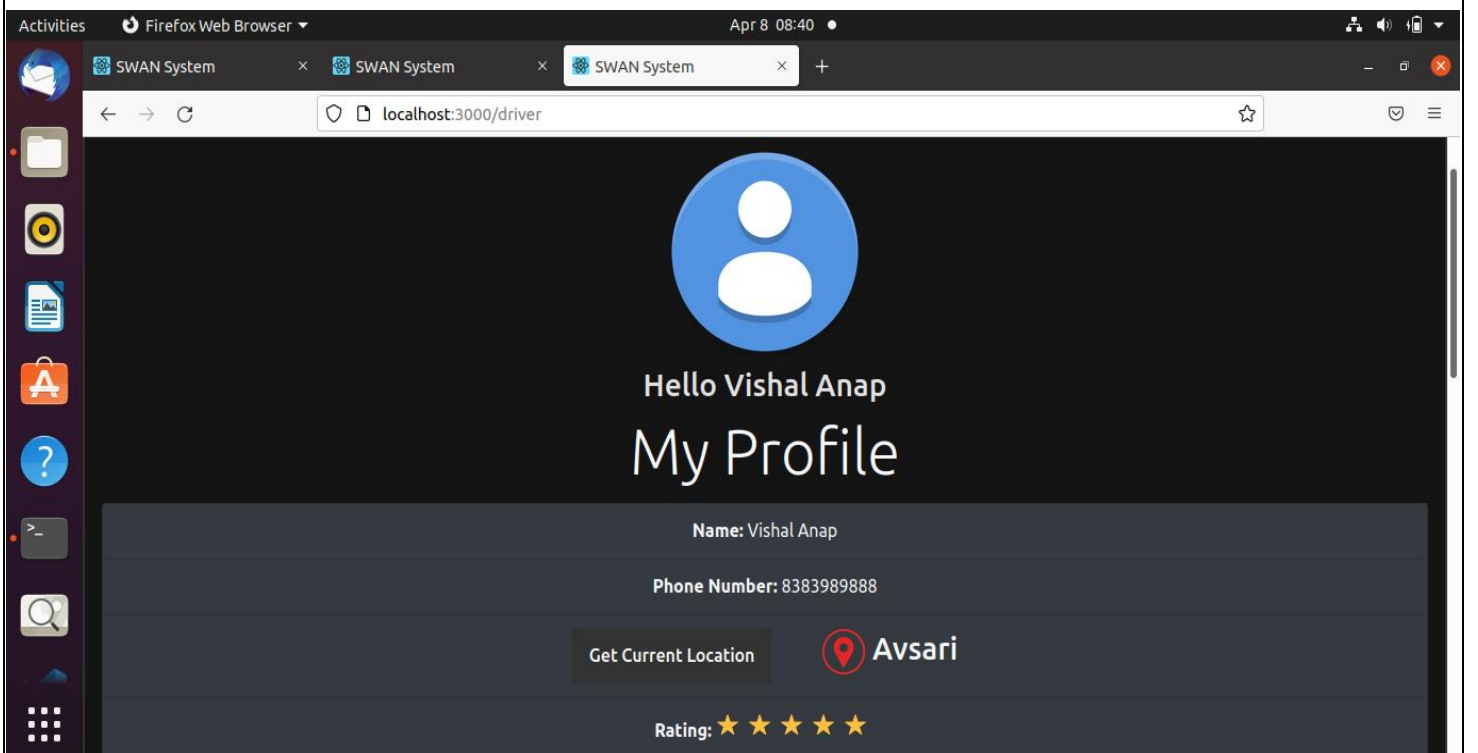


## 7. Driver Dashboard:

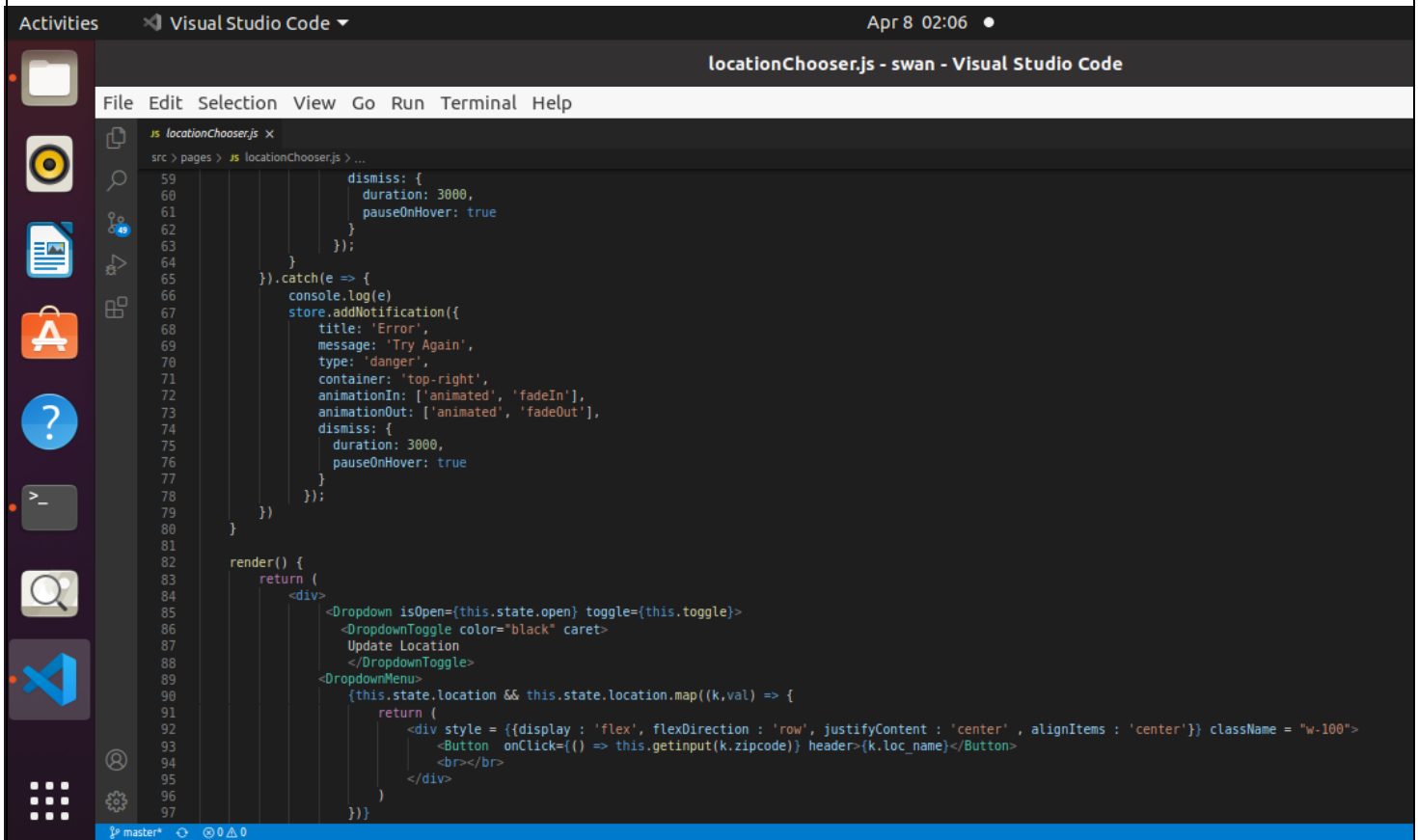


The screenshot shows the Visual Studio Code editor with the file `driverPage.js` open. The code is a React component for a driver's profile page. It includes a header, a jumbotron, a profile card with a profile picture, a card title, and a list of items including a location, a rating, and a shift. The code uses JSX and includes comments for styling and functionality.

```
215 <div>
216   <Header></Header>
217   <Jumbotron className="pl-0 pr-0 pt-0">
218
219     <div className="lead">
220       <ReactNotification />
221       <div style = {{display : 'flex', flexDirection : 'row', justifyContent : 'center', alignItems : 'center'}}>
222       <Card className="w-100" body inverse>
223       <div style= {{display : 'flex', flexDirection : 'column', justifyContent : 'center', alignItems : 'center'}}>
224         </img><br>
225         <h3>Hello {this.props.data.d_name}</h3>
226       </div>
227       <CardTitle className="text-light"><h2 className="display-4">My Profile</h2></CardTitle>
228       <ListGroup>
229         <ListGroupItem className="bg-dark text-light list-group-item list-group-item-action list-group-item-light"><b>Name:</b> {this.props.data.d_name}</div>
230         <ListGroupItem className="bg-dark text-light list-group-item list-group-item-action list-group-item-light"><b>Phone Number:</b> {this.props.data.d_phone}</div>
