**FINAL REPORT**

**github repo link:** **https://github.com/SmartPracticeschool/SBSPS-Challenge-4410-Integrated-Transport-System.git**

**demo video**

**https://drive.google.com/file/d/13y895h2Wm1Z3f55b6kdpRQtDA6\_\_3z-6/view?usp=sharing**

**INFORMATION ABOUT VIDEO**

**FRONTEND**

**RideASAP:-(Post-Lockdown Ticket booking Web-App)**

**RideASAP is a web-app which ensures every person should reach on time even in this current lockdown scenerio.**

**Motto was to organize things in such a way that ensures No-one should suffer in this scenerio for eg," when one has to reach his/her office on tym but buses and metro are not available"**

**\*Front-End\* :-**

**The front-end of this Web-App is made by using AngularJS framework . The motto of using angular rather than using the basics rituals( HTML,CSS and JS) :-**

**-> Our Web-app is highly dynamic in nature as it includes many server side communications therfore it is beneficial to make it using Angular as it provides many In-built Modules for server side communication.**

**-> When this Web-App will be used by the users there will be loads of data which will make any web-app slow in nature and may take more loading time due to continous server communication whereas Angular make one page websites which decrerases loading time and make it faster.**

**\*Pages\* :-**

**-> Signup/Signin (For Users)**

**-> User should signup by providing us the information like Name,Age,Email,Phone number and password**

**-> There are some validations in Signup inputs so that user cannot enter any invalid data inputs.**

**-> These inputs are then sent to Express server using HTTP post request and error response is displayed on the the page itself.**

**-> For Signin user should have signed up otherwise there will be error from server side.**

**-> WE have taken 2 inputs for Sign in i.e. EmailID and password . this data is then again sent to server if successfull he/she will be routed to our Home page**

**and will assigned by a unique token as response from server side or if not error response message will be flash on top of page.**

**->Home page**

**-> Here, User will give inputs like reaching time , source and destination and sent to server side after as HTTp post request after clicking on best route button Best path will be generated as Pop-up window which includes**

**Best path, Total distance, Total time, Time between two stops and Time at which will tell at what time one should get to the initial stop to reach on time.**

**-> This pop-up has a button , when user will click on that button he/she will be routed to ticket page in which unique ticket is generated by using hash which we get from server side , by that hash we generate QR code which wi;; act as an integrated ticket for bus , metro and trains.**

**\*For Authority\***

**-> Sign up and sign in page which also conatains user id which will act as a validation.**

**then dashboard is opened if sign in is successfull.**

**->dashboard contains few features which contains rescheduling , checking information etc.**

**BACKEND**

**Backend of users:**

**In User Sign-up page first we used npm module "joi" for validation of user inputs like No user should have same email or phone number and User will be stored in mongoDB udatabase using mongoose in node,**

**after successfull sign-up, In sign-in page when user will add correct pair of user-id, email and password a JWT( json web token) will be send to front end in json format and it'll be stored as header in users browers  and will be send back for the authorization of user for every private route.**

**In home page node will accept "JWT", "source", "destination", and approximate time to react his desired destination, First JWT will be verified if user is signed-in best path will be generated using Dijsktra algoritm in node.js and path will be generated by integrated all means of transport required to reach his destination, price total distance and intermediate time will be send to angular in json format followed by path.**

**once book now is clicked backend will generate ticket which will include user name, reaching time, total distance, source , destination and qr-code containing hashed value of all the data this qr will be readed in order to authenticate user's valid ticket.**

**In Authority Sign-up page first we used npm module "joi" for validation of user inputs like No user should have same email or phone number, also there should be unique id used provided by govt authorities to their employes and User will be stored in mongoDB udatabase using mongoose in node,**

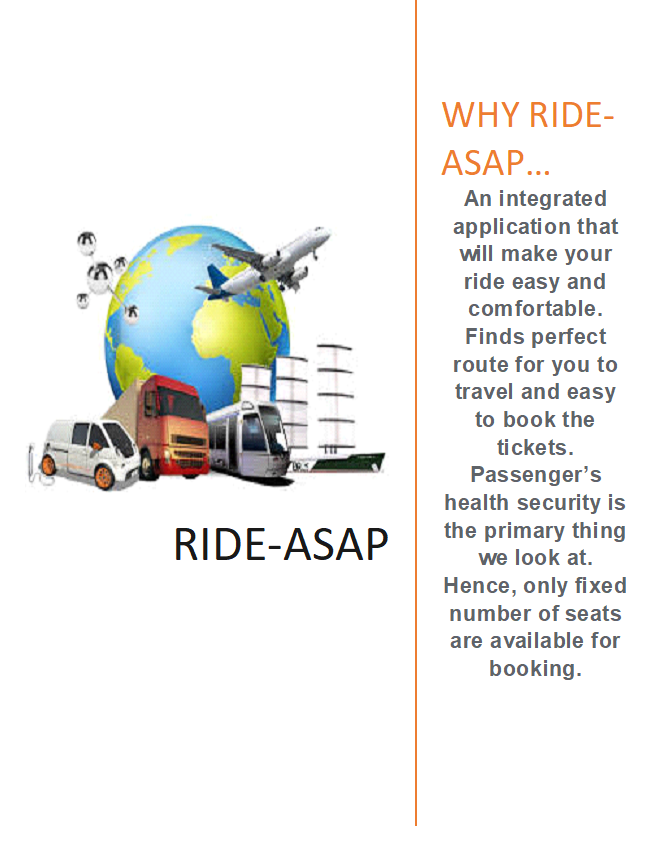
**authority will also be asked about there agency as bus or train and this information will be used to redirect authority to there dashboard as per there designations,**

**after successfull sign-up, In sign-in page when user will add correct pair of user-id, email and password a JWT( json web token) will be send to front end in json format and it'll be stored as header in users browers  and will be send back for the authorization of user for every private route.**

**In dashboard information about total users, total booking, last month revenue and other  things will be generated using information stored in database**

**In reschedule authority is allowed to change any bus route with other buses as more buses may required for some certain path as per requirements due to post lockdown conditions**

**In checking information authority will provide bus number or train number as per there controls and will be able to fetch all the information that particular bus/train**



**INTRODUCTION**

OVERVIEW

 Provides **end to end route** from given source to destination by integrating different modes of transport as required.

 Suggests **best real time route** for the user.

**Single ticket** generation for the entire journey.

 Maintenance of **social distancing** and reducing the contact between persons.

It is SIMPLIFIED:

For users:

 The person just has to enter the source and destination and expected time to reach his/her destination.

 He'll be provided with the **convenient route**.

For authority:

 Authority can check all the transport related information by logging on their respective accounts.

 Flexible to edit the transport related information.

 Update transport

 Cancel transport

PURPOSE

 An intelligent application where all bus/train/metro services can integrate.

 Designs the best possible **end to end route** in real time as per user's requirement.

 Help to **reduce** the **OVER OCCUPANCY.**

 Maintain **SOCIAL-DISTANCING** by integrating different modes of transport.

**USER FRIENDLY:** Schedules the whole route with proper timing.

 Allows the **valid authorities only** to schedule the timing if there are any changes in regular routes or if required by any other reason,

**Minimize manual work:** Make easier and comfortable for authority to manage the services.

 Authorities can insert new transport by entering the required information.

**EASY BOOKING:**  

☆ Single ticket for whole journey.

☆ User can book their ticket easily only if seats are available.

**LITERATURE SURVEY**

EXISTING PROBLEM

 Post-Lockdown, it will be risky to allow the public transportation without proper mechanism to maintain the social distancing and over-occupancy.

