**B.Tech Project Report**

**On**

**Indian Bus**

**Submitted in the partial fulfillment for the award of the degree of**

**BACHELOR OF TECHNOLOGY**

**In**

**Computer Science and Engineering**

**Submitted by**

**Vikas Dhiman (8617111)**

**Under the Supervision of:**

**Mr. Prateek Mishra**

****

**APIIT SD INDIA**

**(Affiliated to Kurukshetra University Kurukshetra, India)**

**August-May | 2020-2021**

**CONTENTS**

|  |  |
| --- | --- |
| **Candidate's Declaration** | i |
| **Certificate** | ii |
| **Acknowledgement** | iv |
| **Abstract** | v |
| **Chapters** | vi |
| **List of Figures** | viii |

**CANDIDATE’S DECLARATION**

I hereby declare that the work presented in this training report entitled **“Indian Bus”**, submitted in partial fulfilment of the requirement for the award of the degree of Bachelor of Technology in **Computer Science And Engineering**, submitted to **Asia Pacific Institute Of Information & Technology, Kurukshetra University, Kurukshetra, India** is anauthentic record of my own work carried out during the period from **1 August, 2020** to **1st May, 2021.**

The work reported in this project report has not been submitted by me for the award of any other degree or diploma.

|  |  |
| --- | --- |
| Date: 01/08/2020 – 01/05/2021  Place: Panipat | Vikas Dhiman (8617111) |

**CERTIFICATE**

This is to certify that the summer training report entitled **“Indian Bus”** done by **Vikas Dhiman , Roll No. 8617111** is an authentic work carried out by him at ICSD under my guidance. The matter embodied in this project work has not been submitted earlier for the award of any degree or diploma to the best of my knowledge and belief.

Mrs. Anshu Sharma

Assistant Professor, CSE

Mr. Prateek Mishra

Training Incharge, CSE

Mr. Ravi Sachidava

HOD , CSE

**ACKNOWLEDGEMENT**

I express my sincere thanks of gratitude to **Mr.Sachin Tuli** , my trainee at **ICSD** who taught me so well that helped me to develop the project **“Indian Bus”** and guided me throughout the project without which it would have been a back-breaking task for me.

It was a great chance of learning which I got through training at ICSD and I consider myself as a very lucky individual to be a part of it. I also got chance to meet many professionals who lead me through my training. I would like to thank**, Mrs. Anshu Sharma (Assistant Professor , APIIT) , Mr. Prateek Mishra (Training Incharge,APIIT)** for guiding me.

I would also not forget to thank my parents and my friends for their co-operation and encouraging me time to time by helping me out with their abilities in completion of my project.

This opportunity was a big milestone in my career. I will try to use my gained skills and knowledge in every possible way and will try to improve more in the same, in order to achieve desired career goals.

(Vikas Dhiman)

**ABSTRACT**

Our project is based on test conduction in which a Indian Bus booking service to get the information about the Bus Service to be choose on the basic of marks which will help them in future.

In this project, we can firstly make advance login and registration and also a social login. And after the user can book the sheet in this project. After there is a payment gateway for the payment service.

After this. The ordered food will be added into his cart and then the payment page can be open at which he can pay the payment for the food and after paying the book the sheet in the bus can be successful and he can also see his bill.

**Features**

* Passenger login
* Bus user login
* Passenger Register
* Bus User register
* Check Your Bus Availability
* Bus Booking
* Check your Bus Status
* Add Bus
* Add Multiple Check Points

**CHAPTER**

|  |  |  |  |
| --- | --- | --- | --- |
| **CHAPTER NO** |  | **DESCRIPTION** | **PAGE NO.** |
| **Chapter 1:** |  | **Introduction** | **1-5** |
| 1.1 |  | Overview | 1-1 |
| 1.2 |  | Introduction of Angular | 1-3 |
| 1.3 |  | Introduction of Spring Boot | 3-4 |
| 1.4 |  | Introduction of Microservices | 4-5 |
| 1.5 |  | Introduction to MySQL | 5-5 |
| **Chapter 2:** |  | **Software and Hardware Requirements** | 6-6 |
| 2.1 |  | Software Requirements | 6-6 |
| 2.2 |  | Hardware Requirements | 6-6 |
| **Chapter 3:** |  | **Software Requirements and Analysis** | 7-8 |
| 3.1 |  | Analysis | 7-8 |
| 3.2 |  | Requirement Specification | 8-8 |
| **Chapter 4:** |  | **Software Design** | **9-11** |
| 4.1 |  | Entity Relationship Diagram | 9-10 |
| 4.2 |  | E-R Diagram | 10-11 |
| **Chapter 5:** |  | **Coding** | **12-37** |
| 5.1 |  | Angular Connectivity with Spring Boot | 12-13 |
| 5.2 |  | UI Code | 13-29 |
| 5.3 |  | Backend Code | 29-37 |
| **Chapter 6:** |  | **Output Screens of the Project** | **38-55** |
| 6.1 |  | Project Output Screens | 38-50 |
| 6.2 |  | Eureka Server Screen | 50-50 |
| 6.3 |  | Database output Screens | 51-55 |
| **Chapter 7:** |  | **References** | **56** |
| 7.1 |  | References Website and links | 56 |

**LIST OF FIGURES**

|  |  |  |
| --- | --- | --- |
| **Figure No.** | **Figure Name** | **Page No.** |
| 1.1 | Angular Icon | 1 |
| 1.2 | Spring Icon | 3 |
| 1.3 | Micro-services Icon | 4 |
| 1.4 | MySQL Icon | 5 |
| 4.2.1 | Diagram for Project Users | 10 |
| 4.2.2 | Diagram for Data Binding | 10 |
| 4.2.3 | Diagram for Data Binding | 11 |
| 5.2.1 | Angular Project Structure | 13 |
| 5.3.1 | Project | 29 |
| 5.3.2 | Eureka Server | 29 |
| 5.3.3 | Zuul Server | 30 |
| 5.3.4 | User Server | 34 |
| 5.3.5 | Bus Server | 36 |
| 6.1.1 | URL | 38 |
| 6.1.2 | Home Page | 38 |
| 6.1.3 | Login Page | 39 |
| 6.1.4 | Register Page | 39 |
| 6.1.5 | Forget Password | 40 |
| 6.1.6 | Login with Social Id’s | 40 |
| 6.1.7 | User Home Page | 41 |
| 6.1.8 | User Profile | 41 |
| 6.1.9 | Add Address | 42 |
| 6.1.10 | Edit Profile | 42 |
| 6.1.11 | Check Bus | 43 |
| 6.1.12 | Book Bus | 43 |
| 6.1.13 | Verify Bill | 44 |
| 6.1.14 | Payment Page | 44 |
| 6.1.15 | Order Placed | 45 |
| 6.1.16 | Contact Page | 45 |
| 6.1.17 | Change Password | 46 |
| 6.1.18 | Booking History | 46 |
| 6.1.19 | Bus Profile | 47 |
| 6.1.20 | Add Bus | 47 |
| 6.1.21 | Bus List | 48 |
| 6.1.22 | Show Main Bus | 48 |
| 6.1.23 | Update Bus | 49 |
| 6.1.24 | Add Check Points | 49 |
| 6.1.25 | Verify Bus Stops | 50 |
| 6.2.1 | Eureka Server | 50 |
| 6.4.1 | Bus Data Table Structure | 51 |
| 6.4.2 | User Table Structure | 52 |
| 6.3.1 | User Details Table | 53 |
| 6.3.2 | Book Seat Booking List | 53 |
| 6.3.3 | User Address | 53 |
| 6.3.4 | User Booking | 53 |
| 6.3.5 | User Booking List no List | 54 |
| 6.3.6 | User Bus Data Details | 54 |
| 6.3.7 | User Bus Payment Status | 54 |
| 6.3.8 | Bus Data | 55 |
| 6.3.9 | Bus Seats | 55 |
| 6.3.10 | Travel Details | 55 |
| 6.3.11 | Bus weeks | 55 |

**Chapter-1**

**Introduction**

**Chapter-1**

**Introduction**

* 1. **Overview**

Indian bus Service, I used to book the seat in this projects. Indian bus service we have a booking seats and manage to seat schedule of the seat service. Indian bus booking system have the very advance booking system. In this booking system have three type of users User, Bus User, Admin User. There have bus users add the bus and add multiple check points for these buses.

* 1. **Introduction to Angular**



**Fig.1.1 Angular Icon**

Angular is a platform and framework for building single-page client applications using HTML and TypeScript. Angular is written in TypeScript. It implements core and optional functionality as a set of TypeScript libraries that you import into your apps.

The architecture of an Angular application relies on certain fundamental concepts. The basic building blocks of the Angular framework are Angular components that are organized into NgModules. NgModules collect related code into functional sets; an Angular app is defined by a set of NgModules. An app always has at least a root module that enables bootstrapping, and typically has many more feature modules.

Components define views, which are sets of screen elements that Angular can choose among and modify according to your program logic and

data.

Components use services, which provide specific functionality not directly related to views. Service providers can be injected into components as dependencies, making your code modular, reusable, and efficient.

An app's components typically define many views, arranged hierarchically. Angular provides the router service to help you define navigation paths among views. The router provides sophisticated in-browser navigational capabilities.

Angular NgModules differ from and complement JavaScript (ES2015) modules. An NgModule declares a compilation context for a set of components that is dedicated to an application domain, a workflow, or a closely related set of capabilities. An NgModule can associate its components with related code, such as services, to form functional units.

Every Angular app has a root module, conventionally named AppModule, which provides the bootstrap mechanism that launches the application. An app typically contains many functional modules.

Like JavaScript modules, NgModules can import functionality from other NgModules, and allow their own functionality to be exported and used by other NgModules. For example, to use the router service in your app, you import the router NgModule.

Organizing your code into distinct functional modules helps in managing development of complex applications, and in designing for reusability. In addition, this technique lets you take advantage of lazy-loading—that is, loading modules on demand—to minimize the amount of code that needs to be loaded at startup.

Every Angular application has at least one component, the *root component* that connects a component hierarchy with the page document object model (DOM). Each component defines a class that contains application data and logic, and is associated with an HTML *template* that defines a view to be displayed in a target environment.

* + 1. **Version**
* Angular JS (Oct 2010)
* Angular 2 (September 2016)
* Angular 3 (December 2016)
* Angular 4 (March 2017)
* Angular 5 (November 2017)
* Angular 6 (May 2018)
* Angular 7 (Oct 2018)
* Angular 8 (Map 2019)
* Angular 9 (Feb 2020)
* Angular 10 (Jun 2020)
* Angular 11 (Nov 2020)
  1. **Introduction of Spring Boot**

****

**Fig.1.2 Spring Icon**

One behalf of the Spring Boot team, and everyone that has contributed, I am pleased to announce that Spring Boot 1.3.0 has been released and is available now from repo.spring.io, Maven Central and Bintray. This release adds a significant number of new features and improvements and builds on the latest release of the Spring Framework. For full upgrade instructions and “new and noteworthy” features please see the release notes.

Here are some of the highlights of v1.3

A new spring boot devtools module has been added which aims to improve the development-time experience. The module provides:

* Sensible property defaults (for example disabling template caches)
* Automatic application restarts
* LiveReload support
* Remote development support (including remote updates and remote debug via an HTTP tunnel).
* Persistent HTTP sessions across restarts

If you’ve got 10 minutes to spare, here’s a short introductory video.

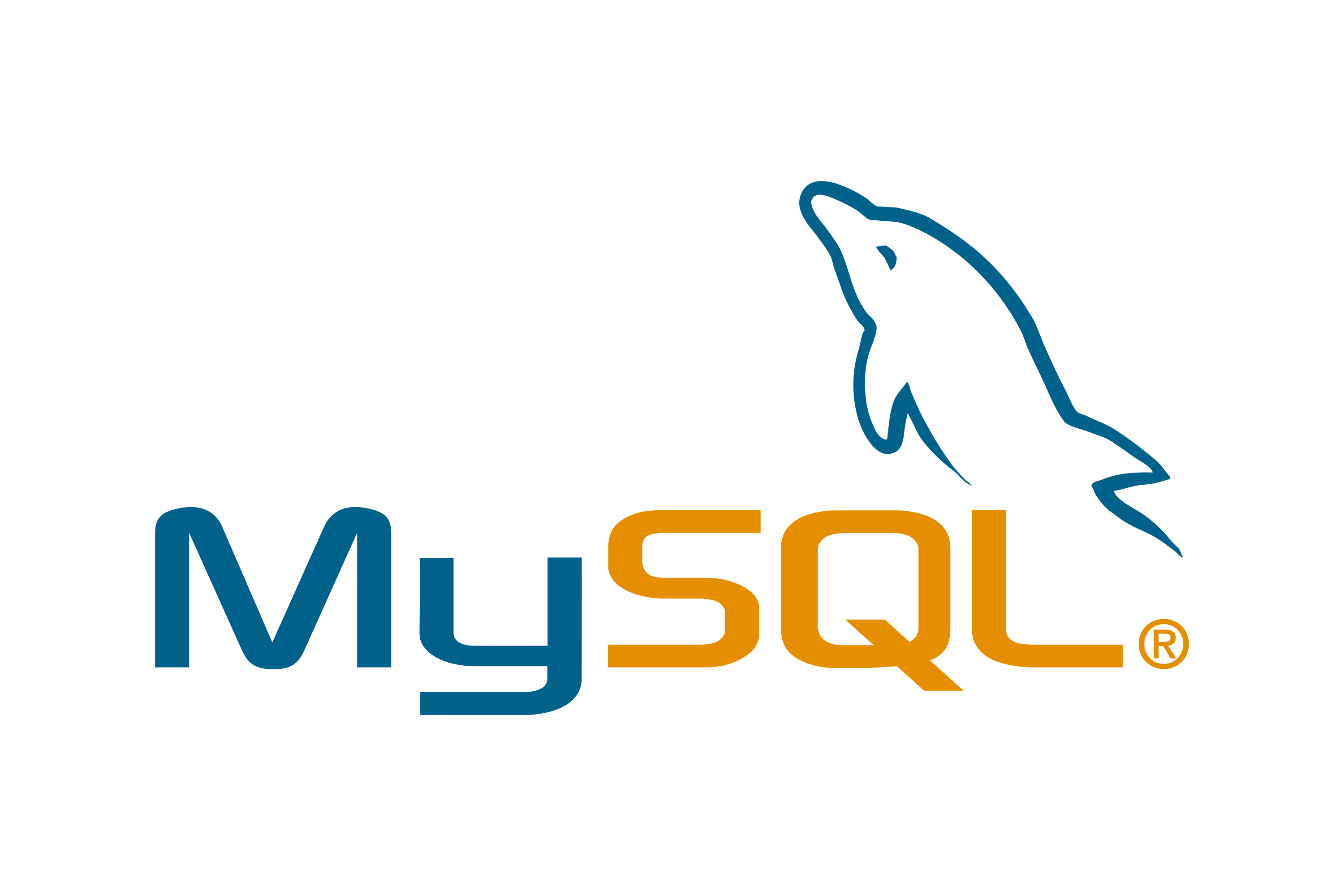
* + 1. **Version**
* 1.0 (2003)
* 2.0 (2006)
* 3.0 (2009)
* 4.0 (2013)
* 5.0 (2017)
* 5.3 (2020)
  1. **Introduction to Microserices**



**Fig.1.3 Microservice Icon**

**Microservices, in a nutshell,** allows us to break our large system into a number of independent collaborating components. which builds on top of Spring Boot, provides a set of features to quickly build microservices. It’s very smart to know how to get them working together, can quickly setup services, with minimal configurations. Things like service registration and discovery, circuit breakers, proxies, logging and log tracking, monitoring, authentication, etc. We’ll walk through the code on how to integrate the common features of Spring Cloud, and we’ll explain each of them. Excited?! So, let’s get started.

* 1. **Introduction to MySQL**



**Fig.1.4 MySQL Icon**

A database is a separate application that stores a collection of data. Each database has one or more distinct APIs for creating, accessing, managing, searching and replicating the data it holds.

Other kinds of data stores can also be used, such as files on the file system or large hash tables in memory but data fetching and writing would not be so fast and easy with those type of systems.

Nowadays, we use relational database management systems (RDBMS) to store and manage huge volume of data. This is called relational database because all the data is stored into different tables and relations are established using primary keys or other keys known as Foreign Keys.