231         <ListGroupItem className="bg-dark text-light list-group-item list-group-item-action list-group-item-light">
232         <div style= {{display : 'flex', flexDirection : 'row', justifyContent : 'center', alignItems : 'center'}}>
233         <Button kind="secondary" className="mr-5" onClick = {this.getloc}>Get Current Location</Button>
234         {
235           this.state && this.state.myloc ?
236           <h3 className="text-light"><img alt="loc" src = "https://i.pinimg.com/originals/29/93/fd/2993fd151e2e1cab871a6c155e22cbcc.png" height="200px" width = "200px"></img>
237           : <h3>Set Location Please</h3>
238         }
239       </div>
240       </ListGroup>
241       <ListGroupItem>
242       <ListGroupItem className="bg-dark text-light list-group-item list-group-item-action list-group-item-light"><b>Rating:</b> <StarRating value={this.props.data.d_rating}></div>
243       <ListGroupItem className="bg-dark text-light list-group-item list-group-item-action list-group-item-light">
244       <div>
245       <b>My Shift: {this.state && this.state.shifts && <h4>{this.state.shifts[0].start_time} to {this.state.shifts[0].end_time}</h4> </b>
246       </div>
247       </ListGroupItem>
248       <ListGroupItem className="bg-dark text-light list-group-item list-group-item-action list-group-item-light">
249       <Location driver = {this.props.data.driver_id}/>
250       </ListGroupItem>
251     </div>
252   </div>
253 </div>
```

