VISVESVARAYA TECHNOLOGICAL UNIVERSITY

BELAGAVI-590018

A Database Management System Mini Project Report

on

" Metro Management System "

Submitted in partial fulfillment of the requirements for the V semester

and award of the degree of Bachelor of Engineering in Computer Science

and Engineering of Visvesvaraya Technological University, Belagavi

Submitted by:

Sanath Kumar N	*1RN20CS128*
Sanket Krishna Hegde	*1RN20CS131*

Under the Guidance of:

Mrs. Soumya N G
Assistant Professor
Dept. of CSE

Department of Computer Science and Engineering
NBA Accredited upto June 2025
RNS Institute of Technology
Channasandra, Dr.Vishnuvardhan Road, Bengaluru-560 098
2022-2023

RNS Institute of Technology

Channasandra, Dr.Vishnuvardhan Road, Bengaluru-560098

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING

(NBA Accredited for academic years 2022-25)

CERTIFICATE

Certified that the mini project work entitled **“Metro Management System”**has been successfully carried out by **”Sanath Kumar N** bearing USN **”1RN20CS128”** and **“Sanket Krishna Hegde** bearing USN **”1RN20CS131”**, bonafide students of **”RNS Institute of Technology”** in partial fulfillment of the requirements for the 5th semester of **”Bachelor of Engineering in Computer Science and Engineering of Visvesvaraya Technological University”**, Belagavi, during academic year 2022-2023. It is certified that all corrections/suggestions indicated for Internal Assessment have been incorporated in the report deposited in the departmental library. The project report has been approved as it satisfies the DBMS laboratory requirements of 5th semester BE, CSE.

Signature of the Guide	Signature of the HOD	Signature of the Principal
Mrs. Soumya N G	**Dr. Kiran P**	**Dr. M K Venkatesha**
Assistant Professor	Professor, HOD CSE	Principal
Dept. of CSE	Dept. of CSE	

External Viva:

Name of the Examiners	Signature with Date
1.	
2.	

Acknowledgement

Any achievement, be it scholastic or otherwise does not depend solely on the individual efforts but on the guidance, encouragement and cooperation of intellectuals, elders and friends. A number of personalities, in their own capacities have helped us in carrying out this project work. We would like to take this opportunity to thank them all.

We are grateful to Management and **Dr. M K Venkatesha**, Principal, RNSIT, Bangalore, for his support towards completing this mini project.

We would like to thank **Dr. Kiran P**, Professor & Head, Department of Computer Science & Engineering, RNSIT, Bangalore, for his valuable suggestions and expert advice.

We deeply express our sincere gratitude to our guide **Mrs. Soumya N G** , Assistant Professor, Department of CSE, **Dr. Sudhamani M J**, Associate Professor, Department of CSE, RNSIT, Bangalore, for their able guidance, regular source of encouragement and assistance throughout this project.

We would like to thank all the teaching and non-teaching staff of Department of Computer Science & Engineering, RNSIT, Bengaluru-98 for their constant support and encouragement.

Abstract

A metro management system is a software that enables the creation and issuance of electronic tickets for a metro system. This system would allow passengers to purchase tickets via various channels, such as desktop, through a mobile phone, or at physical ticket vending machines located at metro stations all via online medium. The system would be integrated with the metro management database, allowing it to access real-time information about train schedules and fare prices.

The ticket generation system would use a variety of technologies, such as QR code, and RFID, to generate and validate tickets. Passengers would receive the electronic ticket printed from a vending machine or online. They would then need to scan the ticket at the turnstile to enter the station and at the destination station to exit. The system would also have the capability to issue refunds or resolve disputes if necessary trough consumer support.

The implementation of a metro management system would bring many benefits to the metro system, such as reducing the need for paper tickets and cash transactions, increasing the speed and efficiency of the ticketing process, and providing valuable data and insights on passenger behavior and fare collection. This system would also allow for the integration of fare discounts and promotions, as well as the possibility of using the same ticket for other forms of public transportation.