* + 1. **Version**
* MySQL 5.1 (2008)
* MySQL 5.5 (2010)
* MySQL 5.6 (2013)
* MySQL 5.7 (2015)
* MySQL 8.0 (2018)

**Chapter-2**

**Software and Hardware Requirements**

**Chapter-2**

**Software and Hardware Requirements**

This project requires software and hardware which is easily available in the market.Most of the resources came from open source which makes it more interesting and innovative where with investing little money you can earn more.

Mostly java project’s require system with better performance to meet the needs of softwares like normal ram with good computation power.

1. **Software Requirements**

Nowadays, Java is supported by almost every operating systems. whether it is a Windows, Macintosh and Unix all supports the Java application development. So you can download any of the operating system on your personal computer. Here are the minimum requirement.

* Operating System
* Java SDK or JRE 1.6 or higher
* Java Servlet Container (Free Servlet Container available)
* Supported Database and library that supports the database connection with Java.

1. **Hardware Requirements**

Minimum hardware requirement to download Java on your Windows operating system as follows:

* Minimum Windows 95 software
* IBM-compatible 486 system
* Hard Drive and Minimum of 8 MB memory
* A CD-ROM drive
* Mouse, keyboard and sound card, if required

**Chapter-3**

**Software Requirement And Analysis**

**Chapter-3**

**Software Requirement And Analysis**

1. **Analysis**

System analysis is the study of sets of interacting entities, including computer systems analysis. This field is closely related to operations research. It is also "an explicit formal inquiry carried out to help someone (referred to as the decision maker) identify a better course of action and make a better decision than he might otherwise have made". Analysis is defined as the procedure by which we break down an intellectual or substantial whole into parts so that we can achieve our end goals.

The development of a computer-based information system includes a systems analysis phase which produces or enhances the data model which itself is a precursor to creating or enhancing a database.

There are a number of different approaches to system analysis. When a computer-based information system is developed, systems analysis would constitute the following steps:

1. The development of a feasibility study, involving determining whether a project is economically, socially, technologically and organizationally feasible.
2. Conducting fact-finding measures, designed to ascertain the requirements of the system's end-users. These typically span interviews, questionnaires, or visual observations of work on the existing system.
3. Gauging how the end-users would operate the system (in terms of general experience in using computer hardware or software), what the system would be used for etc.

Another view outlines a phased approach to the process. This approach breaks systems analysis into 5 phases:

* + Scope definition
  + Problem analysis
  + Requirements analysis
  + Logical design
  + Decision analysis

User cases are a widely-used systems analysis modelling tool for identifying and expressing the functional requirements of a system. Each use case is a business scenario or event for which the system must provide a defined response. Use cases evolved out of object-oriented analysis.

1. **Requirement Specification**

Information gathering is usually the first phase of the software development project. The purpose of this phase is to identify and document the exact requirements for the system. The user’s request identifies the need for a new information system and on investigation redefined the new problem to be based on MIS, which supports management. The objective is to determine whether the request is valid and feasible before a recommendation is made to build a new or existing manual system continues. The major steps are–

* Defining the user requirements.
* Studying the present system to verify the problem.

Defining the performance expected by the candidate to use requirements.

**Chapter-4**

**Software Design**

**Chapter-4**

**Software Design**

1. **Entity Relationship Diagram**

An Entity Relationship Diagram (ERD) is a graphical tool to express the overall structure of a database. It is based on a perception of a real world which consists of a set of basic objects.

* An entity is a person, place, thing or event of interest to the organization and about which data are captured, stored or processed.
* The attributes are various kinds of data that describes an entity.
* An association of several entities in an Entity-Relationship model is called relationship.

An ERD consists of the following major components:

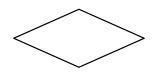
* Rectangles:

Used for representing entity types

* Ellipses:

-

Used for representing attributes

* Diamond:

Used for representing relationship types

* Lines:

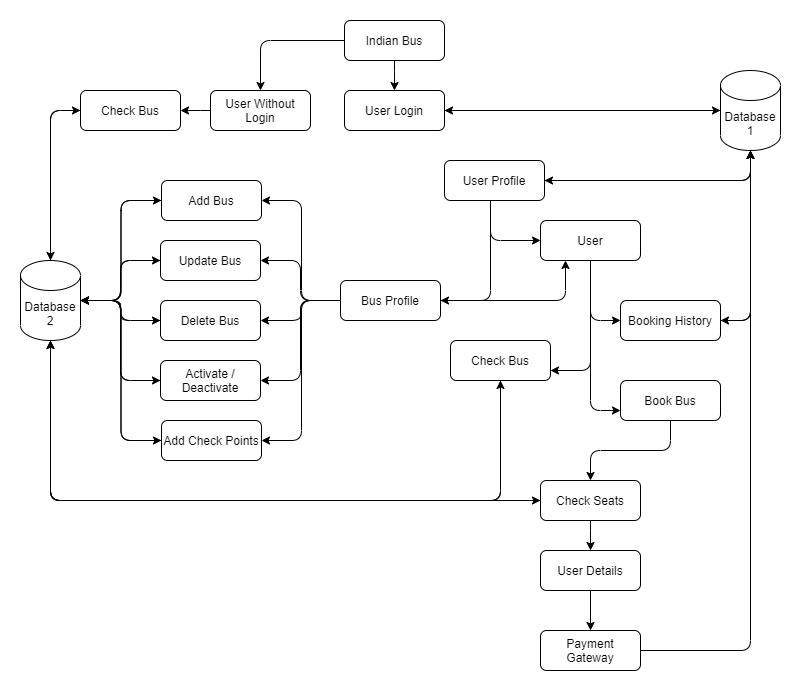


Used for linking attributes to entity types.

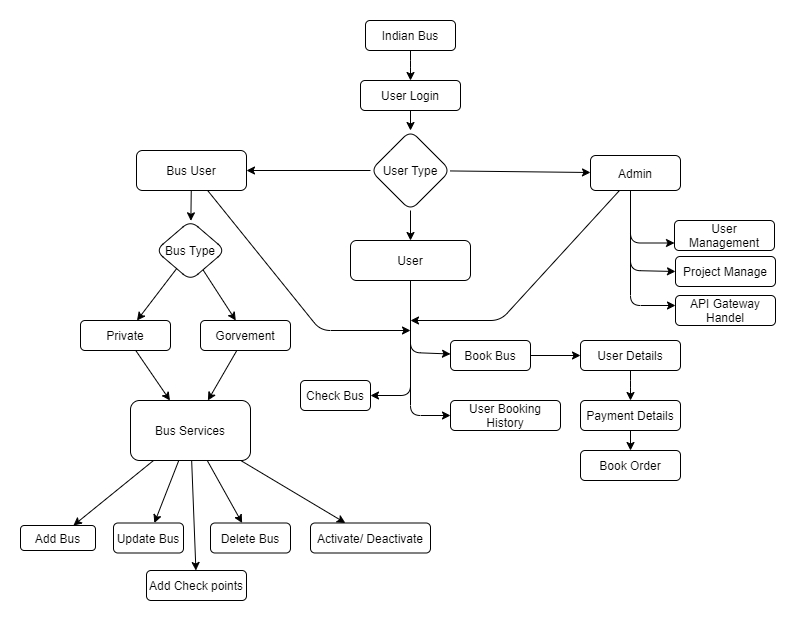
1. **E-R Diagram**



**Fig.4.2.1 Diagram for Project Users**



**Fig.4.2.2 Diagram for Data Binding**



**Fig.4.2.3 Diagram for Data Binding**

**Chapter-5**

**Coding**

**Chapter-5**

**Coding**

The **Indian Bus** project is developed in Angular, Spring Boot, MySQL, Microservice. Which contain multiples packages divided by project.

1. **Angular**
   1. Home
      1. Pipe
      2. Profile
         1. Bus Profile
            1. Bus Working
      3. Search
         1. Book Bus
         2. Payment
   2. Service
   3. Pipe
   4. Guard
   5. Interface
2. **Spring Boot**
   1. Bus\_Eureka\_Server
   2. Bus\_Zuul\_Server
   3. Bus\_User\_Details
   4. Bus\_Bus\_Details
3. **MySQL**
   1. Busdata
   2. Bus\_service\_user\_details
4. **Angular Connectivity with Spring Boot**

Spring Boot and Angular form a powerful tandem that works great for developing web applications with a minimal footprint.

In this tutorial, **we'll use Spring Boot for implementing a RESTful backend, and Angular for creating a JavaScript-based frontend.**

Here are our Spring Boot project's dependencies:

<**dependency**>

<**groupId**>org.springframework.boot</**groupId**>

<**artifactId**>spring-boot-starter-web</**artifactId**>

</**dependency**>

<**dependency**>

<**groupId**>org.springframework.boot</**groupId**>

<**artifactId**>spring-boot-starter-data-jpa</**artifactId**>

</**dependency**>

<**dependency**>

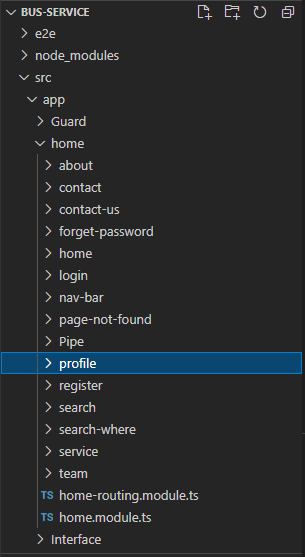
<**groupId**>com.h2database</**groupId**>

<**artifactId**>h2</**artifactId**>

<**scope**>runtime</**scope**>

</**dependency**>

1. **UI Code**

****

**Fig.5.2.1 Angular Project Structure**

**Code**

App-routing.module.ts

import { NgModule } from '@angular/core';

import { Routes, RouterModule } from '@angular/router';

import { PageNotFoundComponent } from './home/page-not-found/page-not-found.component';

const routes: Routes = [

  {

    path:'',redirectTo:'h',pathMatch:'full'

  },

  {

    path:'h',

    loadChildren: ()=> import('./home/home.module').then(m=>m.HomeModule)

  },

  {

    path:'\*\*',component:PageNotFoundComponent

  }

];

@NgModule({

  imports: [RouterModule.forRoot(routes)],

  exports: [RouterModule]

})

export class AppRoutingModule { }

app.component.html

<router-outlet></router-outlet>

App.module.ts

import { BrowserModule } from '@angular/platform-browser';

import { NgModule } from '@angular/core';

import { AppRoutingModule } from './app-routing.module';

import { AppComponent } from './app.component';

import { FormsModule, ReactiveFormsModule } from '@angular/forms';

import { HttpClientModule } from '@angular/common/http';

import { FacebookLoginProvider, GoogleLoginProvider, SocialAuthServiceConfig, SocialLoginModule } from 'angularx-social-login';

import { ToastrModule } from 'ngx-toastr';

import { BrowserAnimationsModule } from '@angular/platform-browser/animations';

@NgModule({

  declarations: [

    AppComponent,

  ],

  imports: [

    BrowserModule,

    FormsModule,

    HttpClientModule,

    SocialLoginModule,

    ReactiveFormsModule,

    AppRoutingModule,

    BrowserAnimationsModule,

    ToastrModule.forRoot(

      {

        timeOut:3000,

      }

    ),

  ],

  providers: [

    {

      provide: 'SocialAuthServiceConfig',

      useValue: {

        autoLogin: false,

        providers: [

          {

            id: GoogleLoginProvider.PROVIDER\_ID,

            provider: new GoogleLoginProvider(

              '976573038155-8fmjdaidg3gf7pb4vg3u2n3f8tbofv4e.apps.googleusercontent.com'

            ),

          },

          {

            id: FacebookLoginProvider.PROVIDER\_ID,

            provider: new FacebookLoginProvider('299821457997851'),

          },

        ],

      } as SocialAuthServiceConfig,

    }

  ],

  bootstrap: [AppComponent]

})

export class AppModule { }

Home.component.html

<ngx-spinner bdColor = "rgba(0,0,0,0.4)" size = "medium" color = "#fff" type = "ball-atom" [fullScreen] = "true"><p style="color: white" > Loading... </p></ngx-spinner>

<app-nav-bar></app-nav-bar>

<header class="masthead text-center text-white d-flex" style="background-image:url('assets/img/header.jpg');">

    <div class="container my-auto">

        <div class="row">

            <div class="col-lg-10 mx-auto">

                <h1 class="text-uppercase"><strong>Indian Bus Service</strong></h1>

                <hr>

            </div>

        </div>

        <div class="col-lg-8 mx-auto">

            <p><strong>Where You Want To Go</strong></p>

            <app-search-where></app-search-where>

        </div>

    </div>

</header>

<section>

    <app-service></app-service>

</section>

<hr>

<app-team></app-team>

<app-about></app-about>

Home.component.ts

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

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

import { NgxSpinnerService } from 'ngx-spinner';

import { URLTrackingService } from 'src/app/Service/url-tracking.service';

import { UserServiceService } from 'src/app/Service/user-service.service';

@Component({

  selector: 'app-home',

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

  styleUrls: ['./home.component.css']

})

export class HomeComponent implements OnInit {

  id:any;

  email:any;

  constructor(private aroute:ActivatedRoute, private service:UserServiceService,private router:Router,

    private urlservice:URLTrackingService,

    private spinner: NgxSpinnerService

    ) {

    this.id=parseInt(this.aroute.snapshot.paramMap.get('id'));

  }

  ngOnInit()

  {

    this.urlservice.checkuserexist();

    this.spinner.show();

    setTimeout(() => {

      this.spinner.hide();

    }, 1000);

  }

}

About.component.html

<div class="row" style="background-color: black; padding-top: 25px; padding-bottom: 25px; color: white; width: 100%;">

    <div class="col-md-12 text-center">

        @copy write by Vikas Dhiman

    </div>

</div>

About.component.ts

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

@Component({

  selector: 'app-about',

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

  styleUrls: ['./about.component.css']

})

export class AboutComponent implements OnInit {

  constructor() { }

  ngOnInit(): void {

  }

}

**Contact.component.html**

**<div class="container">**

**<div class="row">**

**<div class="col-lg-8 mx-auto text-center">**

**<h2 class="section-heading">Let's Get In Touch!</h2>**

**<hr class="my-4">**

**<p class="mb-5">Ready to start your next project with us? That's great! Give us a call or send us an email and we will get back to you as soon as possible!</p>**

**</div>**

**</div>**

**<div class="row">**

**<div class="col-lg-4 ml-auto text-center"><i class="fa fa-phone fa-3x mb-3 sr-contact" data-aos="zoom-in" data-aos-duration="300" data-aos-once="true"></i>**

**<p>+91-9034840635</p>**

**</div>**

**<div class="col-lg-4 mr-auto text-center"><i class="fa fa-envelope-o fa-3x mb-3 sr-contact" data-aos="zoom-in" data-aos-duration="300" data-aos-delay="300" data-aos-once="true"></i>**

**<p><a href="mailto:vikasdhiman835@gmail.com">vikasdhiman835@gmail.com</a></p>**

        </div>

    </div>

</div>

Contact.component.ts

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

@Component({

  selector: 'app-contact',

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

  styleUrls: ['./contact.component.css']

})

export class ContactComponent implements OnInit {

  constructor() { }

  ngOnInit(): void {

  }

}

Contact-us.component.html

<video autoplay muted loop id="myVideo">

    <source src="../../../assets/video/GoogleMaps.mp4" type="video/mp4">

  </video>

<br><br><br>

<div class="container">

    <div class="row text-center">

        <div class="col-md-12">

            <u><h2>Get in touch</h2></u>

        </div>

    </div>

    <br>

    <br>

    <div class="row">

        <div class="col-md-6 box">

            <div class="row text-center">

                <div class="col-md-12">

                    <h3 style="font-weight: bold;">Contact Form</h3>

                </div>

            </div>

            <div class="row">

                <div class="col-md-3">

                    <span class="title">Name: </span>

                </div>

                <div class="col-md-9">

                    <input type="text" class="in"/>

                </div>

            </div>

            <div class="row" style="padding-top: 10px;">

                <div class="col-md-3">

                    <span class="title">Email Id: </span>

                </div>

                <div class="col-md-9">

                    <input type="text" class="in"/>

                </div>

            </div>

            <div class="row" style="padding-top: 10px;">

                <div class="col-md-3">

                    <span class="title">Subject: </span>

                </div>

                <div class="col-md-9">

                    <input type="text" class="in"/>

                </div>

            </div>

            <div class="row" style="padding-top: 10px;">

                <div class="col-md-3">

                    <span class="title">Message: </span>

                </div>

                <div class="col-md-9">

                    <input type="text" class="in"/>

                </div>

            </div>

            <br>

            <div class="row text-right">

                <div class="col-md-12">

                    <input type="submit" class="b" value="Submit">

                </div>

            </div>

        </div>

        <div class="col-md-1"></div>

        <div class="col-md-5 box">

            <div class="row text-center">

                <div class="col-md-12">

                    <h3 style="font-weight: bold;">Address Details</h3>

                </div>

            </div>

            <div class="row">

                <div class="col-md-3">

                    <span class="title">Address:</span>

                </div>

                <div class="col-md-9">

                    <span class="title">H. No. 461, W. No. 15, Near Hanuman Mandir, Gharaunda, Karnal, Haryana, India</span>