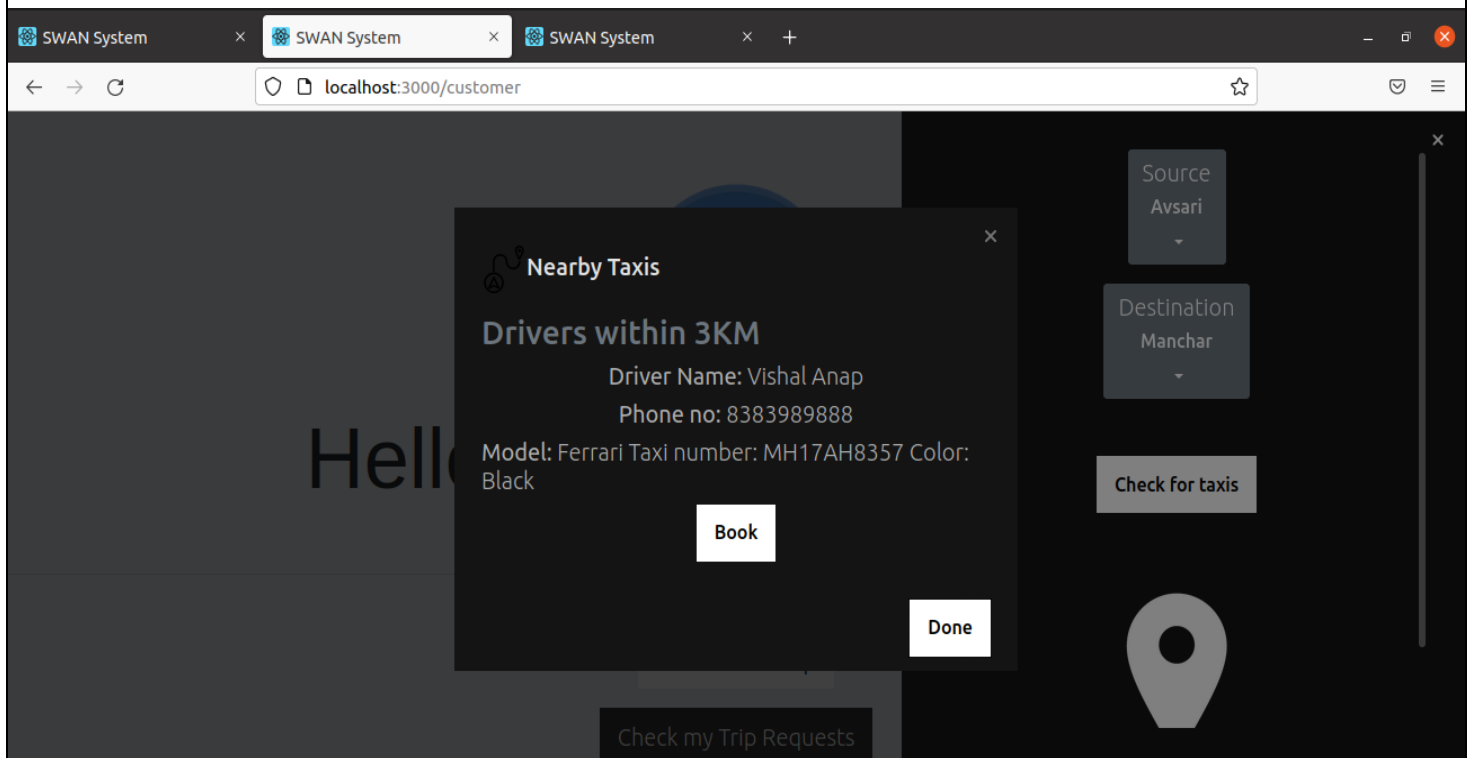


## 8. Choose Locations and Book Taxi:

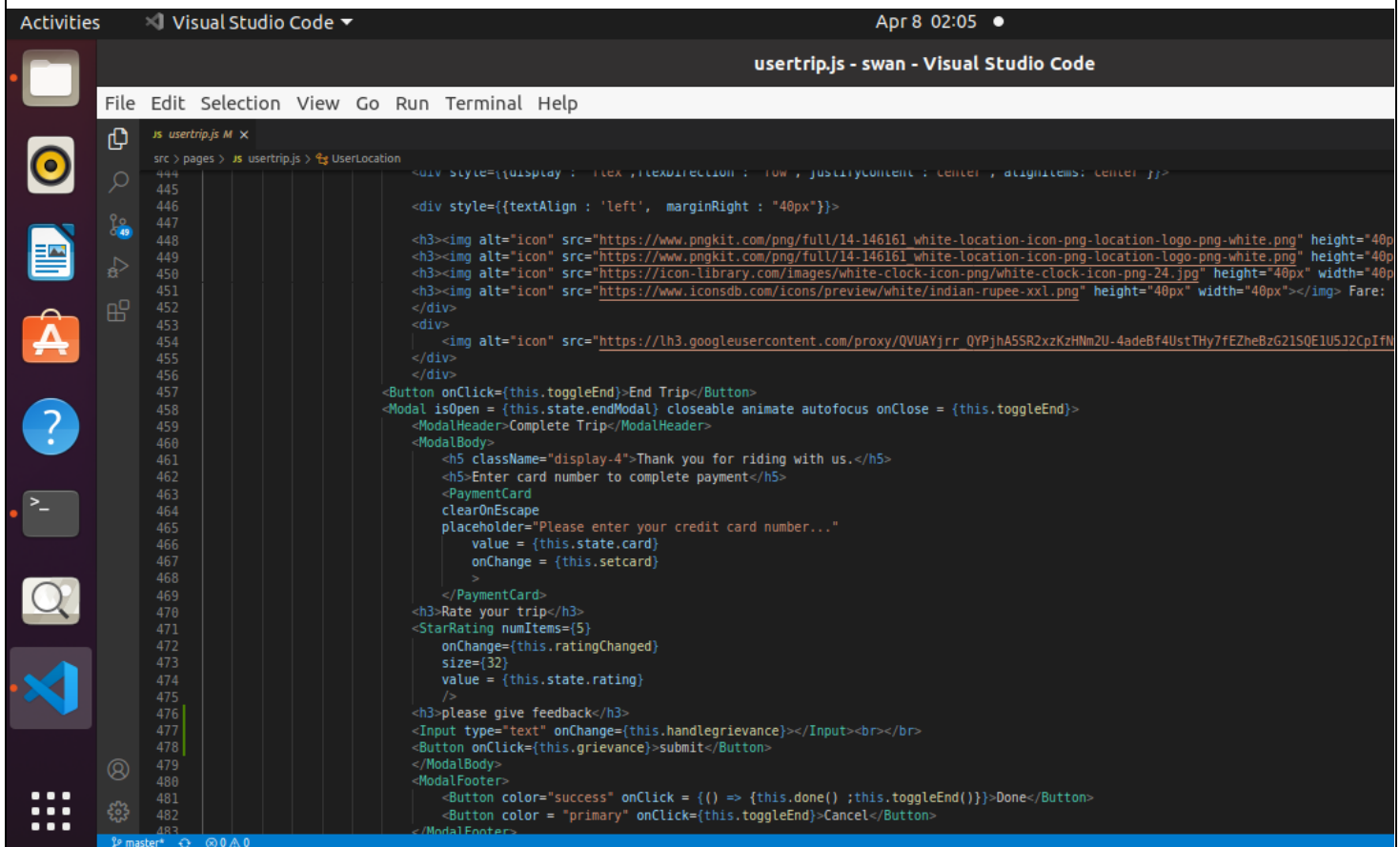


The screenshot shows the Visual Studio Code editor with the file `locationChooser.js` open. The code is written in JavaScript and includes a `dismiss` function, a `render` function, and a `render` function. The `render` function uses React to render a form with a dropdown menu for location selection and a button to book a taxi. The code is as follows:

```
59     dismiss: {
60       duration: 3000,
61       pauseOnHover: true
62     }
63   });
64 }
65 }).catch(e => {
66   console.log(e)
67   store.addNotification({
68     title: 'Error',
69     message: 'Try Again',
70     type: 'danger',
71     container: 'top-right',
72     animationIn: ['animated', 'fadeIn'],
73     animationOut: ['animated', 'fadeOut'],
74     dismiss: {
75       duration: 3000,
76       pauseOnHover: true
77     }
78   });
79 })
80 }
81
82 render() {
83   return (
84     <div>
85       <Dropdown isOpen={this.state.open} toggle={this.toggle}>
86         <DropdownToggle color="black" caret>