Contents

List of Figures

Chapter 1

Introduction

1.1 Database Technologies

The essential feature of database technology is that it provides an internal representation (model) of the external world of interest. Examples are, the representation of a particular date/time/flight/aircraft in an airline reservation or of the item code/item description/quantity on hand/reorder level/reorder quantity in a stock control system.

The technology involved is concerned primarily with maintaining the internal representation consistent with external reality; this involves the results of extensive RD over the past 30 years in areas such as user requirements analysis, data modelling, process modelling, data integrity, concurrency, transactions, file organisation, indexing, rollback and recovery, persistent programming, object-orientation, logic programming, deductive database systems, active database systems... and in all these (and other) areas there remains much more to be done. The essential point is that database technology is a CORE TECHNOLOGY which has links to:

- Information management / processing
- Data analysis / statistics
- Data visualization / presentation
- Multimedia and hypermedia
- Office and document systems
- Business processes, workflow, CSCW (computer-supported cooperative work)

Relational DBMS is the modern base technology for many business applications. It offers flexibility and easy-to-use tools at the expense of ultimate performance. More recently relational systems have started extending their facilities in directions like information retrieval, objectorientation and deductive/active systems which lead to the so-called 'Extended Relational Systems'.

Information Retrieval Systems began with handling library catalogues and then extended to full free-text by utilizing inverted index technology with a lexicon or thesaurus. Modern systems utilize some KBS (knowledge-based systems) techniques to improve the retrieval. Object-Oriented DBMS started for engineering applications in which objects are complex, have versions and need to be treated as a complete entity. OODBMSs share many of the OOPL features such as identity, inheritance, late binding, overloading and overriding. OODBMSs have found favours in engineering and office systems but haven't been successful yet in traditional application areas.

Deductive / Active DBMS has evolved over the last 20 years and combines logic programming technology with database technology. This allows the database itself to react to the external events and also to maintain its integrity dynamically with respect to the real world.

1.2 Characteristics of Database Approach

Traditional form included organising the data in file format. DBMS was a new concept then, and all kinds of research was done to make it overcome the deficiencies in traditional style of data management. A modern DBMS has the following characteristics

- Real-world entity A modern DBMS is more realistic and uses real-world entities to design its architecture. It uses behaviour and attribute too. For example, a school database may use students as an entity and their age as an attribute.
- Relation-based tables DBMS allows entities and relations to form tables. A user can understand the architecture of a database by just looking at the table names.
- Isolation of data and application A database system is entirely different than its data. A database is an active entity, whereas data is said to be passive, on which the database works and organizes. DBMS also stores metadata, which is data about data, to ease its own process.
- Less redundancy DBMS follows the rules of normalization, which splits a relation when any of its attributes has redundancy in its values. Normalization is a mathematically rich and scientific process that will reduces the data redundancy.
- Consistency Consistency is a state where every relation in a database remains consistent. There exists methods and techniques, that can detect an attempt of leaving database in an inconsistent

state. DBMS can provide greater consistency as compared to earlier forms of data storing applications like file-processing systems.

- Query Language DBMS is equipped with query language, which makes it more efficient to retrieve and manipulate data. A user can apply as many and the filtering options as required to retrieve a set of data. Traditionally it was not possible where file-processing system was used.
- ACID Properties DBMS follows the concepts of Atomicity, Consistency, Isolation, and Durability (normally shortened as ACID). These concepts are applied on transactions, which manipulate data in a database. ACID properties help the database to stay healthy in multi-transactional environments and also in case of failure.
- Multiuser and Concurrent Access DBMS supports multi-user environment and allows them to access and manipulate data in parallel. Though there are restrictions on transactions when users attempt to handle the same data item, but users are always unaware of them.
- Multiple views DBMS offers multiple views for different users. A user in the Sales department will have a different view of the database from the person working in the Production department. This feature enables the users to have a concentrate view of the database according to their requirements.
- Security Features like multiple views offer security to certain extent when users are unable to access the data of other users and departments. DBMS offers methods to impose constraints while entering data into the database and retrieving the same at a later stage. DBMS offers many different levels of security features, which enables multiple users to have different views with different features. For example, a user in the Sales department cannot see the data that belongs to the Purchase department. It can also be helpful in deciding how much data of the Sales department should be displayed to the user. Since a DBMS is not saved on the disk as traditional file systems, it is very hard for miscreants to break the code.