                </div>

            </div>

            <div class="row" style="padding-top: 10px;">

                <div class="col-md-3">

                    <span class="title">Email Id:</span>

                </div>

                <div class="col-md-9">

                    <span class="title">vikasdhiman835@gmail.com</span>

                </div>

            </div>

            <div class="row" style="padding-top: 10px;">

                <div class="col-md-3">

                    <span class="title">Phone No:</span>

                </div>

                <div class="col-md-9">

                    <span class="title">9518636811/9034840635</span>

                </div>

            </div>

        </div>

    </div>

</div>

Contact-us.component.ts

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

import { FormBuilder, Validators } from '@angular/forms';

@Component({

  selector: 'app-contact-us',

  templateUrl: './contact-us.component.html',

  styleUrls: ['./contact-us.component.css']

})

export class ContactUsComponent implements OnInit {

  constructor(private fb:FormBuilder) { }

  ngOnInit(): void {

  }

  contactForm=this.fb.group({

    name:['',[Validators.required]],

    email:['',[Validators.required]],

    subject:['',[Validators.required]],

    message:['',[Validators.required]]

  })

  contact()

  {

    console.log(this.contactForm.value);

  }

}

Forgetpassword.component.html

<br>

<br>

<div class="container">

    <div class="row">

        <div class="col-md-12 text-center">

            <h1 style="color: skyblue; font-weight: bold;">Hello, here are three-step to change your password.</h1>

        </div>

    </div>

</div>

<br>

<br>

<div class="row space-rows">

    <div class="col">

        <form [formGroup]="firstForm" (ngSubmit)="first()">

            <div class="card cards-shadown" data-aos="flip-left">

                <div class="card-body">

                    <br>

                    <div class="group">

                        <input type="text" formControlName="email">

                        <span class="highlight"></span>

                        <span class="bar"></span>

                        <label>Email</label>

                    </div>

                </div>

                    <input type="submit" \*ngIf="message1==null" [disabled]="!firstForm.valid" style="width: 120px; border: none;margin-left: 25px;" class="btn btn-primary" value="Next">

                <br>

            </div>

        </form>

    </div>

    <div class="col">

        <form [formGroup]="secoundForm" (ngSubmit)="secound()">

        <div class="card cards-shadown" data-aos="slide-right">

            <div class="card-body">

                <br>

                <div class="group">

                    <input type="password" formControlName="otp">

                    <span class="highlight"></span>

                    <span class="bar"></span>

                    <label>OTP</label>

                </div>

                {{message2}}

            </div>

                <input type="submit" \*ngIf="message2==null" [disabled]="!secoundForm.valid" style="width: 120px; border: none;margin-left: 25px;" class="btn btn-primary" value="Verify">

                <br>

        </div>

        </form>

    </div>

    <div class="col">

        <form [formGroup]="thirdForm" (ngSubmit)="third()">

            <div class="card cards-shadown" data-aos="flip-up">

                <div class="card-body">

                    <br>

                    <div class="group">

                        <input type="password" formControlName="newpassword">

                        <span class="highlight"></span>

                        <span class="bar"></span>

                        <label>New Password</label>

                    </div>

                    <div class="group">

                        <input type="password" formControlName="confirmpassword">

                        <span class="highlight"></span>

                        <span class="bar"></span>

                        <label>Confirm Password</label>

                    </div>

                    {{message3}}

                </div>

                    <input type="submit" \*ngIf="message3==null" [disabled]="!thirdForm.valid" style="width: 120px; border: none;margin-left: 25px;" class="btn btn-success" value="Update">

                    <br>

            </div>

        </form>

    </div>

</div>

Forgetpassword.comonent.ts

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

import { FormBuilder, Validators } from '@angular/forms';

import { Location } from "@angular/common";

import { passwordValidator } from 'src/app/shared/password-validator';

import { ForgetPasswordServiceService } from 'src/app/Service/forget-password-service.service';

import { ToastrService } from 'ngx-toastr';

@Component({

  selector: 'app-forget-password',

  templateUrl: './forget-password.component.html',

  styleUrls: ['./forget-password.component.css']

})

export class ForgetPasswordComponent implements OnInit {

  message1:any;

  message12:any;

  code:any;

  emails:any;

  message2:any;

  message3:any;

  constructor(private fb:FormBuilder,

    private service:ForgetPasswordServiceService,

    private toastr: ToastrService,

    private loc:Location) { }

  ngOnInit(){

  }

  firstForm=this.fb.group({

    email:['',[Validators.required,Validators.minLength(3)]]

  });

  secoundForm=this.fb.group({

    otp:['',[Validators.required,Validators.minLength(2),Validators.maxLength(4)]]

  });

  thirdForm=this.fb.group({

    email:[''],

    newpassword:['',[Validators.required,Validators.minLength(3)]],

    confirmpassword:['',[Validators.required,Validators.minLength(3)]]

  },{validator: passwordValidator});

  first()

  {

    this.emails=this.firstForm.value.email;

    this.service.verifyemail(this.firstForm.value).subscribe((Response)=>

    {

      this.message1=Response;

      this.message12=this.message1.message;

      this.code=this.message1.code;

      this.showmessage();

    })

  }

  secound()

  {

    if(this.code==this.secoundForm.value.otp)

    {

      this.toastr.success('Verification Successful', 'Success');

    }

    else

    {

      this.toastr.error("OTP is incorrect");

    }

  }

  third()

  {

    this.thirdForm.value.email=this.emails;

    this.service.updatepassword(this.thirdForm.value).subscribe((Response)=>

    {

      this.message3=Response;

      this.loc.back();

    });

  }

  showmessage()

  {

    if(this.message12=="User is not exist")

    {

      this.toastr.error(this.message12, 'Check your Email');

    }

    if(this.message12=="Mail send Successful")

    {

      this.toastr.success('Check your mail, there have a verification code', 'Success');

    }

  }

}

Login.comonent.html

<ngx-spinner bdColor = "rgba(0,0,0,0.4)" size = "medium" color = "#fff" type = "ball-atom" [fullScreen] = "true"><p style="color: white" > Loading... </p></ngx-spinner>

<div class="container-fluid">

    <div class="row mh-100vh">

        <div class="col-10 col-sm-8 col-md-6 col-lg-6 offset-1 offset-sm-2 offset-md-3 offset-lg-0 align-self-center d-lg-flex align-items-lg-center align-self-lg-stretch bg-white p-5 rounded rounded-lg-0 my-5 my-lg-0" id="login-block">

            <div class="m-auto w-lg-75 w-xl-50">

                <h2 class="text-info font-weight-light mb-5"><i class="fa fa-diamond"></i>&nbsp;Bus Service</h2>

                <form [formGroup]="loginForm" (ngSubmit)="login()">

                    <div class="group">

                        <input type="text" formControlName="email"/>

                        <span class="highlight"></span>

                        <span class="bar"></span>

                        <label>Email</label>

                    </div>

                    <div class="group">

                        <input type="password" formControlName="password">

                        <span class="highlight"></span>

                        <span class="bar"></span>

                        <label>Password</label>

                    </div>

                    <button class="btn btn-info mt-2" [disabled]="loginForm.invalid" type="submit">Log In</button>

                    <span><img width="15%" class="img text-center" \*ngIf="status" src="../../../assets/loader/1.gif"></span>

                    <button class="btn btn-success mt-2 float-right" type="button" (click)="register()">Register</button>

                </form>

                <br>

                <div class="row">

                    <div class="col-md-12 text-center">

                        <button style="border-radius: 20px; height: 40px; " class="btn btn-danger btn-social-icon btn-google mx-1" (click)="signInWithGoogle()"><i class="fa fa-google"></i></button>

                        <button style="border-radius: 20px; height: 40px;" class="btn btn-primary btn-social-icon btn-facebook mx-1" (click)="signInWithFB()"><i class="fa fa-facebook"></i></button>

                    </div>

                </div>

                <p style="cursor: pointer;" class="mt-3 mb-0"><a class="text-info small" (click)="forget()">Forgot your email or password?</a></p>

            </div>

        </div>

        <div class="col-lg-6 d-flex align-items-end" id="bg-block" style="background-image:url(&quot;assets/img/aldain-austria-316143-unsplash.jpg&quot;);background-size:cover;background-position:center center;">

            <p class="ml-auto small text-dark mb-2"><a class="text-dark" href="https://unsplash.com/photos/v0zVmWULYTg?utm\_source=unsplash&amp;utm\_medium=referral&amp;utm\_content=creditCopyText" target="\_blank"></a><br></p>

        </div>

    </div>

</div>

Login.comonent.ts

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

import { FormBuilder, Validators } from '@angular/forms';

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

import { FacebookLoginProvider, GoogleLoginProvider, SocialAuthService, SocialUser } from 'angularx-social-login';

import { UserServiceService } from 'src/app/Service/user-service.service';

import { ToastrService } from 'ngx-toastr';

import { NgxSpinnerService } from 'ngx-spinner';

@Component({

  selector: 'app-login',

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

  styleUrls: ['./login.component.css']

})

export class LoginComponent implements OnInit {

  message:any;

  user: SocialUser;

  email:any;

  status:boolean=false;

  loginsession:any;

  constructor(private router:Router,

    private toastr: ToastrService,

    private fb:FormBuilder,

    private service:UserServiceService,

    private authService: SocialAuthService,

    private spinner: NgxSpinnerService) { }

  ngOnInit(): void {

    this.spinner.show();

    setTimeout(() => {

      this.spinner.hide();

    }, 500);

    this.authService.authState.subscribe(user => {

      this.user = user;

      this.email=user.email;

      this.checkloginemail();

    });

  }

  loginForm=this.fb.group({

    email: ['',[Validators.required]],

    password: ['',[Validators.required]]

  });

  signInWithGoogle(): void {

    this.authService.signIn(GoogleLoginProvider.PROVIDER\_ID);

  }

  signInWithFB(): void {

    this.authService.signIn(FacebookLoginProvider.PROVIDER\_ID);

  }

  register()

  {

    this.router.navigate(['/h/register']);

  }

  login()

  {

    this.spinner.show();

    this.service.login(this.loginForm.value).subscribe((Response)=>

    {

      this.loginsession=Response;

      let id=this.loginsession.userid;

      this.message=this.loginsession.message;

      this.email=this.loginsession.email;

      this.checkmessage();

      setTimeout(() => {

        this.spinner.hide();

      }, 1000);

    });

  }

  checkloginemail()

  {

    this.spinner.show();

    this.service.loginsoc(this.user).subscribe((Response)=>

    {

      this.loginsession=Response;

      this.message=this.loginsession.message;

      this.email=this.loginsession.email;

      this.checkmessage();

      setTimeout(() => {

        this.spinner.hide();

      }, 1000);

    });

  }

  forget()

  {

    this.router.navigate(['/h/forgetpassword']);

  }

  checkmessage()

  {

    if(this.message=="Password is Incorrect")

    {

      this.status=false;

      this.toastr.error(this.message, 'Check your password');

    }

    if(this.message=="User exist")

    {

      this.status=false;

      this.toastr.success('Login Successfull', 'Success');

      localStorage.setItem('userdata', JSON.stringify(this.loginsession.user));

      localStorage.setItem('email',this.email);

      this.router.navigate(['/h/u',this.email]);

    }

    if(this.message=="User is not exist")

    {

      this.status=false;

      this.toastr.error(this.message, 'Please Register');

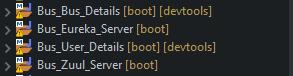
    }

  }

}

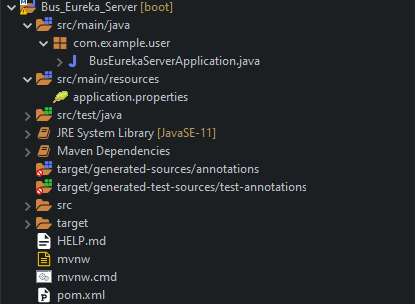
1. **Backend Code**

* **Project’s**

****

**Fig 5.3.1 (Project’s)**

* **Eureka Server**

****

**Fig 5.3.2 (Eureka Server)**

BusEurekaServerMain.java

package com.example.user;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;

@SpringBootApplication

EnableEurekaServer

public class BusEurekaServerApplication {

public static void main(String[] args) { SpringApplication.run(BusEurekaServerApplication.class, args);

}}

Application.properties

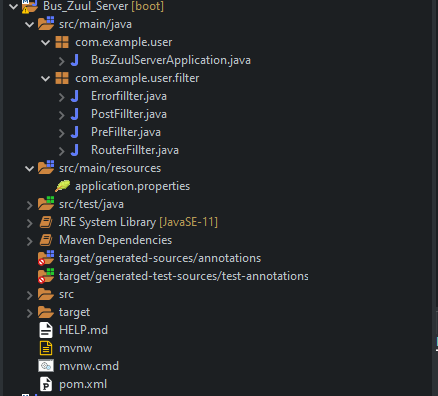
spring.application.name=eureka

server.port=8761

eureka.client.registerWithEureka= false

eureka.client.fetchRegistry= false

* **Zuul Server**



**Fig 5.3.3 (Zuul Server)**

BusZuulServerMain.java

package com.example.user;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.cloud.netflix.eureka.EnableEurekaClient;

import org.springframework.cloud.netflix.zuul.EnableZuulProxy;

import org.springframework.context.annotation.Bean;

import com.example.user.filter.Errorfillter;

import com.example.user.filter.PostFillter;

import com.example.user.filter.PreFillter;

import com.example.user.filter.RouterFillter;

@SpringBootApplication

@EnableEurekaClient

@EnableZuulProxy

public class BusZuulServerApplication {

public static void main(String[] args) {

SpringApplication.run(BusZuulServerApplication.class, args);

}

@Bean

public PreFillter pre()

{

return new PreFillter();

}

@Bean

public PostFillter post()

{

return new PostFillter();

}

@Bean

public Errorfillter error()

{

return new Errorfillter();

}

@Bean

public RouterFillter router()

{

return new RouterFillter();

}

}

ErrorFilter.java

package com.example.user.filter;

import com.netflix.zuul.ZuulFilter;

import com.netflix.zuul.context.RequestContext;

import javax.servlet.http.HttpServletResponse;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

public class Errorfillter extends ZuulFilter

{

private static Logger log = LoggerFactory.getLogger(PostFillter.class);

@Override

public String filterType() {

return "error";

}

@Override

public int filterOrder() {

return 1;

}

@Override

public boolean shouldFilter() {

return true;

}

@Override

public Object run() {

HttpServletResponse response = RequestContext.getCurrentContext().getResponse();

log.info("ErrorFilter: " + String.format("response status is %d", response.getStatus()));

return null;

}

}

Application.properties

#application details

spring.application.name=zuulserver

server.port=8095

# eureka related information-----------------

eureka.client.serviceUrl.defaultZone=http://localhost:8761/eureka/

eureka.client.register-with-eureka=true

eureka.client.fetch-registry=true

#userappliaction

zuul.routes.userdetails.path=/userdetails/\*\*

zuul.routes.userdetails.service-id=userdetails

zuul.routes.userdetails.stripPrefix: false

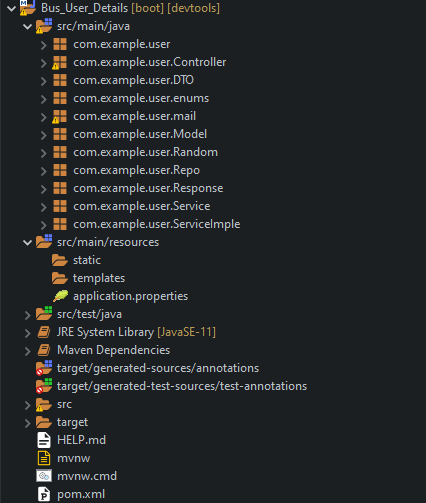
#busapplication

zuul.routes.busdata.path=/busdata/\*\*

zuul.routes.busdata.service-id=busdata

zuul.routes.busdata.stripPrefix: false

* **User Server**

****

**Fig 5.3.4 (User Server)**

BusUserDetailsApplication.java

package com.example.user;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.cloud.netflix.eureka.EnableEurekaClient;