 Especially the frequency of buses, trains and metros shall be managed properly to utilize the capacity with social distancing criteria.

 The transport authorities must integrate together to maintain the system properly.

PROPOSED SOLUTION

 Our app will assign an integrated ticket for every public transport which means one can travel to his/her destination by using only one ticket for all means of public transport.

 User will enter his or her source, destination and reaching time as input and our app will calculate the best route.

 The ticket is generated online with unique QR Code which will reduce the contact between people.

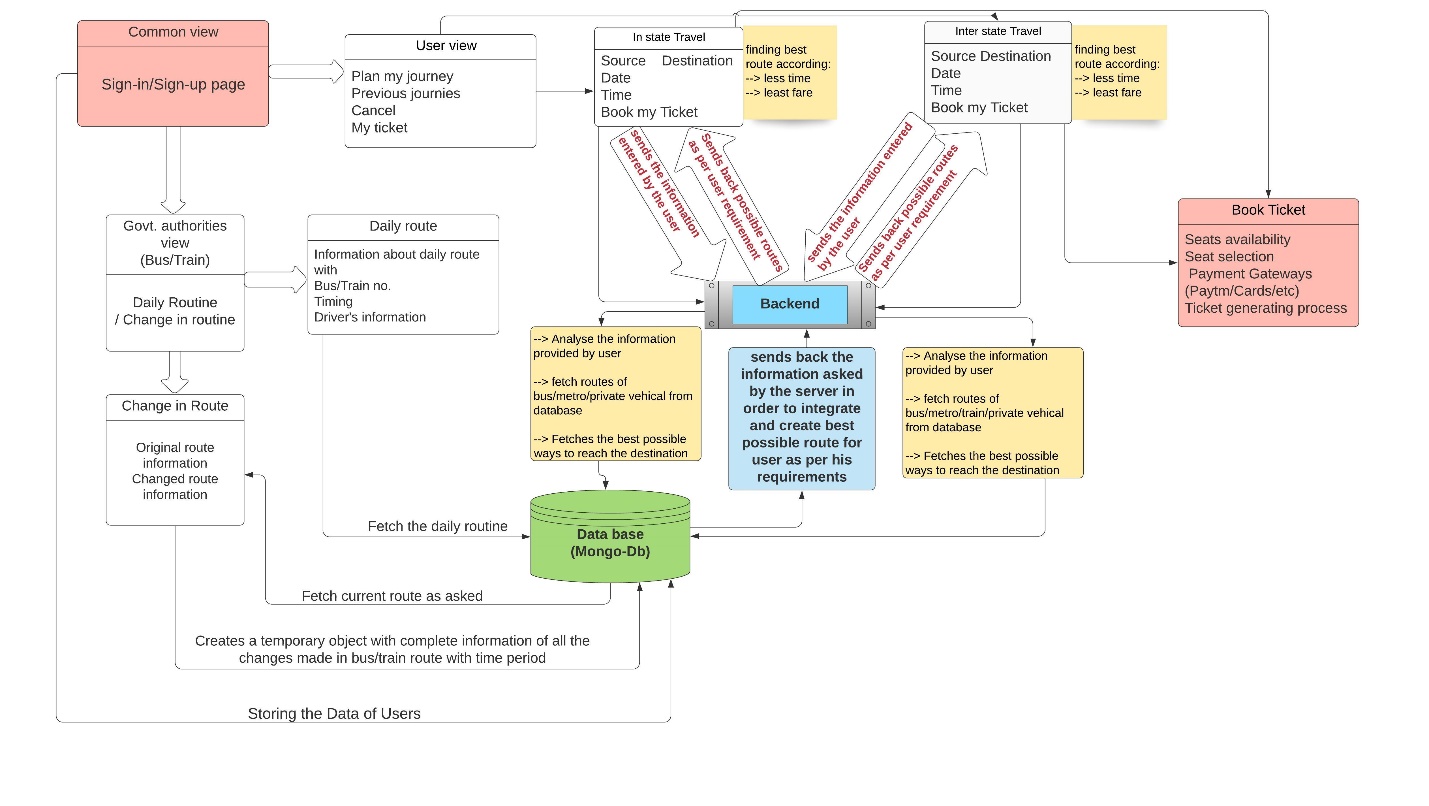
 The problem of frequency of transport will be covered under integration of transport systems.

 Capacity of buses and trains will be choose by authorities and only that much seats will be available for user to book which will help to overcome problem of over-occupancy.

 Our app will run in real-time location so that app will be aware of available seats in real-time and that seats can be allotted to other user after using proper sanitization techniques.