87           Update Location
88         </DropdownToggle>
89         <DropdownMenu>
90           {this.state.location && this.state.location.map((k,val) => {
91             return (
92               <div style = {{display : 'flex', flexDirection : 'row', justifyContent : 'center', alignItems : 'center'}} className = "w-100">
93                 <Button onClick={() => this.getInput(k.zipcode)} header={k.loc_name}</Button>
94                 <br></div>
95             )
96           })
97         </DropdownMenu>
98       </div>
99     )
100   )
101 }
```



## 9. Payment, Ratings and Feedback:

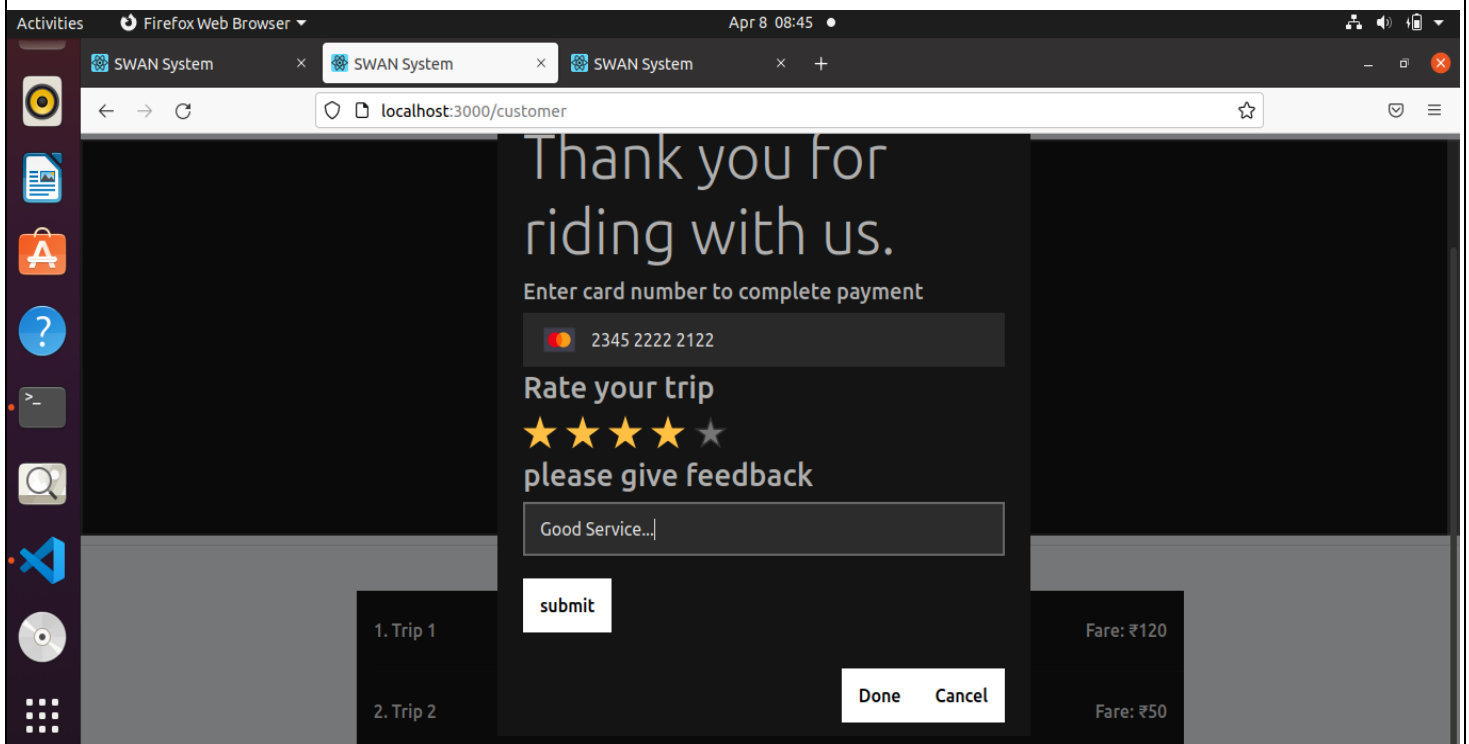


```
usertrip.js - swan - Visual Studio Code

File Edit Selection View Go Run Terminal Help

JS usertrip.js M X
src > pages > JS usertrip.js > UserLocation
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483

<div style={{display : 'flex', flex-direction : 'row', justify-content : center , alignItems: center ;}}>
  <div style={{textAlign : 'left', marginRight : '40px'}}>
    <h3></h3>
    <h3></h3>
    <h3></h3>
    <h3></h3> Fare:
  </div>
  <div>
    
  </div>
</div>
<Button onClick={this.toggleEnd}>End Trip</Button>
<Modal isOpen = {this.state.endModal} closeable animate autofocus onClose = {this.toggleEnd}>
  <ModalHeader>Complete Trip</ModalHeader>
  <ModalBody>
    <h5 className="display-4">Thank you for riding with us.</h5>
    <h5>Enter card number to complete payment</h5>
    <PaymentCard
      clearOnEscape
      placeholder="Please enter your credit card number..."
      value = {this.state.card}
      onChange = {this.setcard}
    >
  </PaymentCard>
    <h3>Rate your trip</h3>
    <StarRating numItems={5}
      onChange={this.ratingChanged}
      size={32}
      value = {this.state.rating}
    />
    <h3>please give feedback</h3>
    <Input type="text" onChange={this.handlegrievance}></Input><br></br>
    <Button onClick={this.grievance}>submit</Button>
  </ModalBody>
  <ModalFooter>
    <Button color="success" onClick = {() => {this.done(); this.toggleEnd()}}>Done</Button>
    <Button color = "primary" onClick={this.toggleEnd}>Cancel</Button>
  </ModalFooter>
</Modal>
```