@SpringBootApplication

@EnableEurekaClient

public class BusUserDetailsApplication {

public static void main(String[] args) {

SpringApplication.run(BusUserDetailsApplication.class, args);

}

}

Application.properties

#Zuul Application

spring.application.name=userdetails

#server.servlet.context-path=/userdetails

#eureka URL

eureka.client.register-with-eureka=true

eureka.client.fetch-registry=true

eureka.client.serviceUrl.defaultZone= http://localhost:8761/eureka/

spring.jpa.show-sql=true

server.port=8910

#server.address=localhost

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL5InnoDBDialect

spring.jpa.hibernate.ddl-auto=update

spring.datasource.url = jdbc:mysql://localhost:3306/bus\_service\_user\_details

spring.datasource.username = XXXXXXXXX

spring.datasource.password = XXXXXXXX

#mail service

spring.mail.host=smtp.gmail.com

spring.mail.port=587

spring.mail.username=XXXXXXXXXXXXXXXXXXXXXX

spring.mail.password=XXXXXXXXXXXXXX

spring.mail.properties.mail.smtp.starttls.enable=true

debug=true

# Other mail properties

spring.mail.properties.mail.debug=true

spring.mail.properties.mail.transport.protocol=smtp

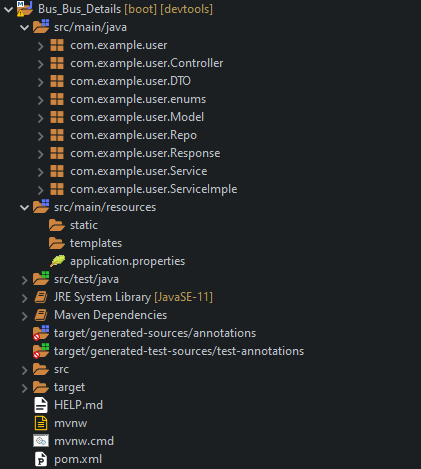
spring.mail.properties.mail.smtp.auth=true

spring.mail.properties.mail.smtp.connectiontimeout=5000

spring.mail.properties.mail.smtp.timeout=5000

spring.mail.properties.mail.smtp.writetimeout=5000

* **Bus Server**

****

**Fig 5.3.5 (Bus Server)**

BusBusDetailsApplication.java

package com.example.user;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.cloud.netflix.eureka.EnableEurekaClient;

@SpringBootApplication

@EnableEurekaClient

public class BusBusDetailsApplication {

public static void main(String[] args) {

SpringApplication.run(BusBusDetailsApplication.class, args);

}

}

Application.properties

spring.application.name=busdata

spring.jpa.show-sql=true

server.port=8908

#server.address=localhost

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL5InnoDBDialect

spring.jpa.hibernate.ddl-auto=update

spring.datasource.url = jdbc:mysql://localhost:3306/busdata

spring.datasource.username = XXXXXXXXX

spring.datasource.password = XXXXXXXXXXXXX

#eureka server

eureka.client.register-with-eureka=true

eureka.client.fetch-registry=true

eureka.client.serviceUrl.defaultZone= http://localhost:8761/eureka/

**Chapter-6**

**Output Screens of the Project**

**Chapter-6**

**Output Screens of the Project**

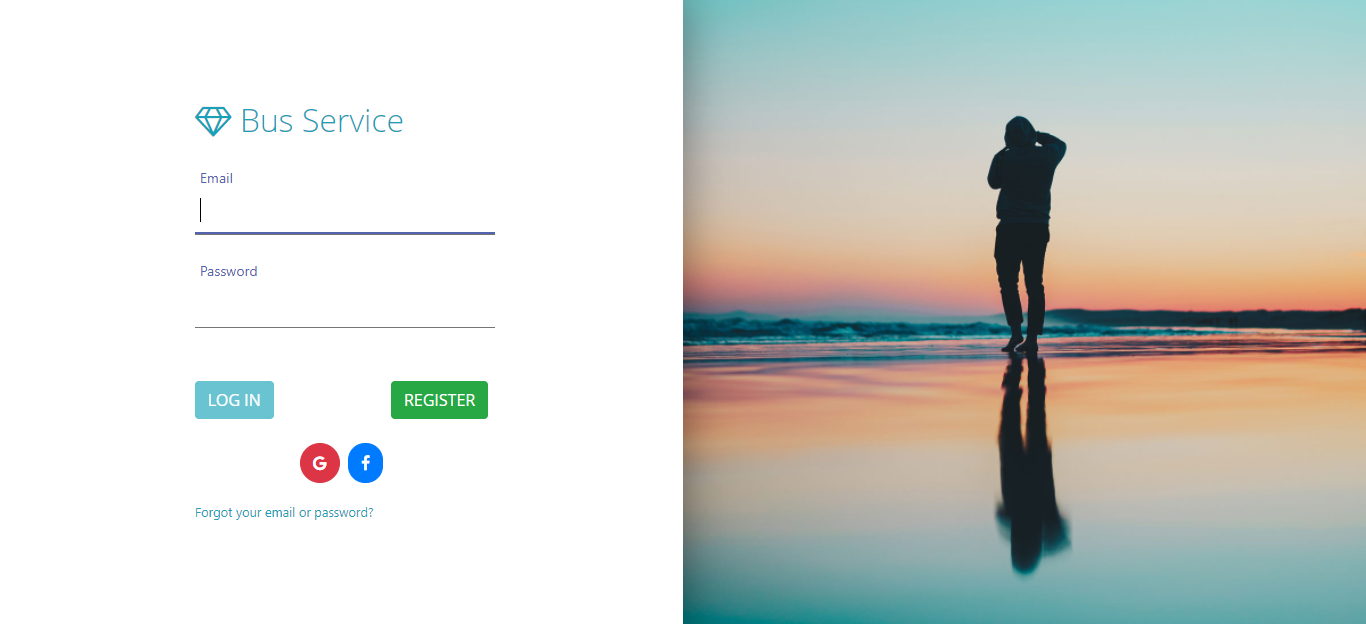
1. **Project Output Screens**

****

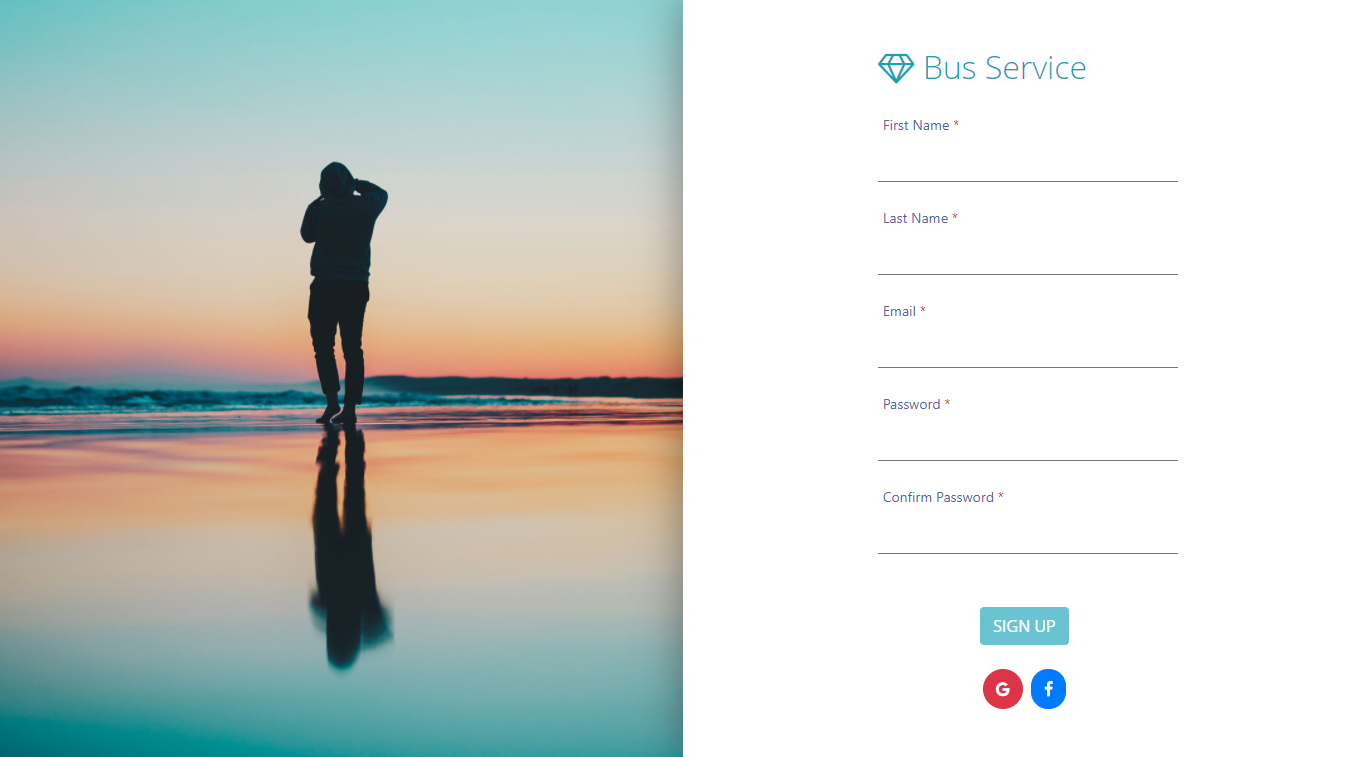
**Figure no: 6.1.1 (URL (Running in browser))**

****

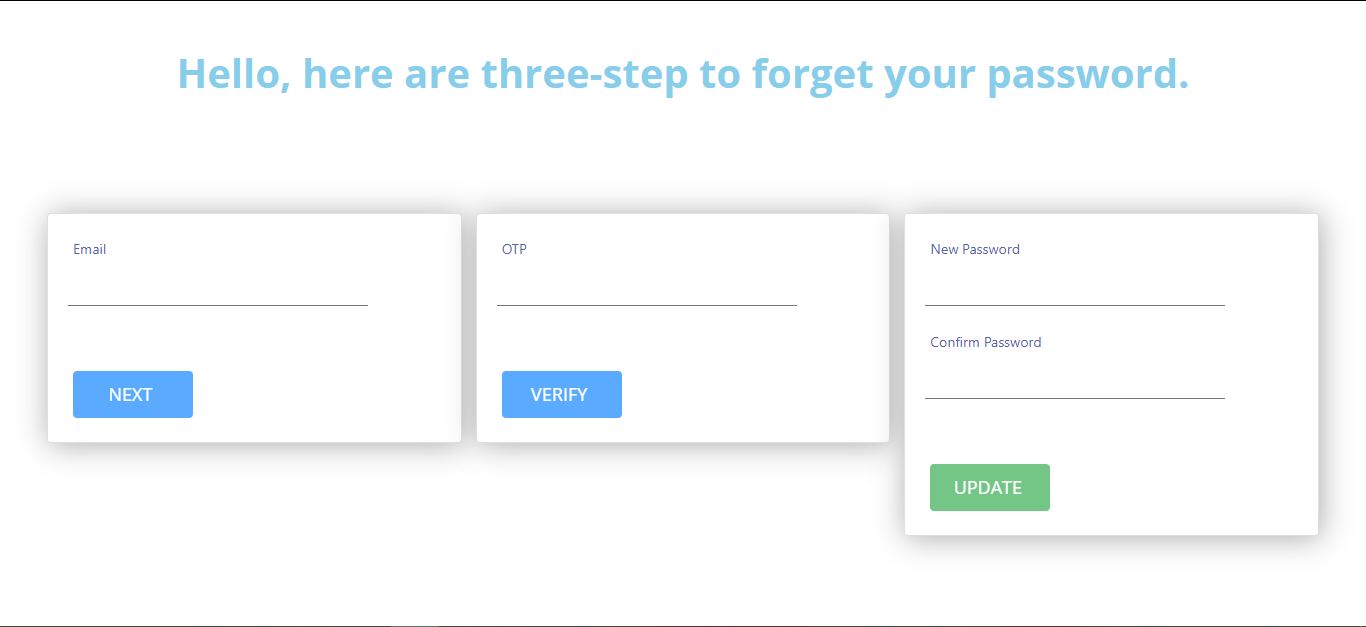
**Figure no: 6.1.2 (Home Page)**

****

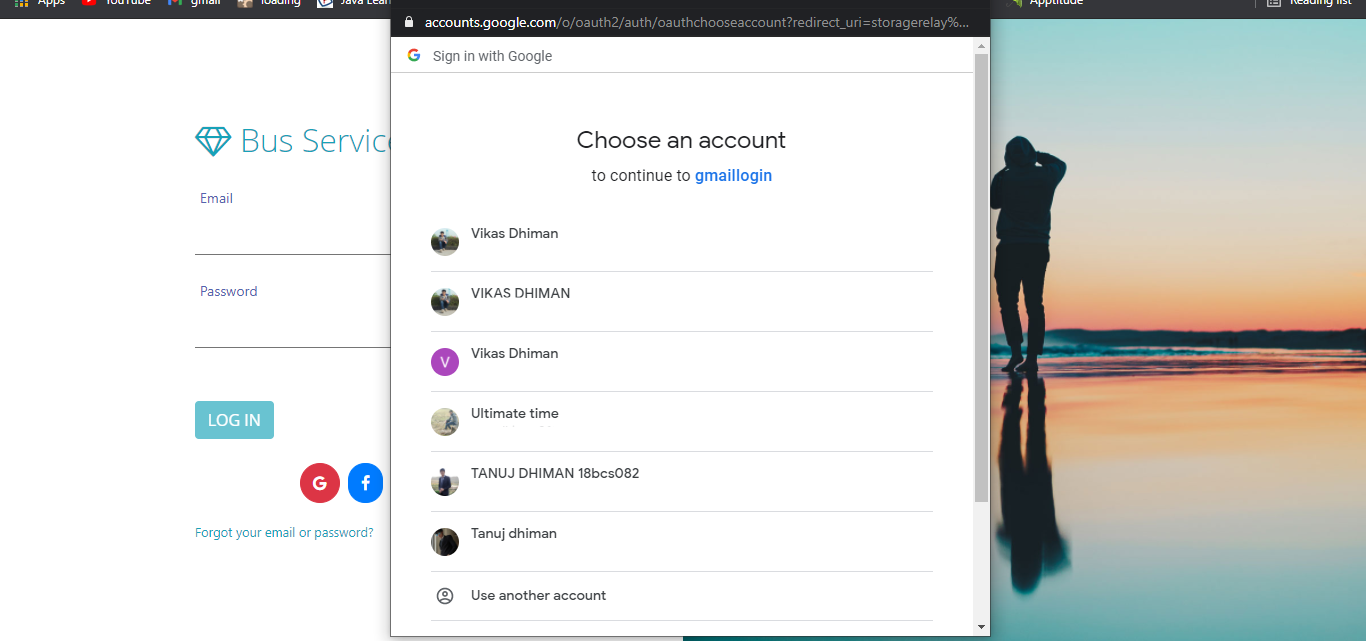
**Figure no: 6.1.3 (Login Page)**

****

**Figure no: 6.1.4 (Register Page)**

****

**Figure no: 6.1.5 (Forget Password)**

****

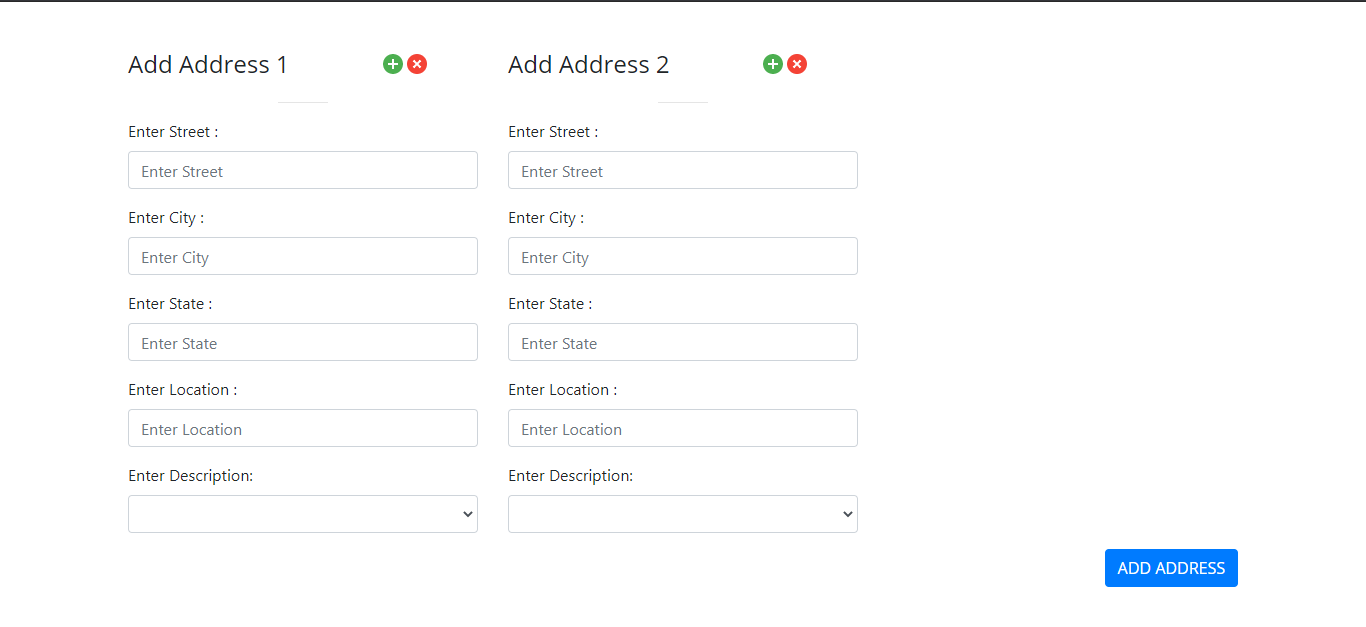
**Figure no: 6.1.6 (Login with Social Id’s)**