**THEORETICAL ANALYSIS**

**BLOCK DIAGRAM:**

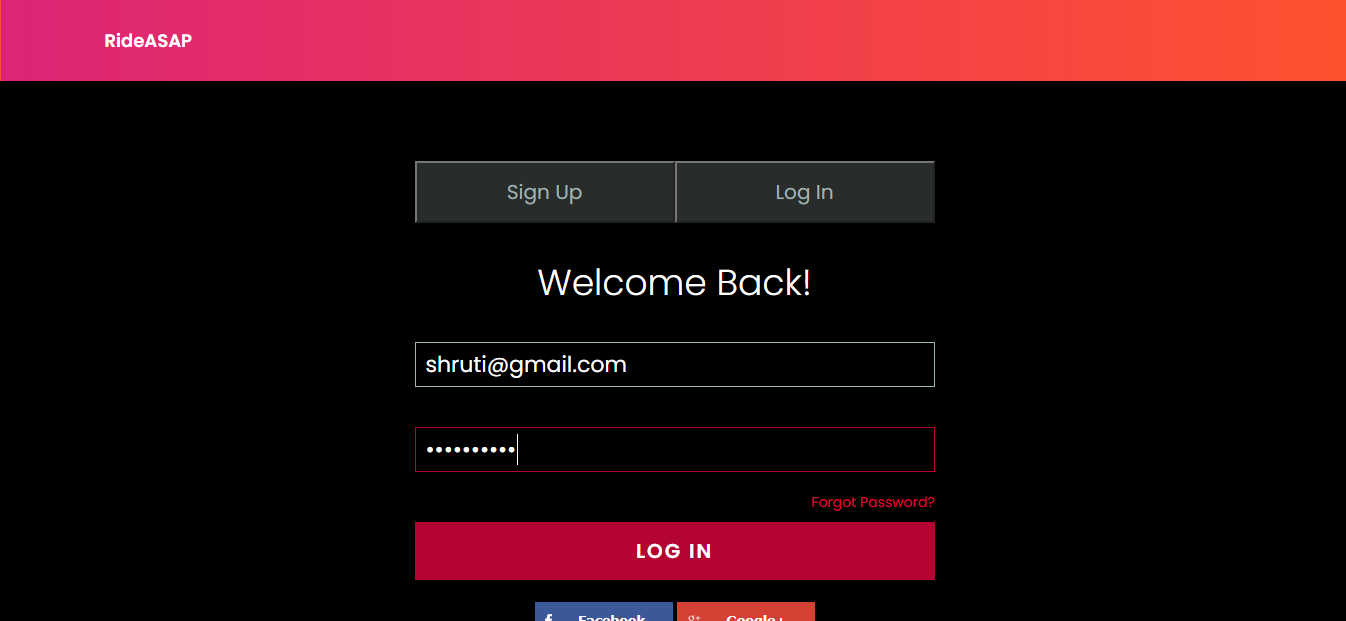


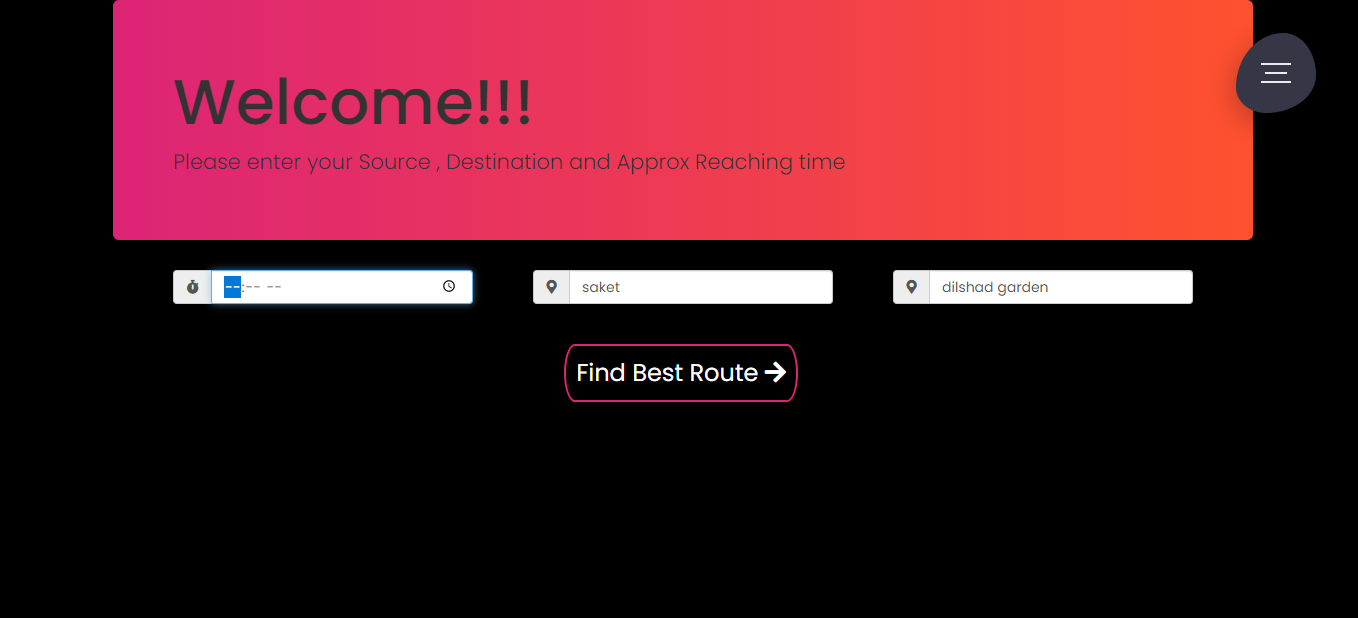
**HARDWARE/SOFTWARE DESIGNING:**

 VS CODE

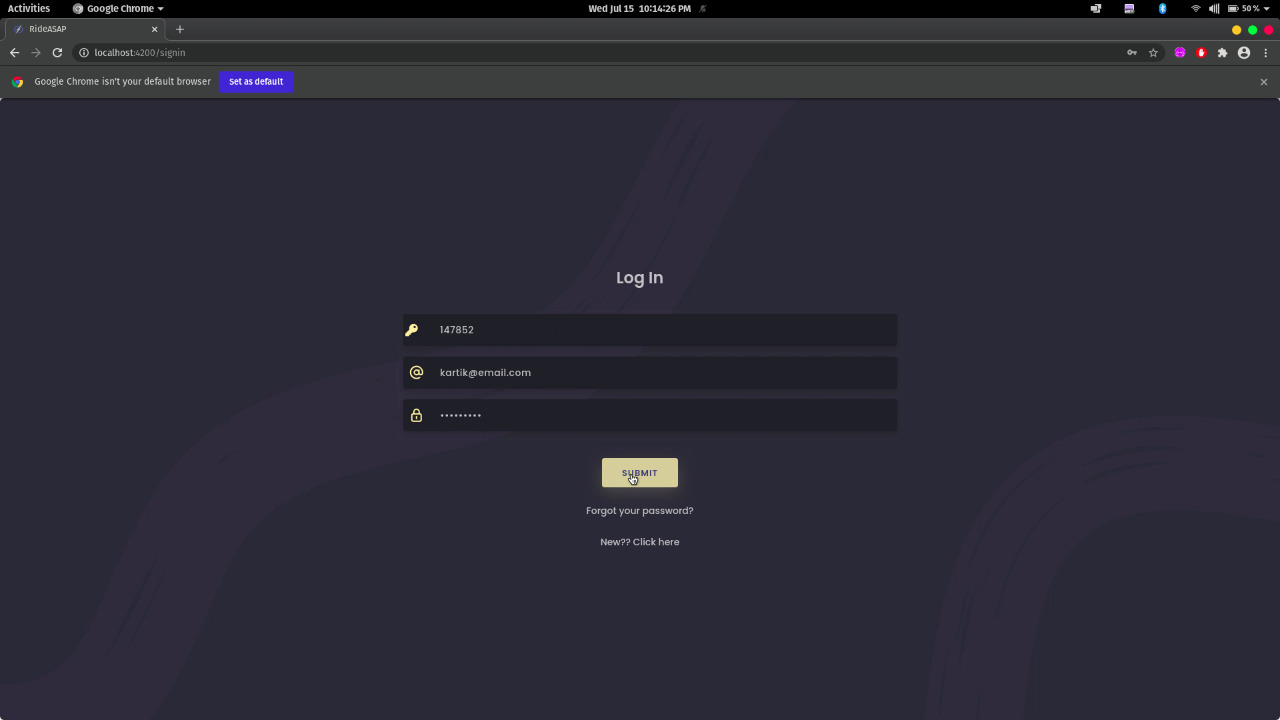
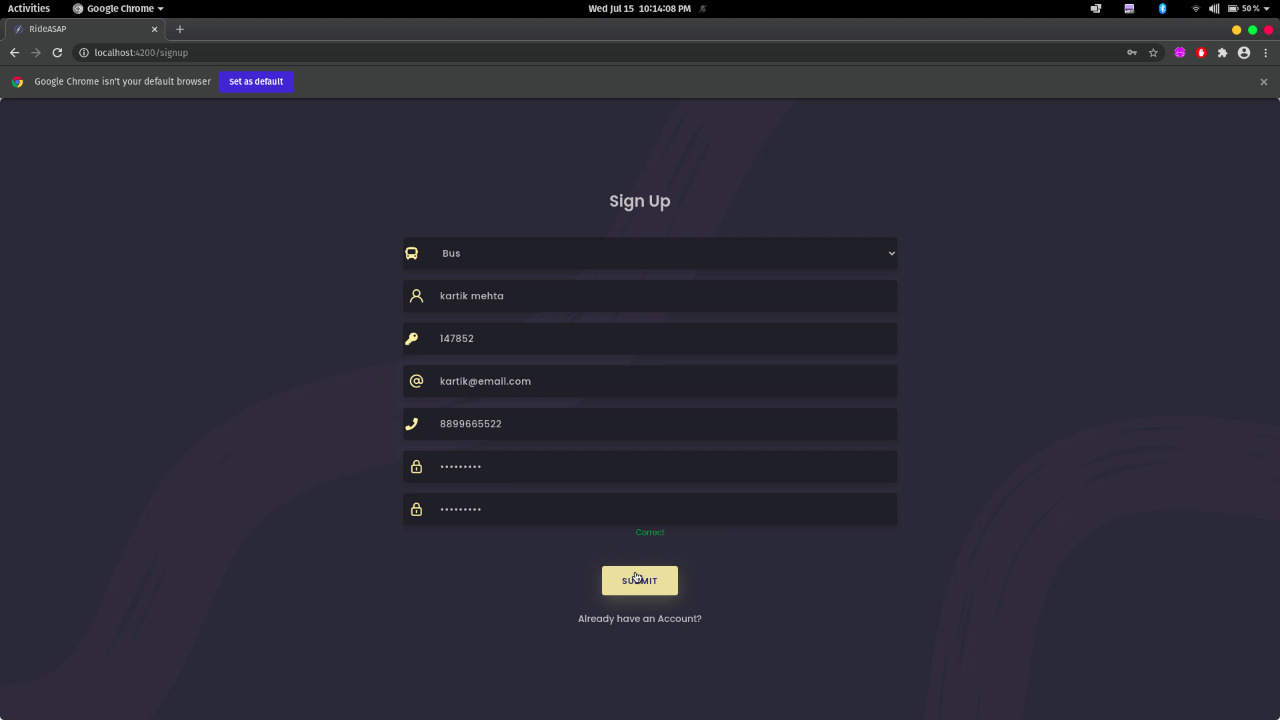
**EXPERIMENTAL INVESTIGATIONS**