## **TESTING:**

Security testing is a non-functional software testing technique used to determine if the information and data in a system is protected. The goal is to purposefully find loopholes and security risks in the system that could result in unauthorized access to or the loss of information by probing the application for weaknesses.

In our project we used automated security testing plugin “HCL AppScan CodeSweep” for Static Application Security Testing (SAST) functionality for detecting vulnerabilities early in the development life cycle.

## **CONCLUSION:**

To conclude, this is an efficient Taxi Management System application which enables its users to book the taxis which are present in the proximity to their location. Along with that, it provides its users with the complete details of driver and taxi. User’s views about the system are also taken into consideration to improve the system in future. It also facilitates the drivers working with the system by providing easier ride requests and giving them freedom to choose their passenger. The admin can maintain the whole system, by recording the garage status, owner’s information. Admin can add new driver to the system and can expand the network of whole system. So, this application can be used as location based reactive system for taxi booking.

## **FUTURE SCOPE:**

This project can be extended further by adding several functionalities to it. Currently, this application provides the list of places to book taxi, instead of that Google maps functionality can be added to it in future. The feedback part of the application will be made optional for users. Further, we will add rent functionality which will help admin to pay the rent to respective taxi owner. Also, we will allow users and drivers to upload their profile photo so that they can easily recognize each other. This application will be improved further, to secure the user and driver information.

**GITHUB LINK:**

[https://github.com/vishalanap/Taxi\\_Management\\_System](https://github.com/vishalanap/Taxi_Management_System)

**REFERENCES:**

1. Taxi management system approach- Taxi Management Software Solutions – Taxi Management System - Taxi Fleet Management (aistechnolabs.com)
2. Connecting mysql server using nodejs -  
<https://www.tutorialkart.com/nodejs/nodejs-mysqlconnect-to-database>
3. Research paper -  
[https://www.researchgate.net/publication/348304336\\_Design\\_and\\_Implementation\\_of\\_Taxi\\_Management\\_System](https://www.researchgate.net/publication/348304336_Design_and_Implementation_of_Taxi_Management_System)