****

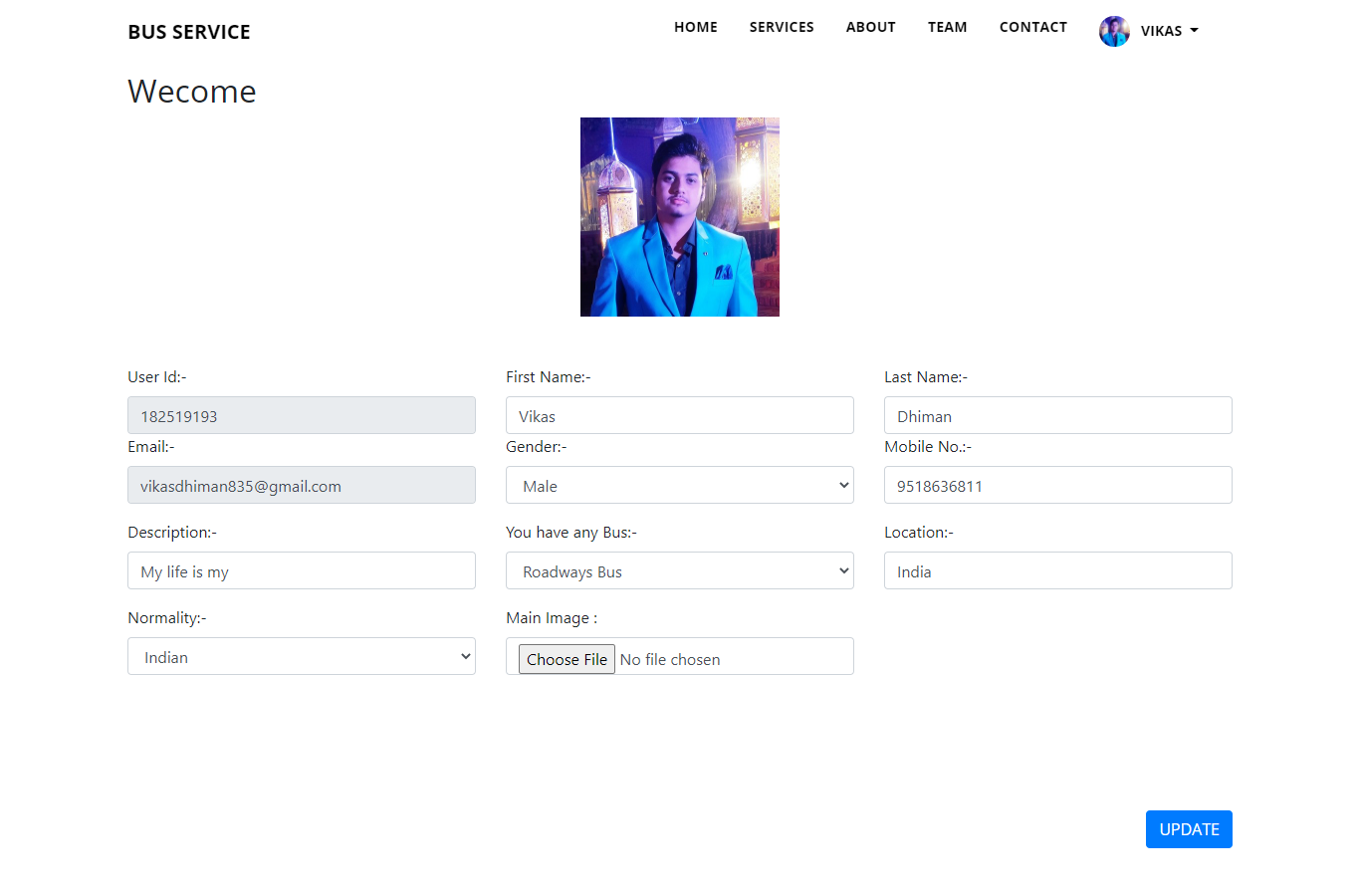
**Figure no: 6.1.7 (User Home Page)**

****

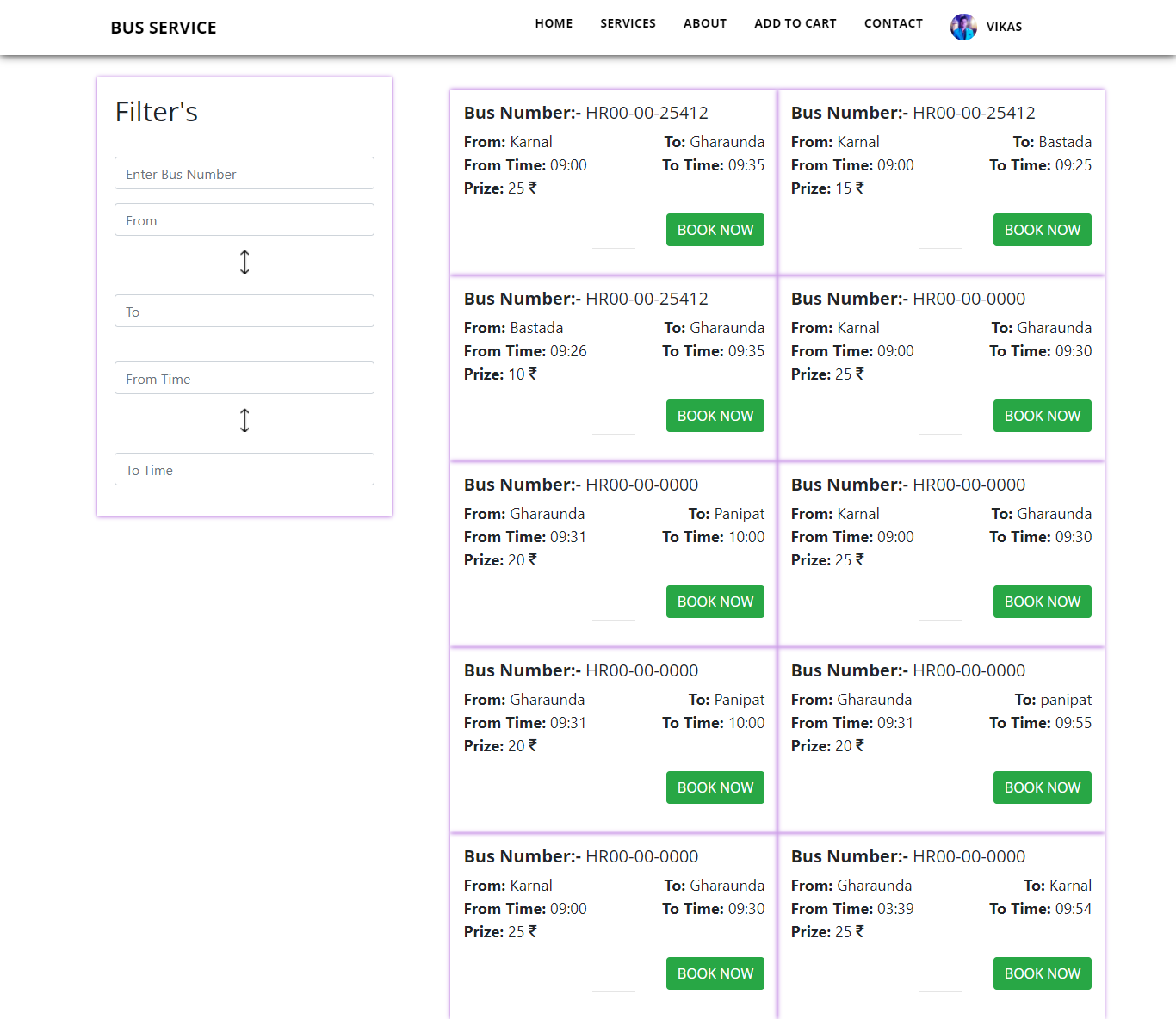
**Figure no: 6.1.8 (User Profile)**

****

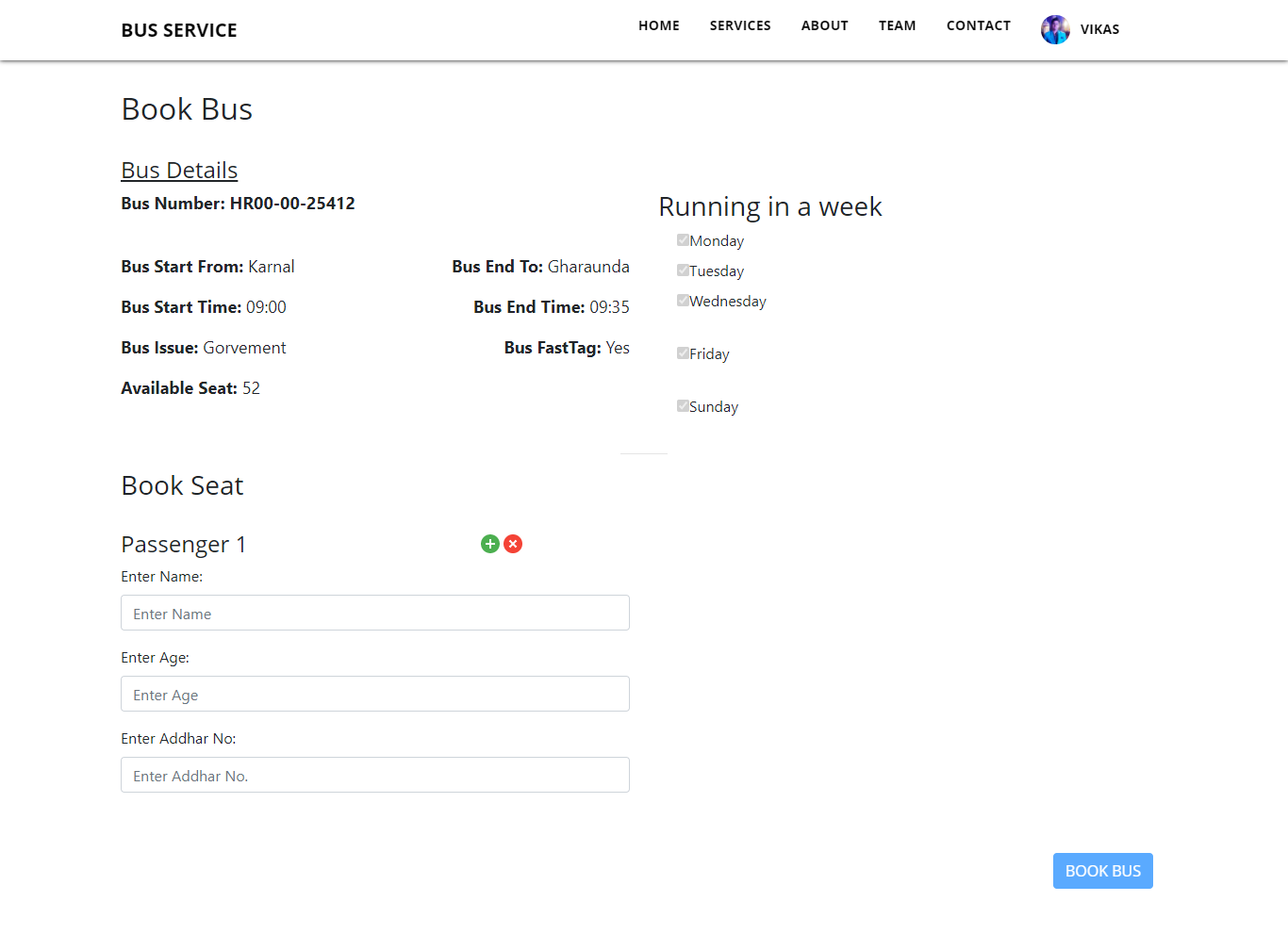
**Figure no: 6.1.9 (Add Address)**

****

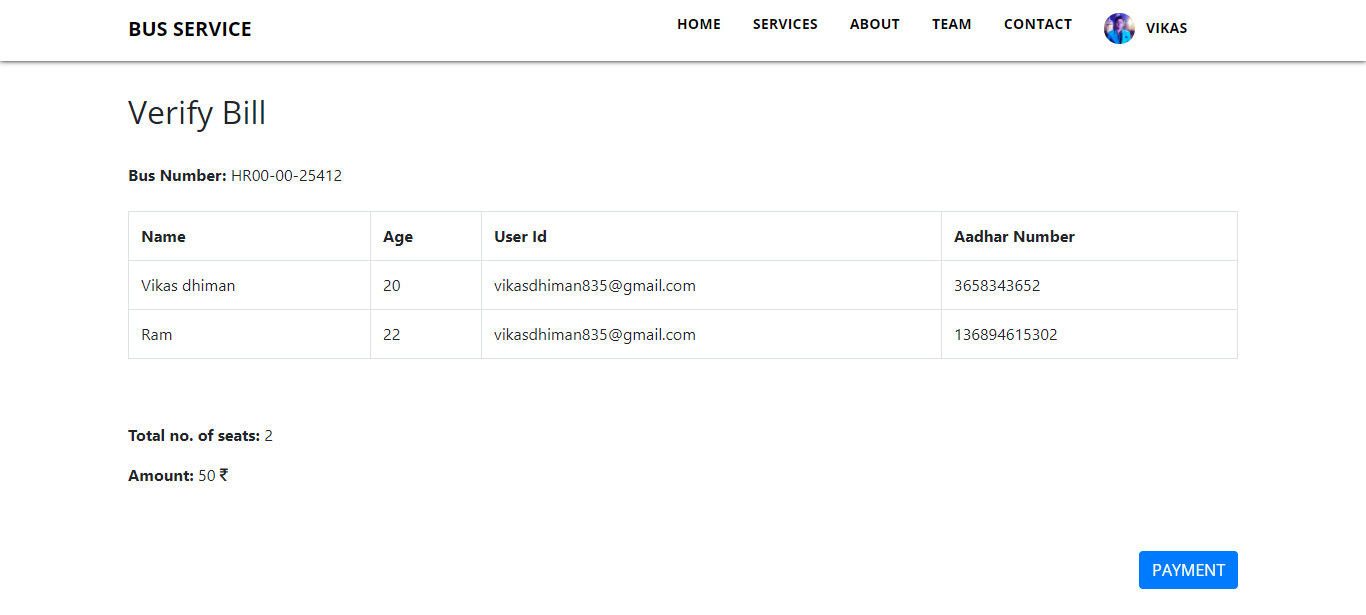
**Figure no: 6.1.10 (Edit Profile)**

****

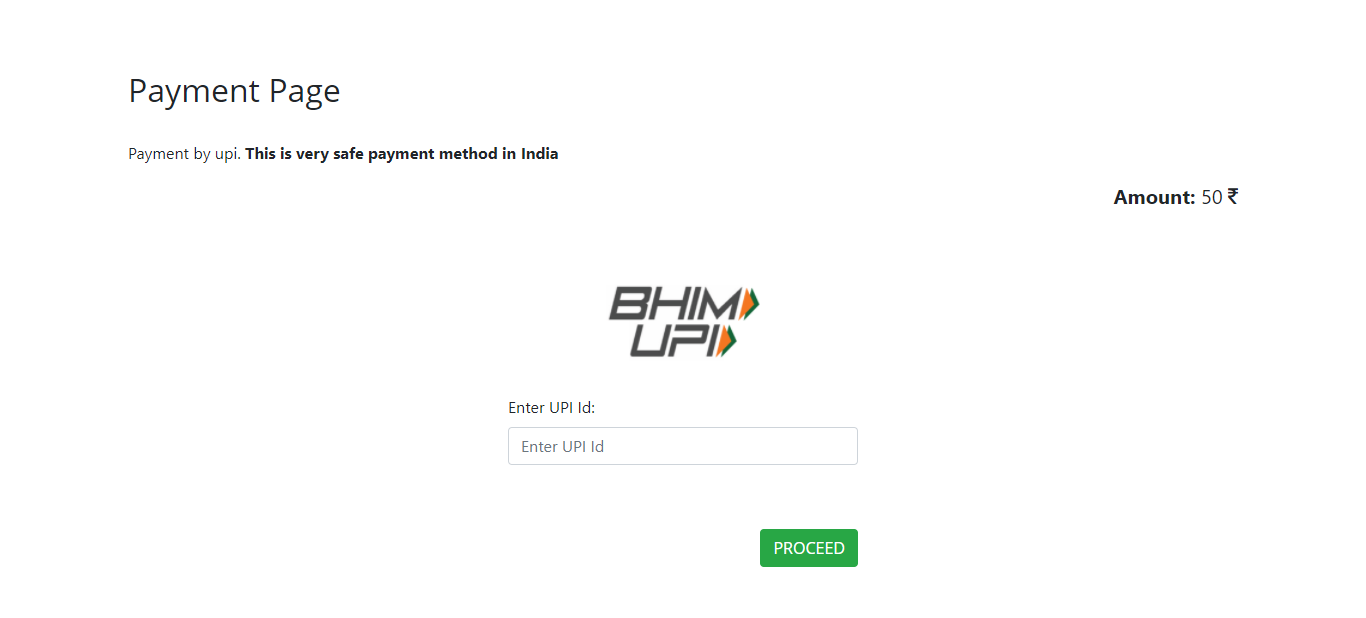
**Figure no: 6.1.11 (Check Bus)**

****

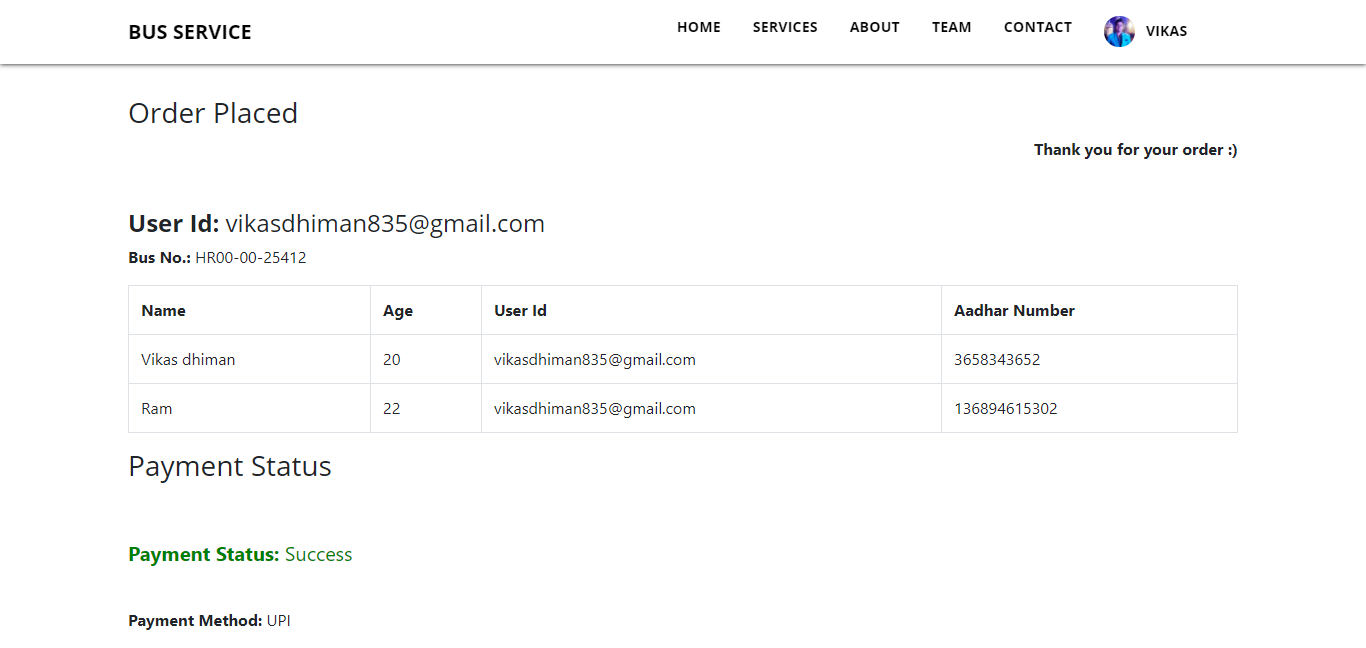
**Figure no: 6.1.12 (Book Bus)**

****

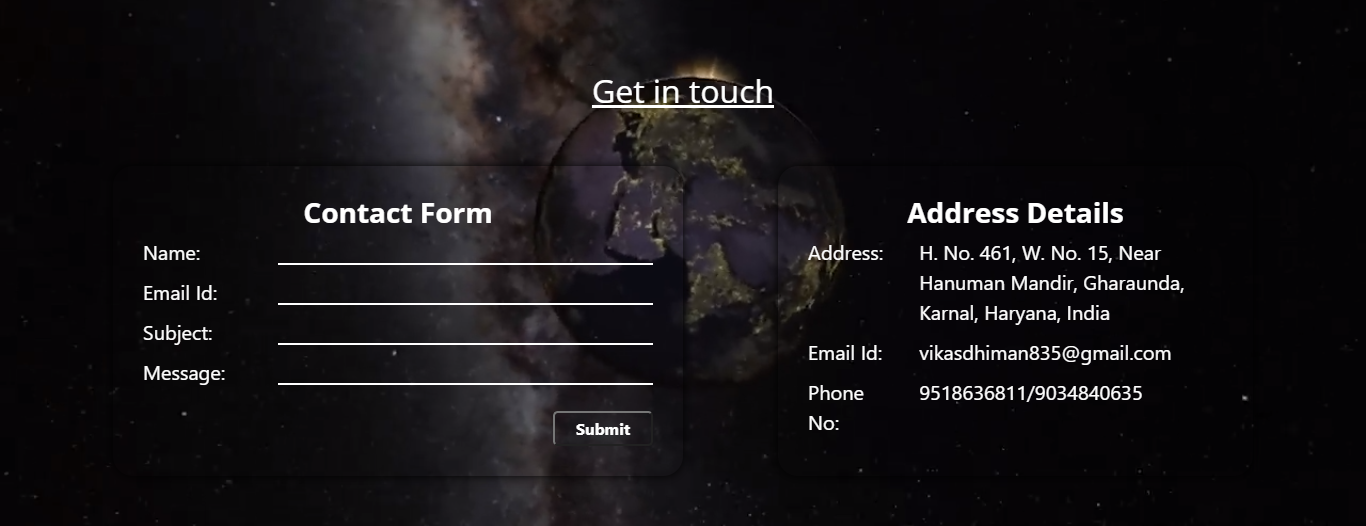
**Figure no: 6.1.13 (Verify Bill)**

****

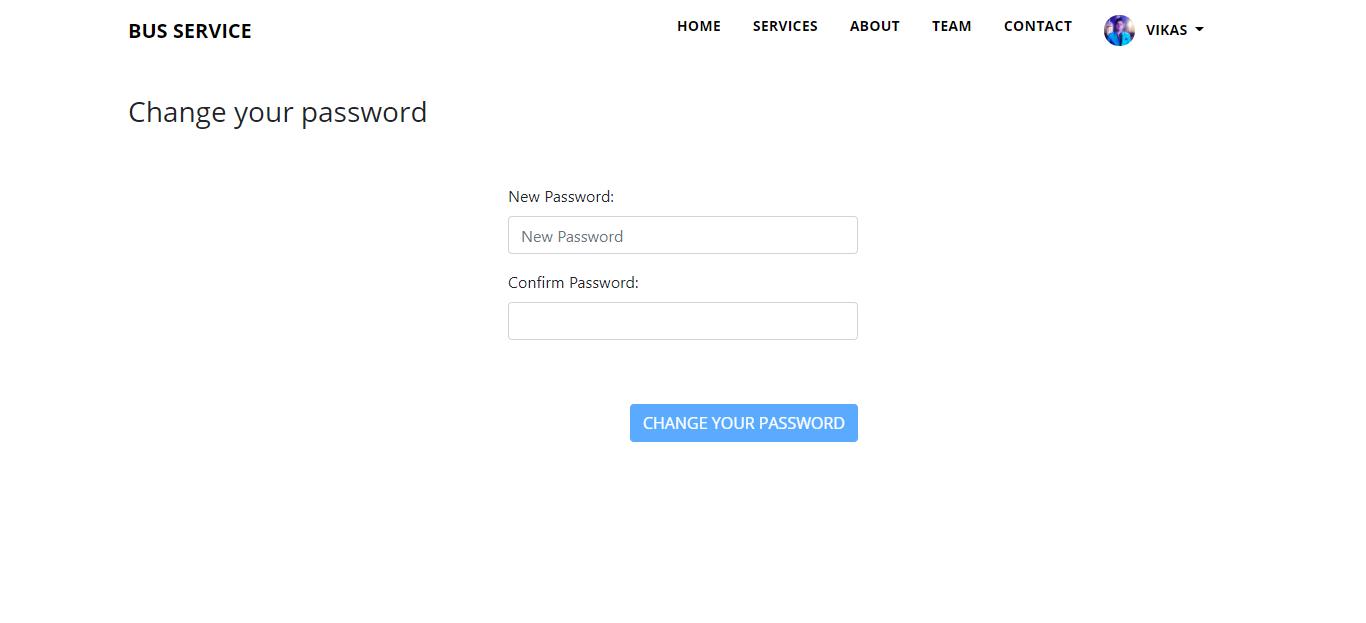
**Figure no: 6.1.14 (Payment Page)**

****

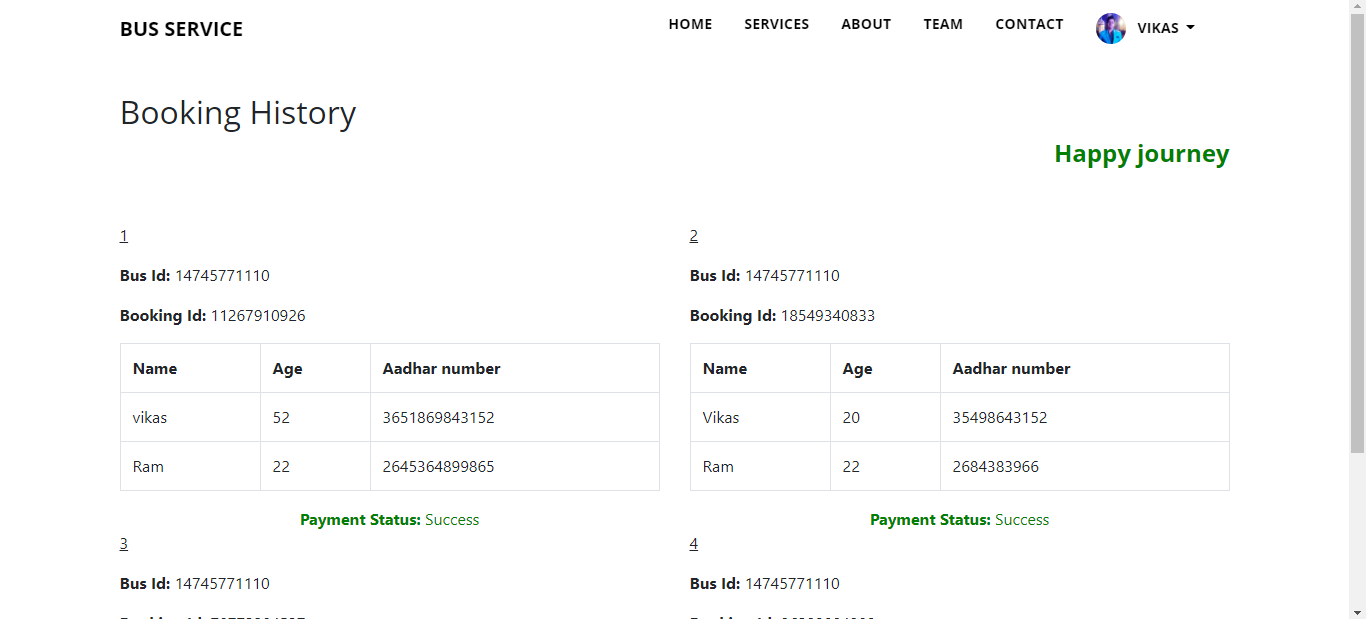
**Figure no: 6.1.15 (Order Placed)**

****

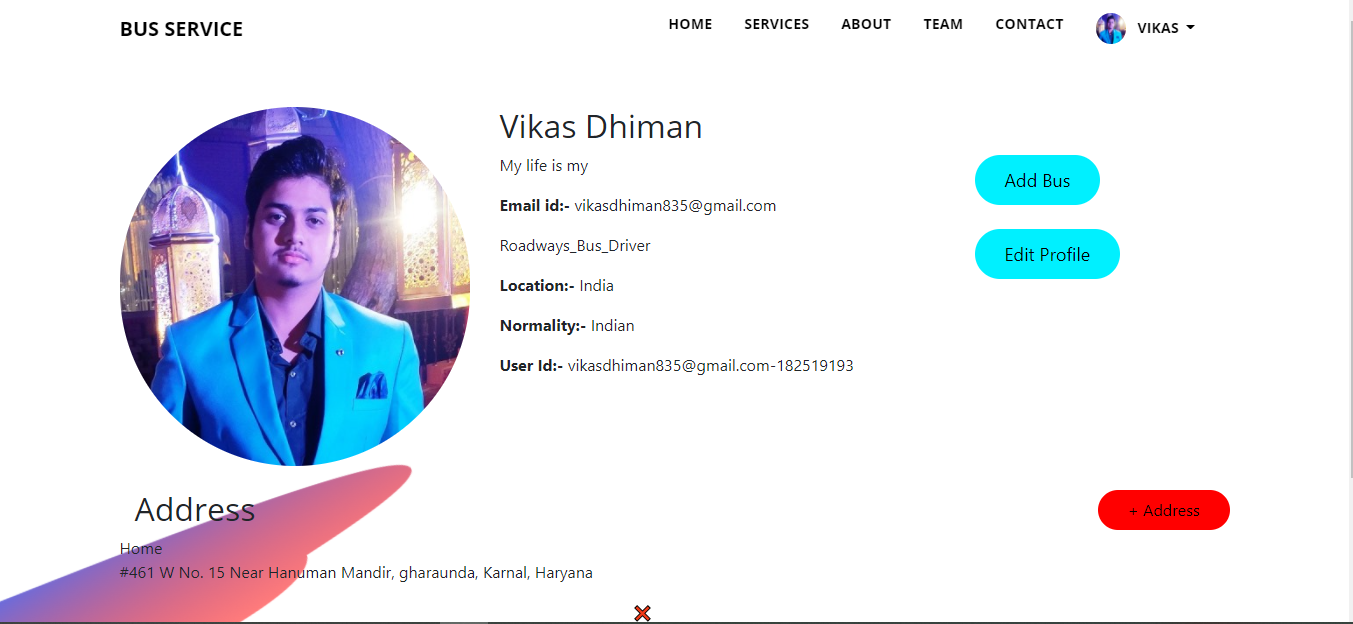
**Figure no: 6.1.16 (Contact Page)**

****

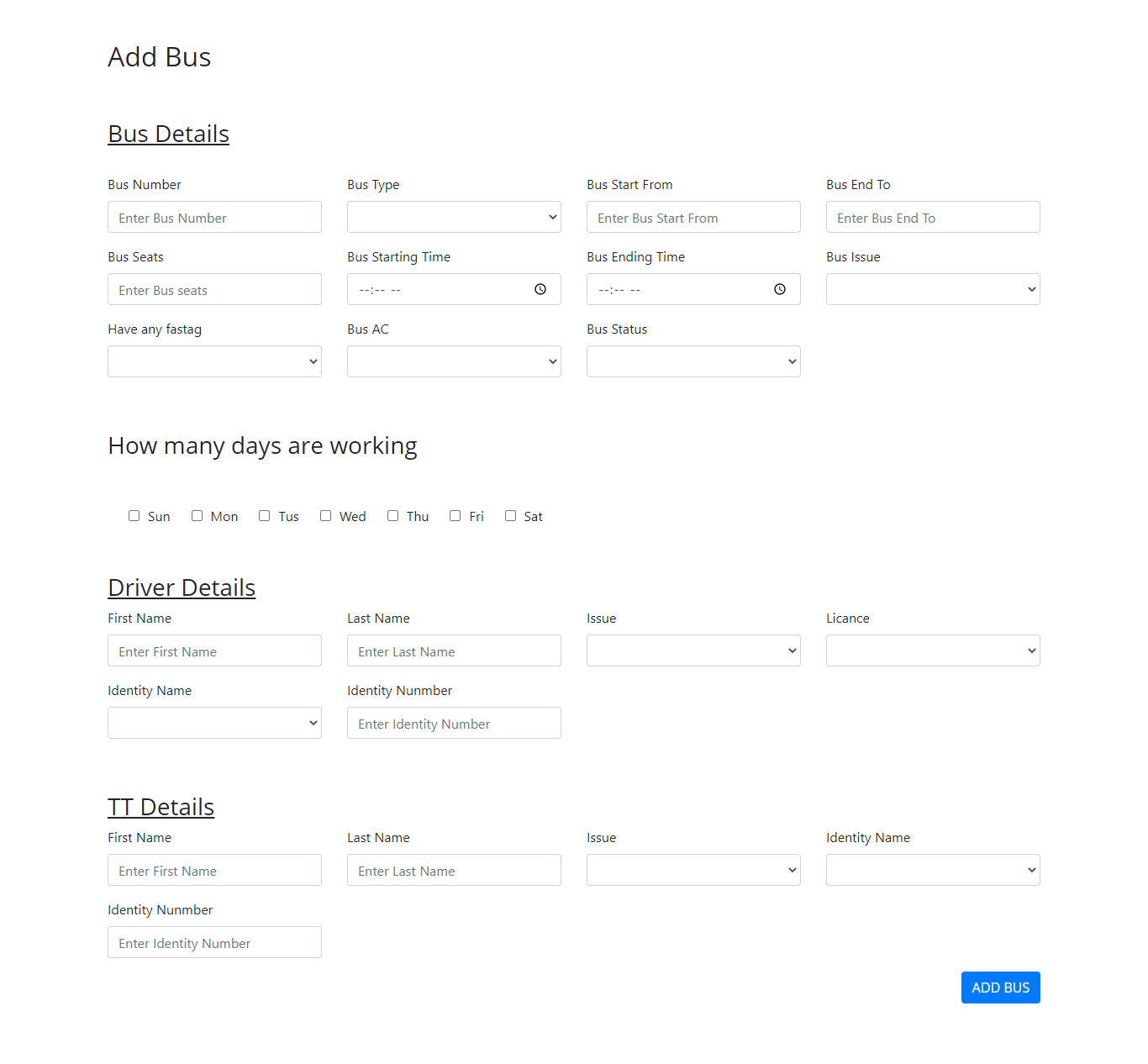