  USER SIDE:

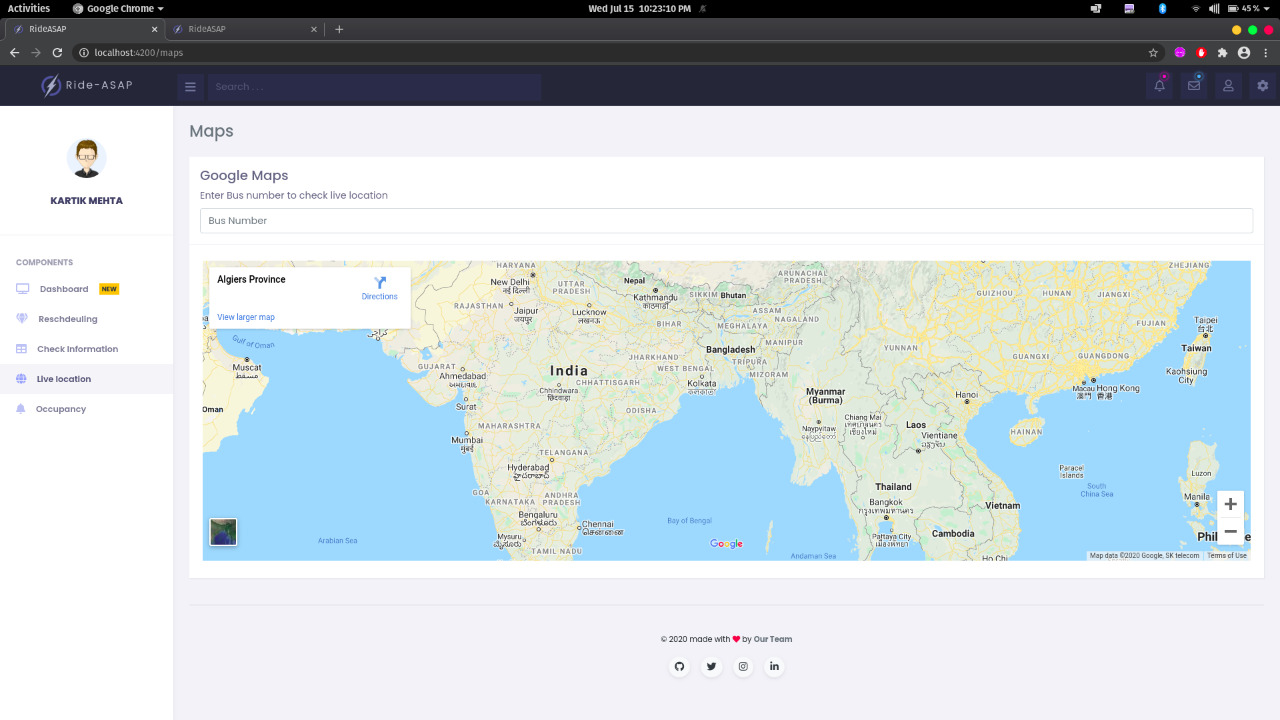
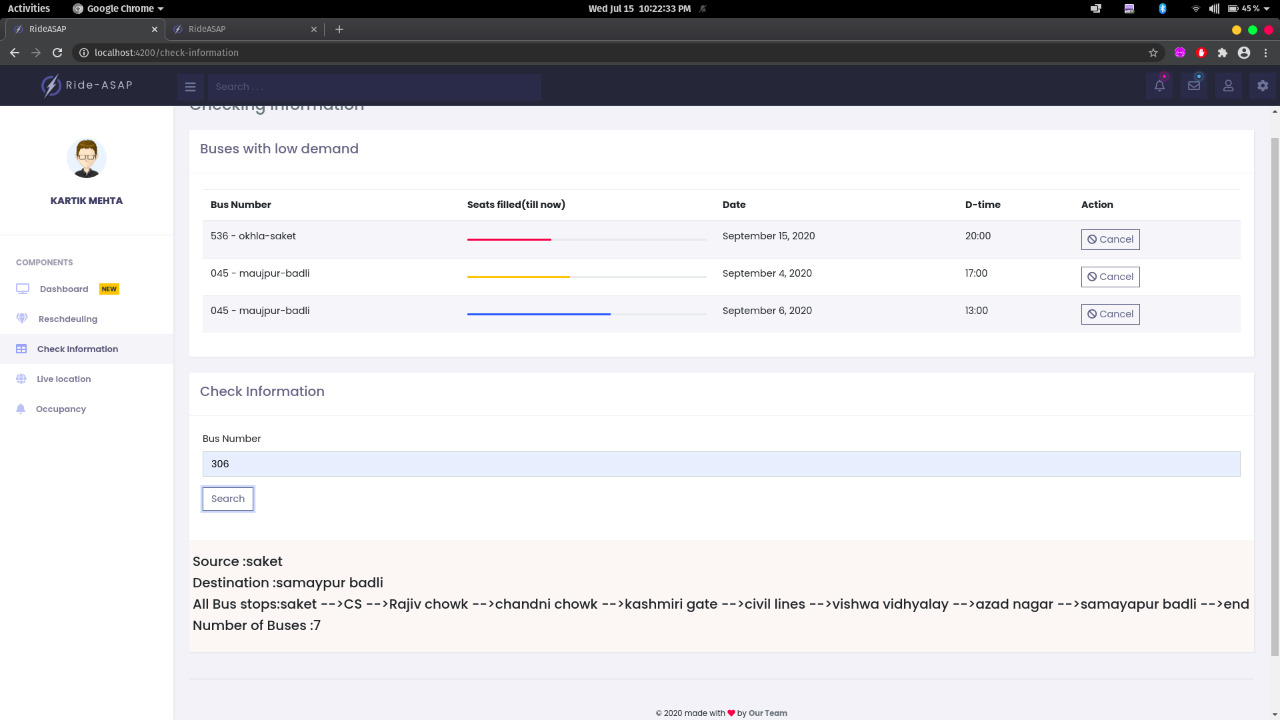
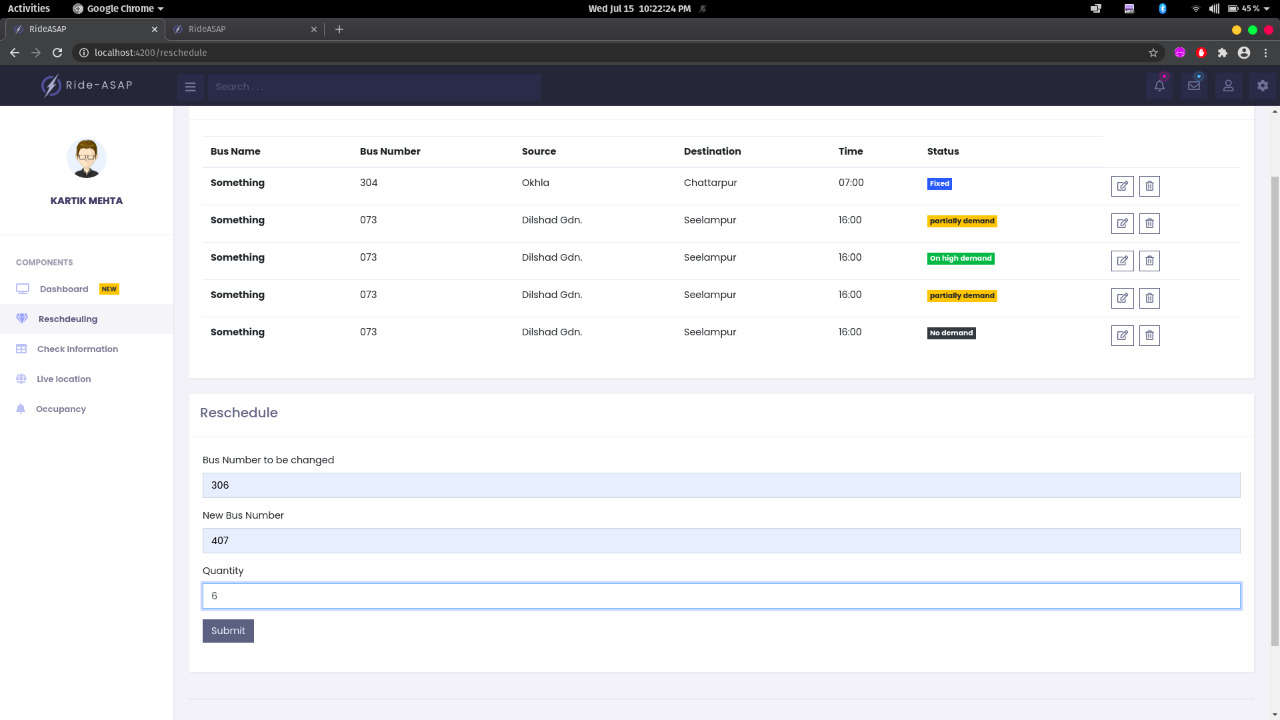
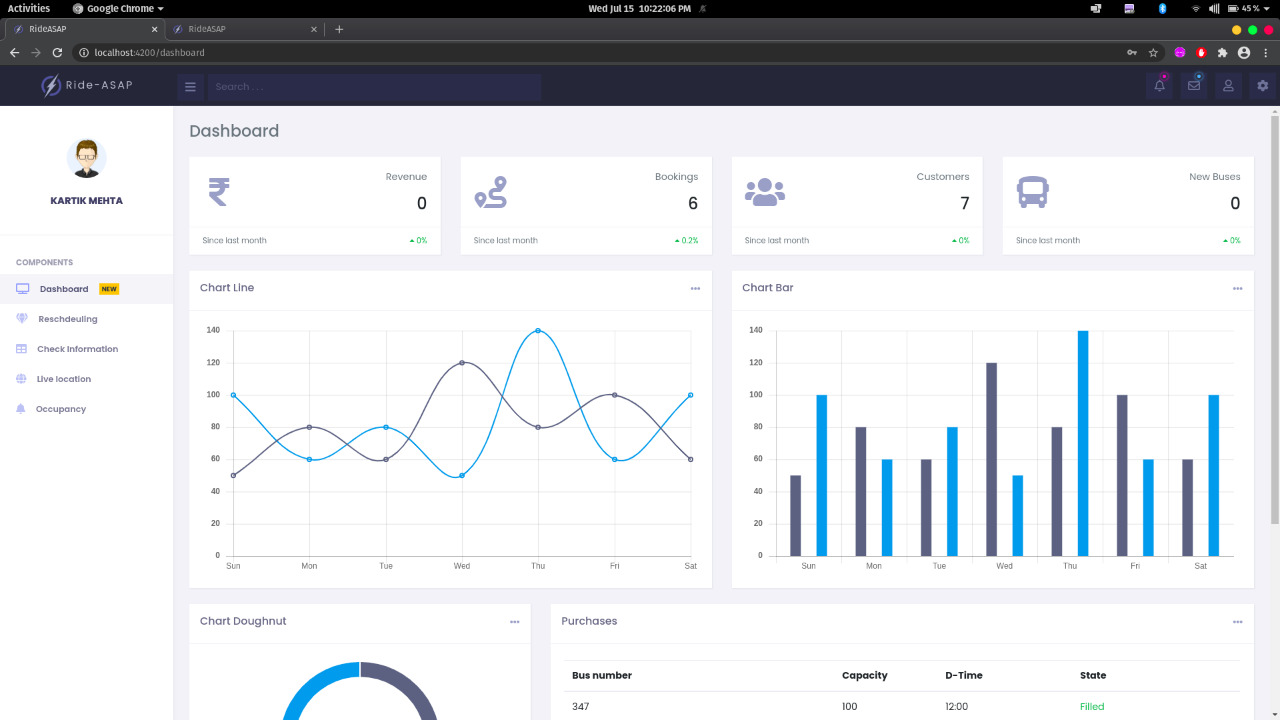