**Figure no: 6.1.17 (Change Password)**

****

**Figure no: 6.1.18 (Booking History)**

****

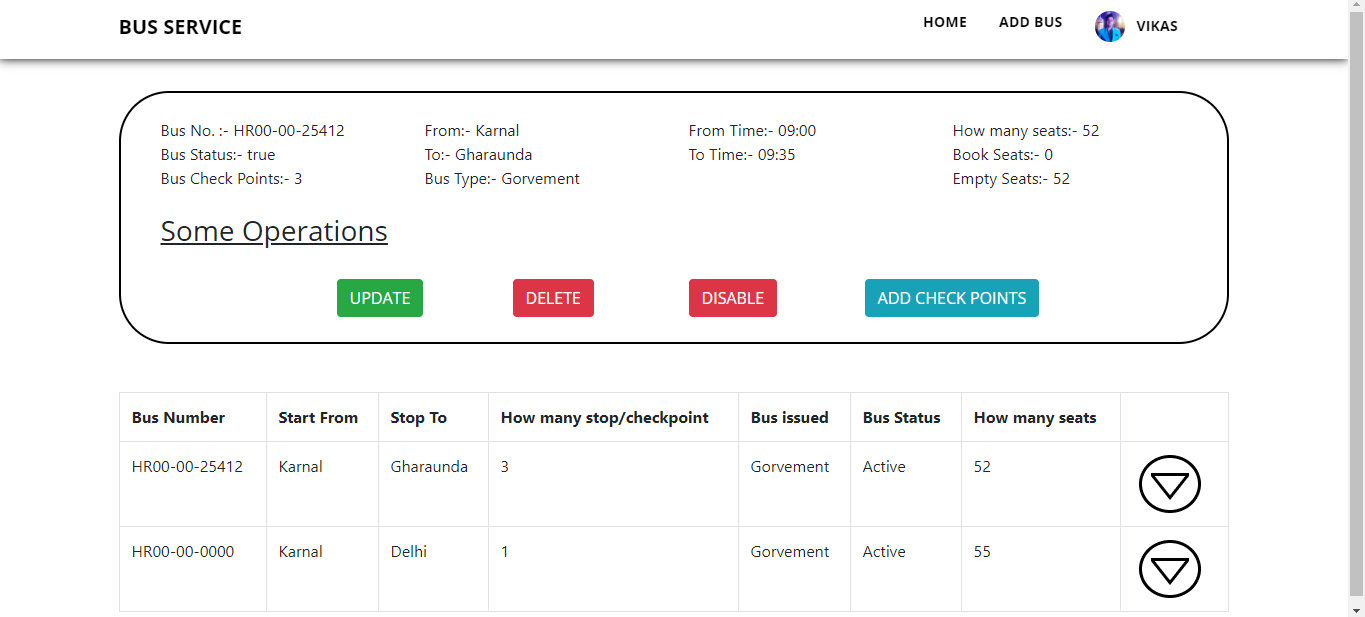
**Figure no: 6.1.19 (Bus Profile)**

****

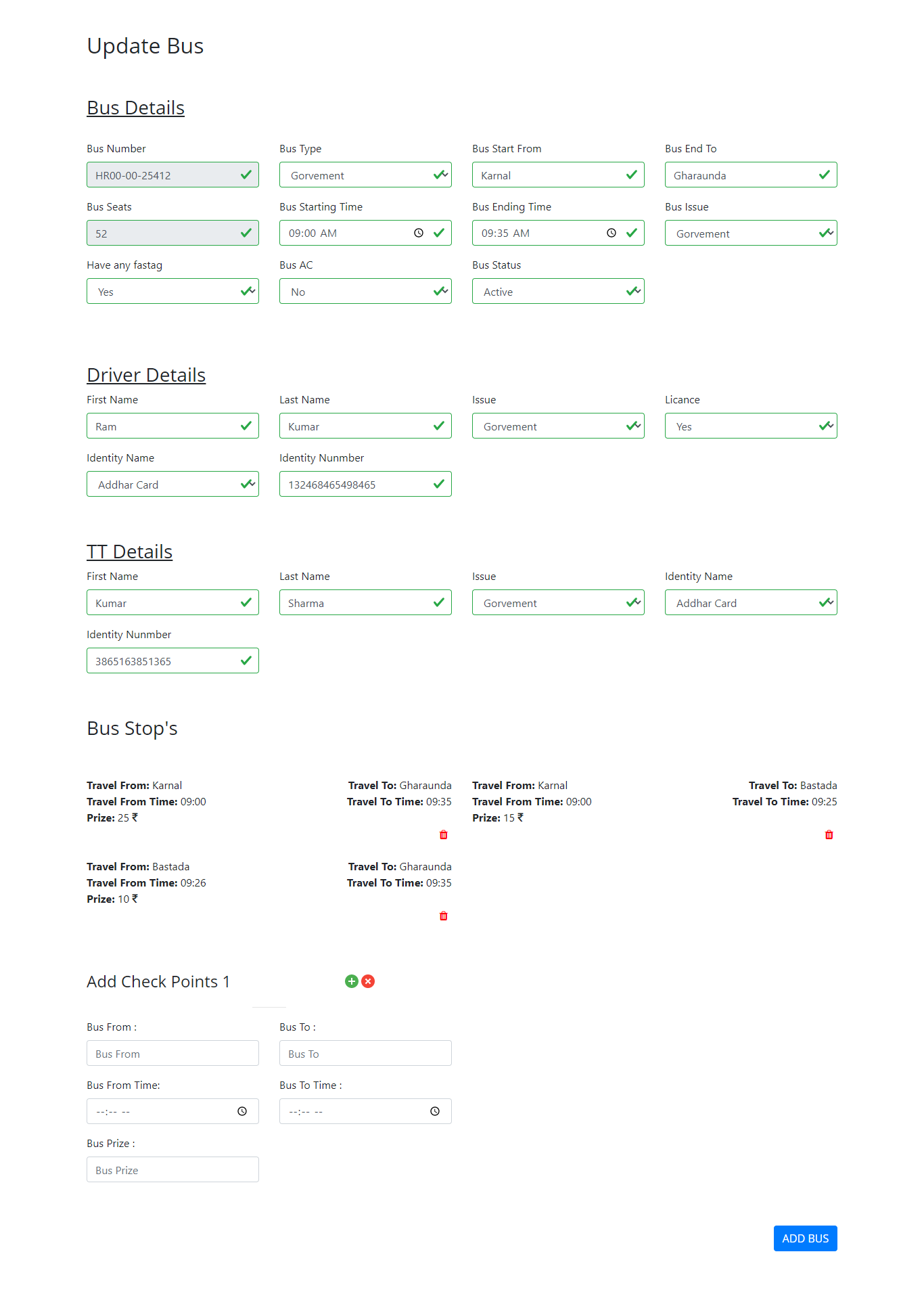
**Figure no: 6.1.20 (Add Bus)**

****

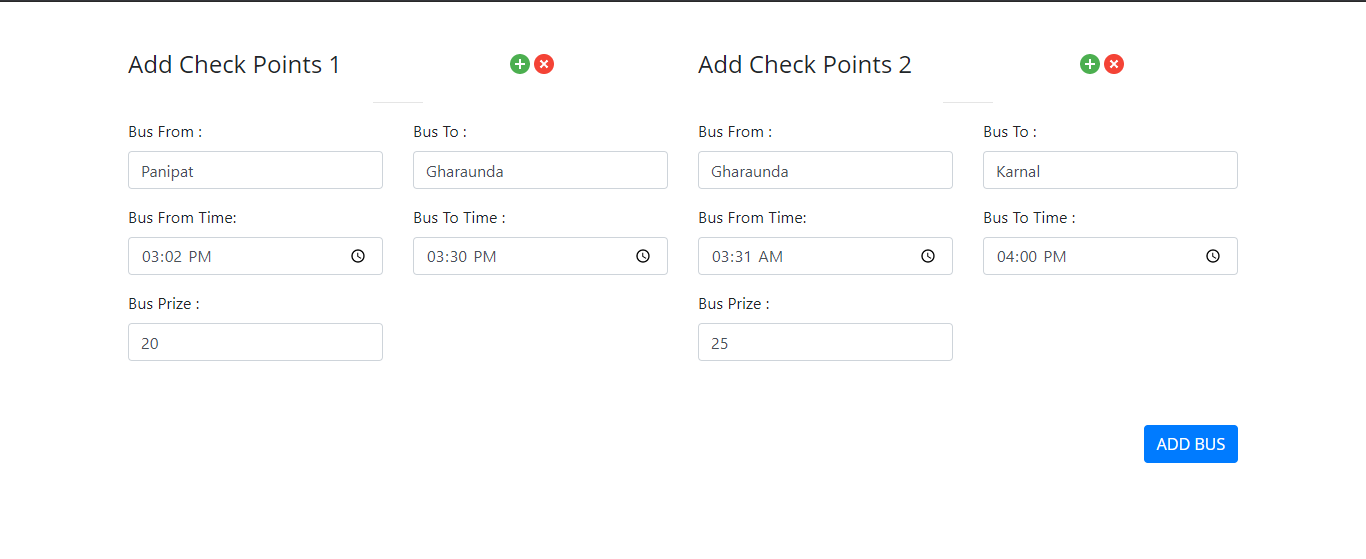
**Figure no: 6.1.21 (Bus List)**

****

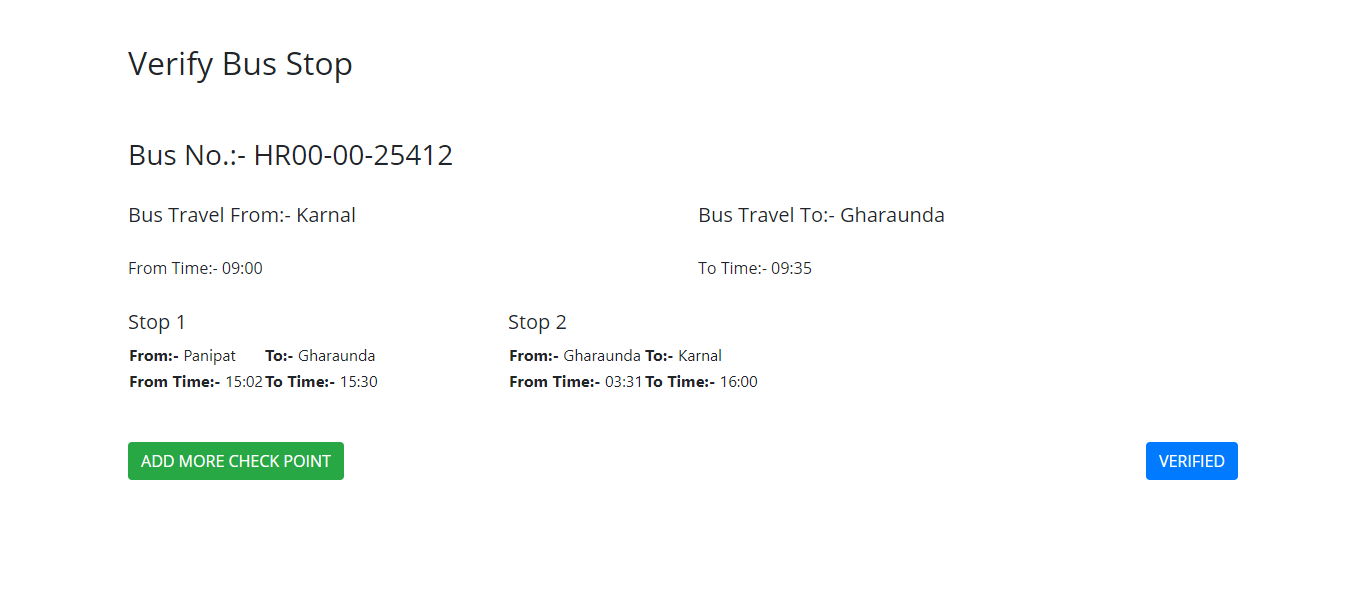
**Figure no: 6.1.22 (Show Main Bus)**

****

**Figure no: 6.1.23 (Update Bus)**

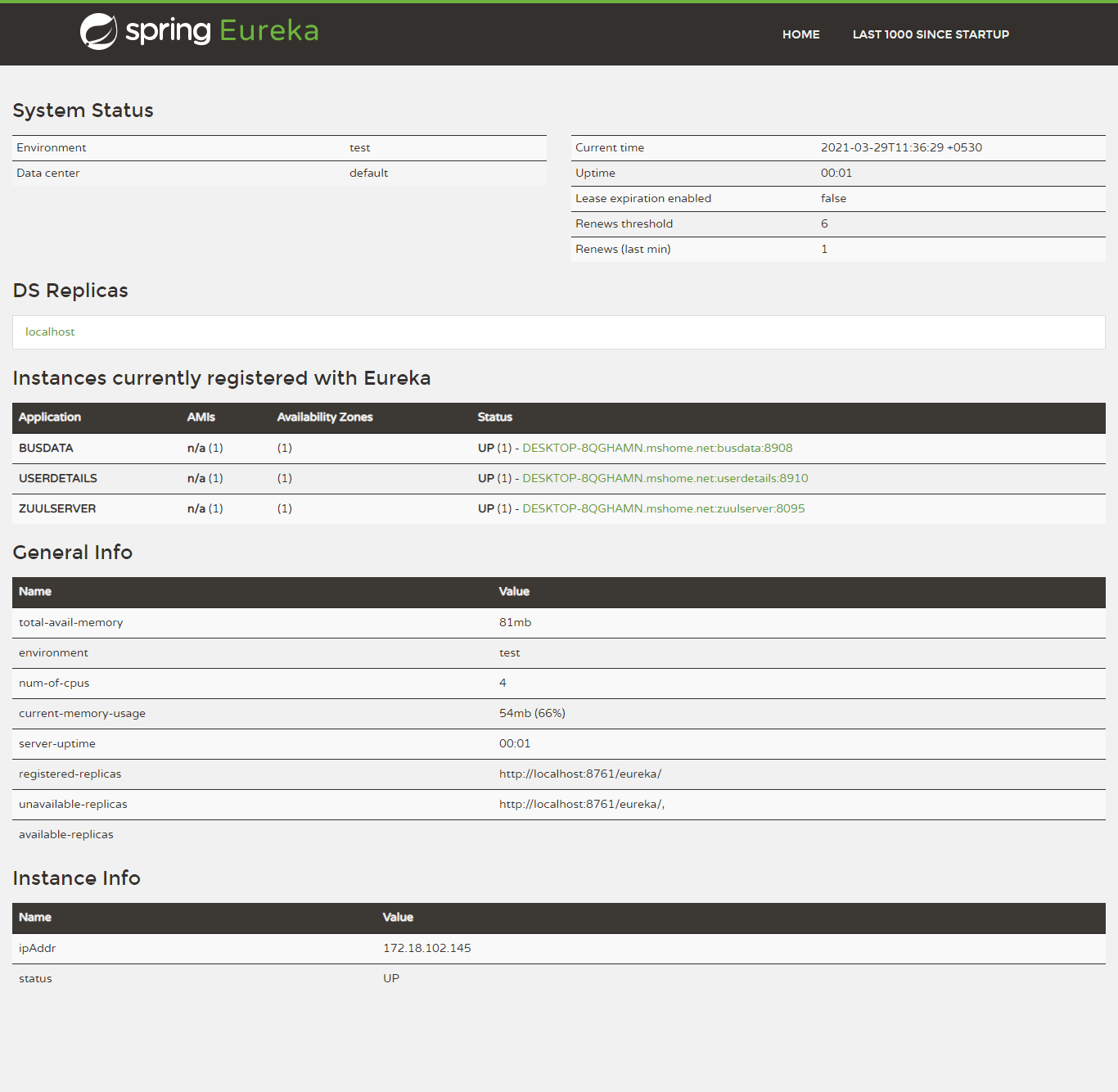
****

**Figure no: 6.1.24 (Add Check Points)**

****

**Figure no: 6.1.25 (Verify Bus Stops)**

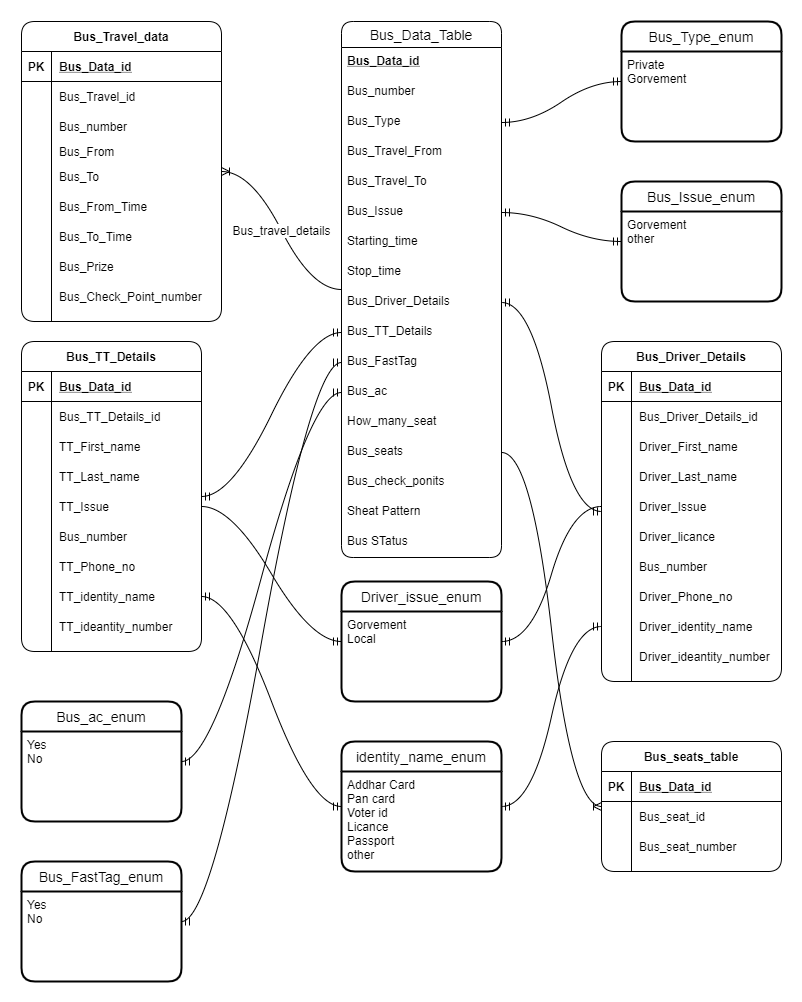
1. **Eureka Server Screen**

****

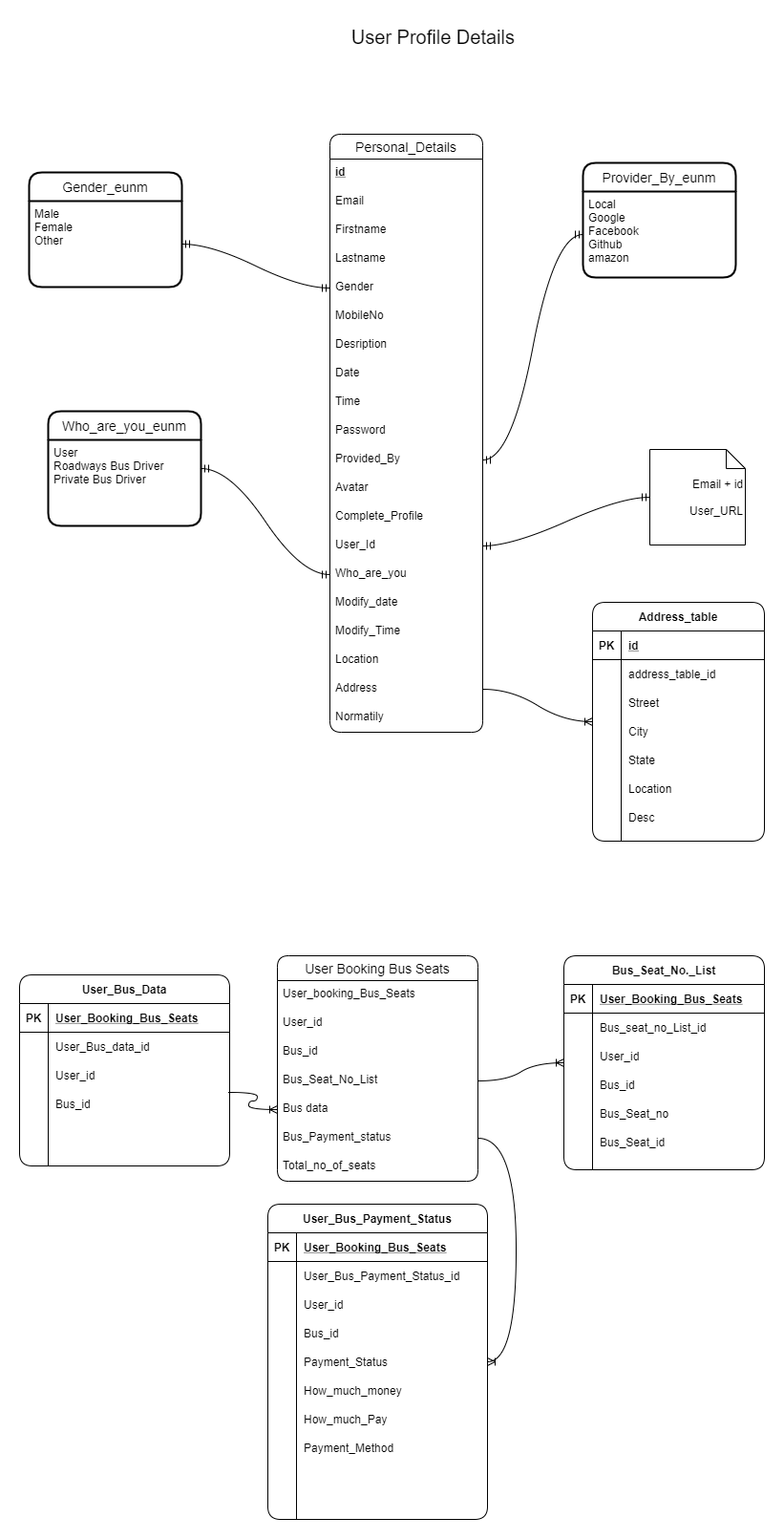
**Figure no: 6.2.1 (Eureka Server)**

1. **Database Output Screens**

* **E-R Diagram**

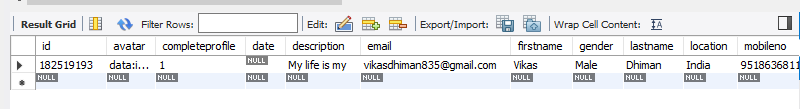