 AUTHORITY SIDE:

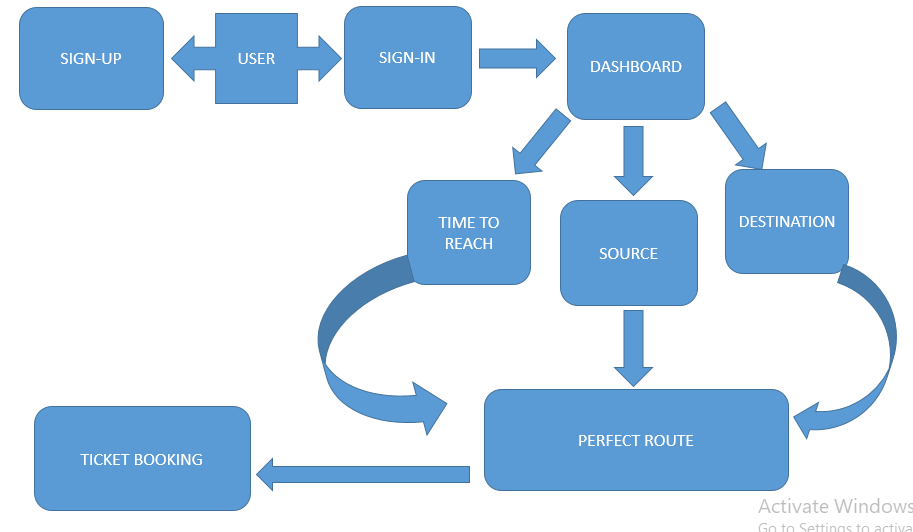


DASHBOARD

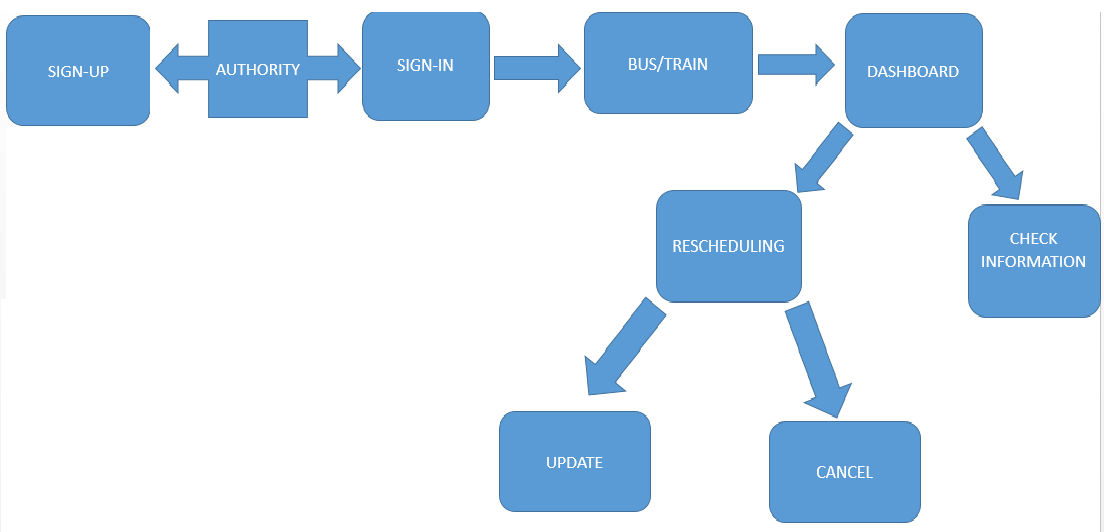


**FLOWCHART**

**For user:**

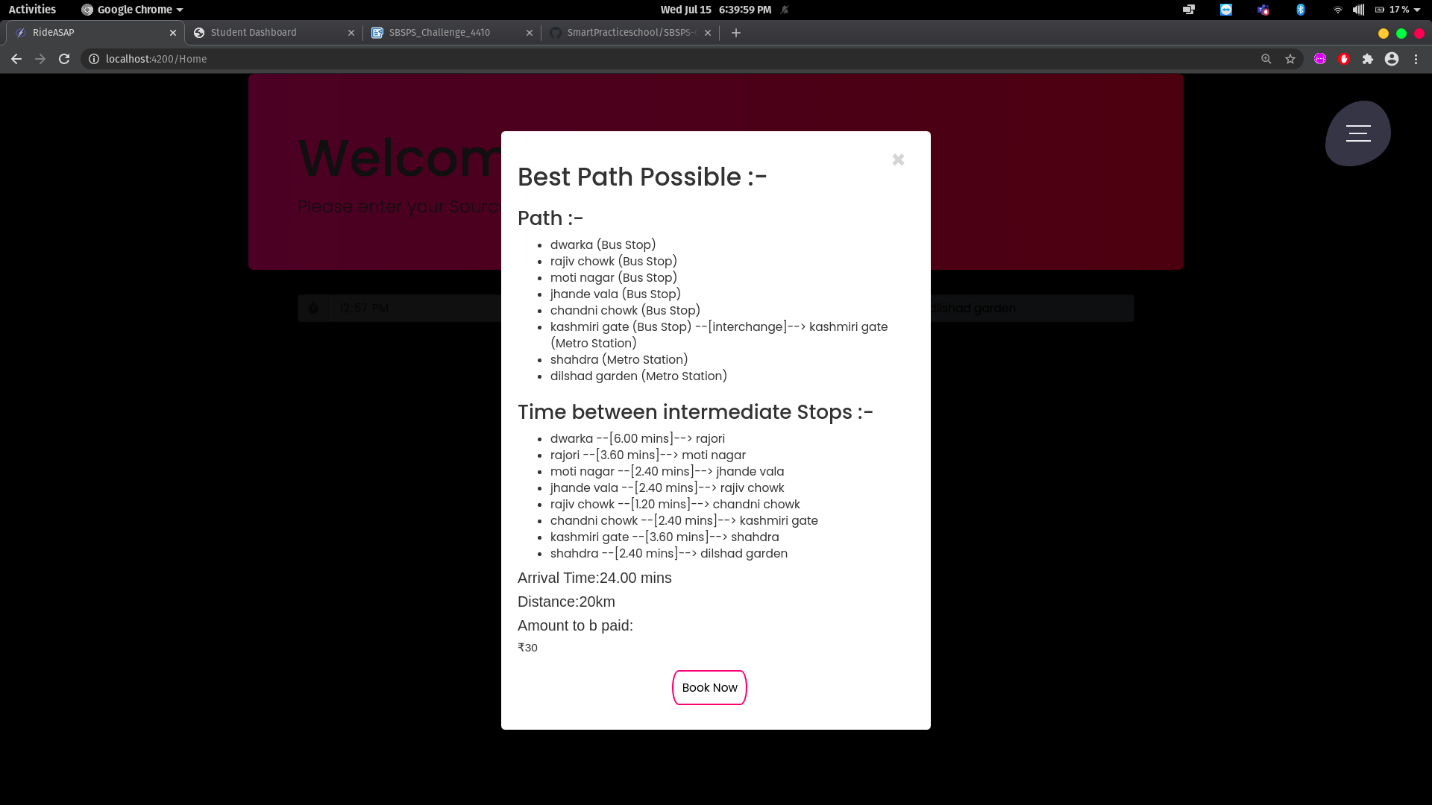


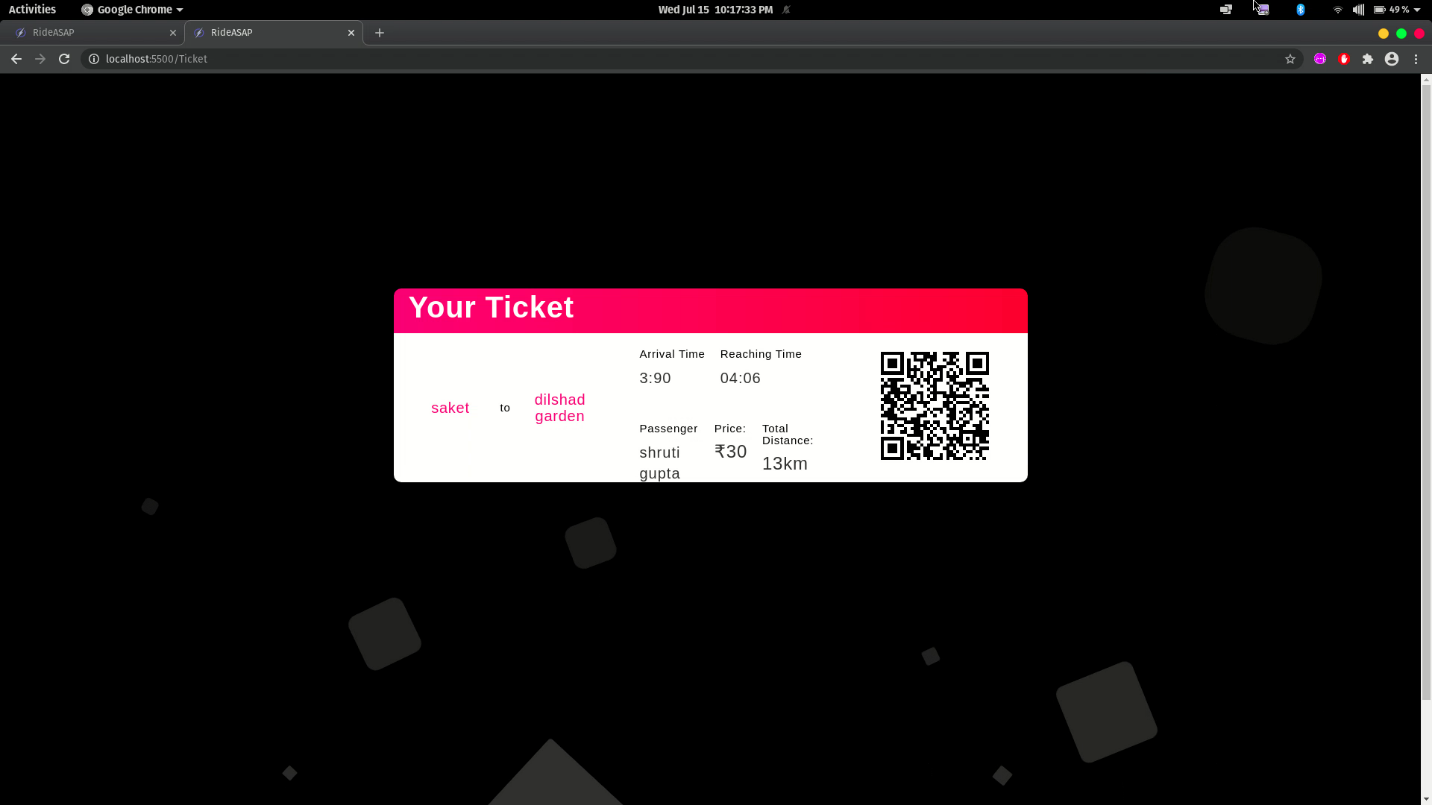
**For authority:**



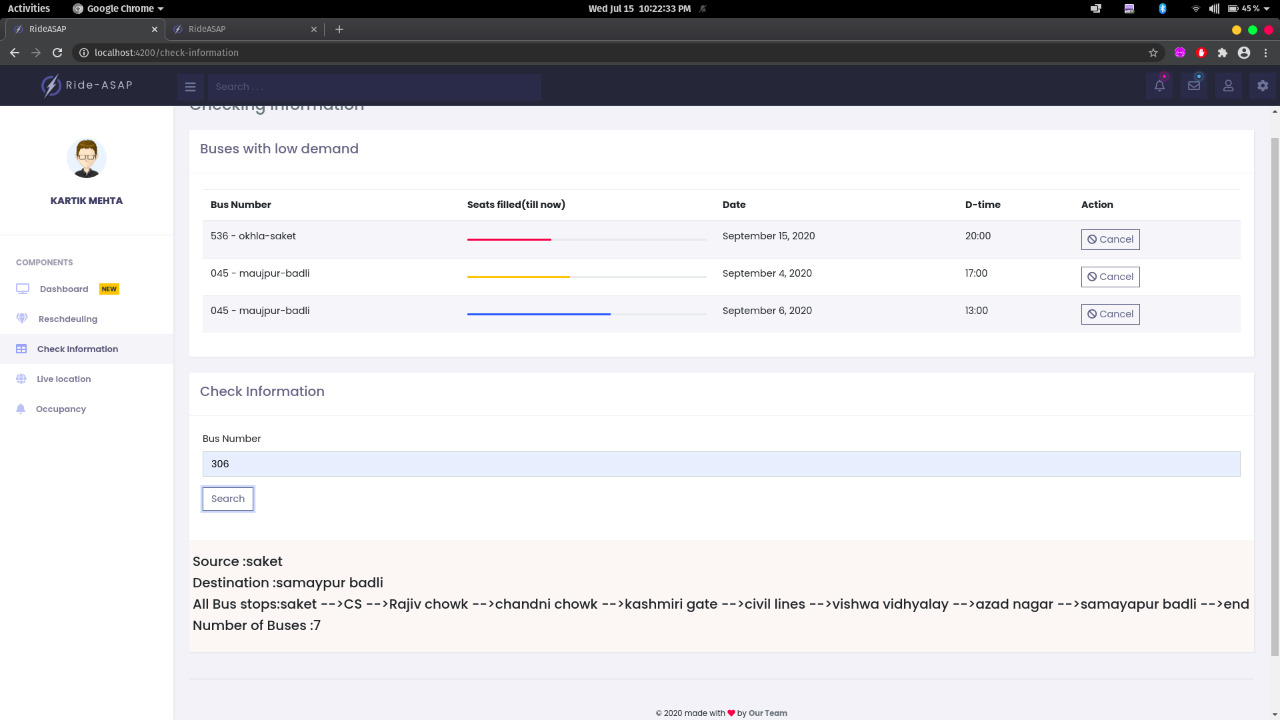
**RESULT**

**Result we get after entering the source, destination and time to reach.....**

 **Ticket generation:**



**Checking information of the transport by the authority:**



**ADVANTAGES & DISADVANTAGES**

**ADVANTAGES: F**

**USER FRIENDLY:**

 Services are easy to use

 Easy to book a ride.

 Suggests best real time route for the user.

 Public as well as private facilities are available

 Calculate the best and shortest time for user

 Individual can check the fare before booking

 Show bus/train/metro timing with best route at one place

 Easy to book e- tickets

ER EFFICIEGREATER EFFICIENCY:

**GREAT EFFICIENCY:**

 Provides end to end route from given source to destination by integrating different modes of transport as required.

 Show bus/train/metro timing with best route at one place

 Single ticket for all means of transport

**MINIMIZE MANUAL WORK:**

 Reduces time to check schedules of different transports on different platforms

 Easy for authority to make changes to transport related information

 Convenient and Cashless  services

**MAINTAIN SOCIAL DISTANCING:**

 Social distancing is maintained for passenger’s security.

 Maintenance of social distancing and reducing the contact between persons.

 Passengers with e-ticket can take advantage of service

 The ticket generated will be in the form of Bar-Code which will reduce the contact between them.

**REDUCE OVER OCCUPANCY:**

 Overcome problem of over occupancy

 Only fixed number of seats are available for booking

**DISADVANTAGES:**

 Fixed price, no concession.

 If seats are not available passenger cannot apply for that service, i.e. no waiting policy.

 Sign-up is mandatory for both authority and passengers.

 Passengers and authorities have to sign-in every time to take advantages of this application.

 Authority can cancel the ride any time.

 Highly dependent on internet.

**APPLICATION**

 For integrating government transportations.

 Management of over -occupancy.

 Management of social distancing while travelling.

 Single ticket for whole integrated journey.

**CONCLUSION**

 Suggests **best real time route** for the user.

**Single ticket** generation for the entire journey.

 Maintain **SOCIAL-DISTANCING** by integrating different modes of transport.

 Overcome the problem of **over-occupancy**

 The transport authorities can integrate together to maintain the system properly.

**FUTURE SCOPE**

**LIVE LOCATION OF THE TRANSPORT**

**PASSENGER'S LIVE LOCATION**

**IMPLEMENT IN ALL OTHER STATES**

**PASSENGERS CAN SELECT ANY ROUTE BY SELECTING FROM MULTIPLE ROUTES**

**MAKE USE OF NEW TECHNOLOGIES,SUCH AS SMARTPHONE APPLICATIONS**

**REAL-TIME PARKING MANAGEMENT**

**ELECTRONIC TOLL COLLECTION**

**COLLISION AVOIDANCE SYSTEM**

**BIBLIOGRAPHY**

 http://www.tsi.lv/sites/default/files/editor/science/Publikacii/RelStat\_13/session\_3\_ed\_poliakova\_ok.pdf

**APPENDIX**

**A.SOURCE CODE:**

**Front-End(Angular)-Route**

import { Component, OnInit } from '@angular/core';

import { Location } from '../travelloc';

import { TravelService } from '../travel.service';

import {Book} from '../book';

import {Router} from '@angular/router';

@Component({

  selector: 'app-mainpage',

  templateUrl: './mainpage.component.html',

  styleUrls: ['./mainpage.component.scss']

})

export class MainpageComponent implements OnInit {

  distance="";

  time="";

  totalPrice="";

  deptime="";

  dispath=[];

  apple= [];

  locate = new Location("","","");

  book = new Book(this.travel.email,this.travel.source,this.travel.destination,this.travel.total\_price);

  constructor(private travel: TravelService,private router: Router) { }

  ngOnInit(): void {

  }

  Openpopup()

  {

    document.getElementById("overlay").style.visibility = "visible";

    document.getElementById("overlay").style.opacity = "1";

    document.getElementById("marketing").style.opacity="0.2";

  }

  Closepopup()

  {

    document.getElementById("overlay").style.visibility = "hidden";

    document.getElementById("overlay").style.opacity = "0";

    document.getElementById("marketing").style.opacity="1";

  }

  sendbook()

  {

    this.book.source=this.travel.source;

    this.book.destination=this.travel.destination;

    this.book.price=this.travel.total\_price;

    this.book.email=this.travel.email;

    console.log(this.book);

    this.travel.bookinfo(this.book)

    .subscribe(

      data =>{

        console.log('Success!',data);

        this.travel.hashqr=data.ticket;

        console.log(this.travel.hashqr);

        this.router.navigate(['/Ticket']);

        },

      error => console.log('Error',error)

    )

  }

  onSubmit()

  {

      this.travel.loc(this.locate)

    .subscribe(

      data =>{

        console.log('Success!',data);

        console.log(data.source);

        var Ppath = Object.keys(data.path)

        // data at book now

        this.travel.source=data.source;

        this.travel.destination=data.destination;

        this.travel.total\_price=data.total\_price;

        this.travel.arrtime=data.arrival\_time;

        this.travel.reachTime=data.reach\_time;

        this.travel.tottime=data.total\_time;

        this.travel.distance=data.total\_distance;

        //

        this.distance=data.total\_distance;

        this.time=data.time\_taken;

        this.deptime=data.arrival\_time;

        this.totalPrice=data.total\_price;

        this.dispath=data.distributed\_path;

        this.apple=[];

        for(Ppath[0] in data.path){

          this.apple.push(data.path[Ppath[0]])

         }

        },

      error => console.log('Error',error)

    )

}

}

service file:

import { async, ComponentFixture, TestBed } from '@angular/core/testing';

import { MainpageComponent } from './mainpage.component';

describe('MainpageComponent', () => {

  let component: MainpageComponent;

  let fixture: ComponentFixture<MainpageComponent>;

  beforeEach(async(() => {

    TestBed.configureTestingModule({

      declarations: [ MainpageComponent ]

    })

    .compileComponents();

  }));

  beforeEach(() => {

    fixture = TestBed.createComponent(MainpageComponent);

    component = fixture.componentInstance;

    fixture.detectChanges();

  });

  it('should create', () => {

    expect(component).toBeTruthy();

  });

});

Sign-Up backend:

// LOGIN

router.post('/login', async (req, res)=>{

    // Validate data before making the user

    const { error } = authority\_login\_validation(req.body);

    if(error){

        console.log(error);

        return res.status(400).send(error.details[0].message);   }

    // CHECKING IF ID\_number EXIST

    const id = await Authority.findOne({id\_number: req.body.id\_number});

    if(!id)

        return res.status(400).send("id number doesn not exist");

    // CHECKING IF EMAIL EXIST

    const authority = await Authority.findOne({email: req.body.email});

    if(!authority)

        return res.status(400).send("email doesn not exist");

    // CHECKING PASSWORD

    const validpass = await bcrypt.compare(req.body.password, authority.password)

    if(!validpass)

        return res.status(400).send('invalid password');

    // CREATE AND ASSIGN TOKEN

    const token = jwt.sign({\_id: authority.\_id}, process.env.AUTHORITY\_TOKEN\_SECRET);

    res.header('authority-token', token).json({token:token,

                                                type: authority.type,

                                            name: authority.full\_name});

})

router.post('/register',  async (req, res)=> {

    // Validate data before making the user

    const { error } = authority\_register\_validation(req.body);

    if(error){

        console.log(error)

        return res.status(400).send(error.details[0].message);}

    // CHECKING IF USER EXIST IN DATABASE

    const emailExist = await Authority.findOne({email: req.body.email});

    if(emailExist)

        return res.status(400).send("email already exist");

    // checking if contact already exist

    const contact\_exist = await Authority.findOne({contact: req.body.contact});

        if(contact\_exist)

            return res.status(400).send("contact already exist!!")

    // PASSWORD HASHING

    const salt = await bcrypt.genSalt(10);

    const hashedpassword = await bcrypt.hash(req.body.password, salt);

    // CREATING USER

    const authority = new Authority({

        full\_name: req.body.full\_name,

        contact: req.body.contact,

        id\_number: req.body.id\_number,

        type: req.body.type,

        email: req.body.email,

        password: hashedpassword

    });

    try{

        const savedAuthority = await authority.save();

        res.send({user\_id: authority.\_id,

                  user\_email: authority.email

        });

    }catch(err){

        res.status(400).send(err);

    }

});

BOOKING API

    // FETCHING USER DETAILS FROM DB

    const email\_exist = await User.findOne({email: req.body.email});

    if(!email\_exist)

        res.send('nope')

    var name =  email\_exist.first\_name + " " + email\_exist.last\_name

    var age = email\_exist.age

    // CREATING TICKET

    const ticket = new Ticket({

        name: name,

        age: age,

        source: source,

        destination: destination,

        price: price

    })

    try{

        const booked = await ticket.save();

    }catch(err){

        res.status(400).send(err)

    }

    let ticket\_id = (ticket.\_id).toString()

    const salt = await bcrypt.genSalt(10);

    const hashed\_Ticket = await bcrypt.hash(ticket\_id, salt)

    let Ddate = ((ticket.date).toString()).split(" ")

Authority rescediuling Bus/Train:

MongoClient.connect(url, { useUnifiedTopology: true }, function (err, db) {

      if (err) throw err;

      var dbo = db.db("ibm6");

      var myquery = { number: prev\_bus\_no };

      var newvalues = { $inc: { quantity: -quantitys } };

      dbo.collection("buses").updateOne(myquery, newvalues, function (err, result) {

        if (err) throw err;

        res.status(200)

      });

      var myquerys = { number: new\_bus\_no };

      var newvaluess = { $inc: { quantity: +quantitys } };

      dbo.collection("buses").updateOne(myquerys, newvaluess, function (err, result) {

        if (err) throw err;

        res.status(200).json('Document updated')

        db.close();

      });

    });

});

router.post('/reschedule/train',  (req, res) => {

  var url = "mongodb://localhost:27017/";

  const number = Number(req.body.number);

  const time = String(req.body.time);

  const name = String(req.body.name);

    MongoClient.connect(url, function (err, db) {

      if (err) throw err;

      var dbo = db.db("ibm6");

      var myquery = { number: number  };

      var newvalues = { $set: { time: time } };

      dbo.collection("trains").updateOne(myquery, newvalues, function (err, result) {

        if (err) throw err;

        res.status(200).json('Document updated')

        db.close();

      });

    });

});