**Figure no: 6.4.1 (Bus Data Table Structure)**

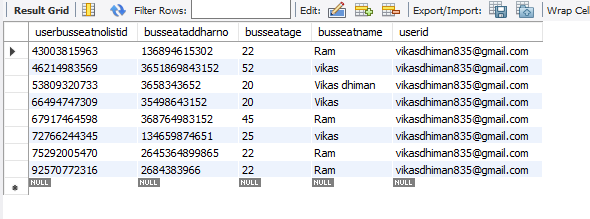


**Figure no: 6.4.2 (User Table Structure)**

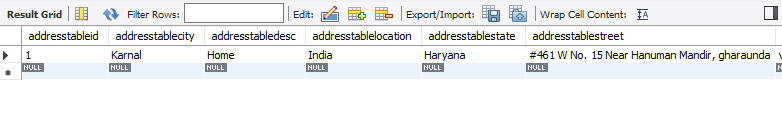
**Tables**

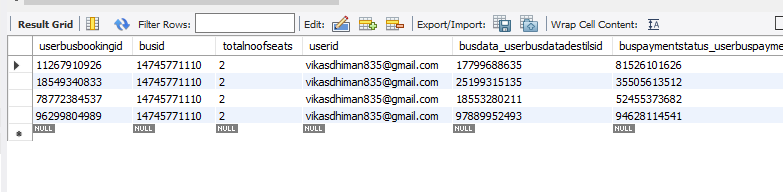
****

**Figure no: 6.3.1 (User Details Table)**

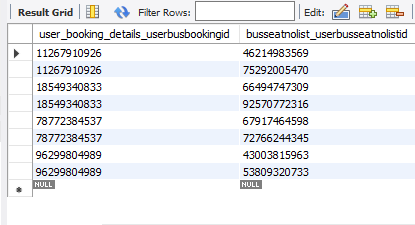
****

**Figure no: 6.3.2 (Book Seat Booking List)**

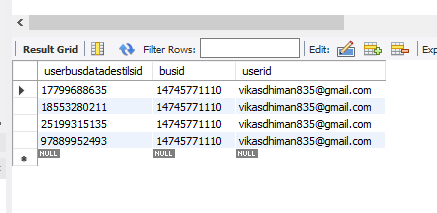
** Figure no: 6.3.3 (User Address)**

****

**Figure no: 6.3.4 (User Bookings)**

****

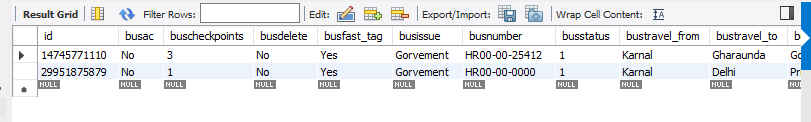
**Figure no: 6.3.5 (User Bookings lists no list)**

****

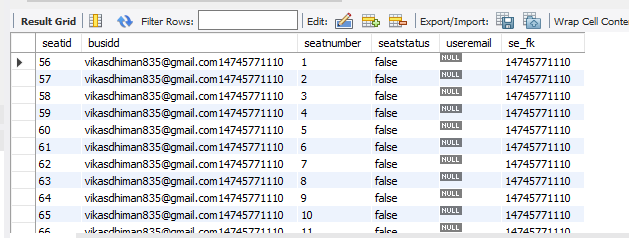
**Figure no: 6.3.6 (User Bus Data Details)**

****

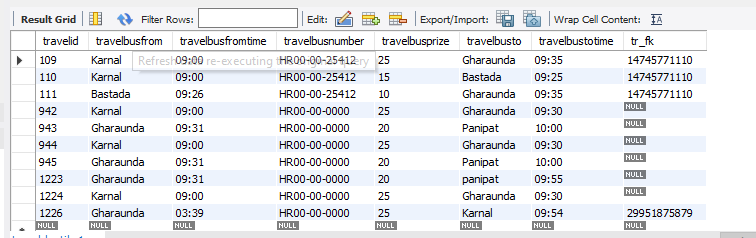
**Figure no: 6.3.7 (User Bus Payment Status)**

****

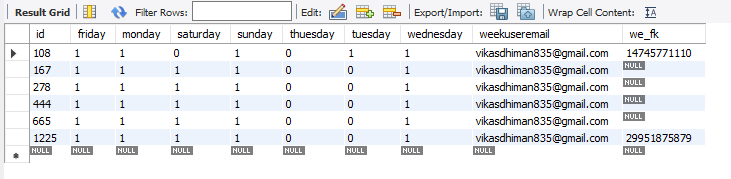
**Figure no: 6.3.8 (Bus Data)**

****

**Figure no: 6.3.9 (Bus Seats)**

****

**Figure no: 6.3.10 (Travel Details)**

****

**Figure no: 6.3.11 (Bus weeks)**

**Chapter -7**

**References**

**Chapter -7**

**References**

1. **Reference Website and Link**
2. <https://angular.io>[/](https://angular.io/)
3. <https://spring.io/projects/spring-boot>
4. <https://www.tutorialspoint.com/index.htm>
5. <https://www.javatpoint.com/>
6. <https://youtube.com/>
7. [https://www.tutorialspoint.com/spring\_boot/spring\_boot\_eureka\_server.htm#:~:text=Eureka%20Server%20is%20an%20application,also%20known%20as%20Discovery%20Server](https://www.tutorialspoint.com/spring_boot/spring_boot_eureka_server.htm)
8. <https://spring.io/microservices>
9. <https://dzone.com/articles/service-discovery-with-eureka-and-zuul>
10. <https://spring.io/guides/gs/service-registration-and-discovery/>