1.3 Applications of DBMS

Applications of Database Management Systems :

- Telecom: There is a database to keeps track of the information regarding the calls made, network usage, customer details etc. Without the database system it is hard to maintain such huge amounts of data which gets updated every millisecond.

- Industry: Whether it is a manufacturing unit, a warehouse or a distribution centre, each one needs a database to keep the records of the ins and outs. For example, a distribution centre should keep a track of the product units that were supplied to the centre as well as the products that got delivered from the distribution centre on each day; this is where DBMS comes into picture.
- Banking System: For storing information regarding a customer, keeping a track of his/her day to day credit and debit transactions, generating bank statements etc is done with through Database management systems.
- Education sector: Database systems are frequently used in schools and colleges to store and retrieve the data regarding the student , staff details, course details, exam details, payroll data, attendance details, fees details etc. There is lots of inter-related data that needs to be stored and retrieved in an efficient manner.
- Online shopping: You must be aware of the online shopping websites such as Amazon, Flip kart etc. These sites store the product information, your addresses and preferences, credit details and provide you the relevant list of products based on your query. All this involves a Database management system.

1.4 Problem Description/Statement

This is a service based project which provide all information about the metro rail and along with a functionality of ticket generation for public. The proposed system is a web based application which provides information regarding timings, routes, fair. This system manages public feedback about services through it's complaint management system. This system also containsa feature to provide feedback where the customers can provide feedback to help us enhance our service.

There is also an admin module where admin can add stations, trains, routes and also update the fairs.

Chapter 2

Requirement Analysis

2.1 Hardware Analysis

The Hardware requirements are very minimal and the program can be run on most of the machines.

Processor : i5 processor

Processor Speed : 1.2 GHz

RAM : 1 GB

Storage Space : 40 GB

Monitor Resolution : 1024*768 or 1336*768 or 1280*1024

2.2 Software Requirements

System specifications and software required are,

Operating System used: Windows 10

Technologies used: HTML, CSS, PHP, Django, Bootstrap

Server: MySQL, DjangoMyAdmin

IDE used: Visual Studio Code, Jupyter

Browser that supports HTML

2.3 End User Requirements

The technical requirements for the project are mentioned below:

1.Django 2.MySQL 3.HTML 4.CSS

2.3.1 Django

Django is a high-level Python web framework that enables the rapid development of secure and maintainable websites and web applications. It follows the Model-View-Controller (MVC) architectural pattern and is built on top of Python's standard library, making it easy to use and understand for developers familiar with Python.

One of the key features of Django is its built-in support for models, which represent the data structures of a web application. Models are defined using Python classes, and they can be easily created, modified, and deleted using Django's Object-Relational Mapping (ORM) system. This allows developers to focus on the logic of their application, rather than writing complex SQL queries to interact with the database.

Another important aspect of Django is its views, which handle the logic of a web application. Views are Python functions that handle incoming HTTP requests and return appropriate HTTP responses. They can be used to perform actions such as fetching data from the database, processing form submissions, and rendering templates. Django's built-in URL dispatcher makes it easy to map URLs to specific views, allowing developers to define the routing of their application.

Django also includes a powerful template engine that allows developers to separate the presentation logic of their application from the business logic. Templates are written using a simple language and can include variables, loops, and conditional statements, making it easy to create dynamic, reusable HTML templates.

Additionally, Django provides a built-in administration interface that allows developers to easily manage the data of their application. This feature is particularly useful for creating applications that need to be maintained by non-technical users. The interface can be customized and extended, and allows the management of users, groups, permissions and more.

Django also provides built-in support for security features such as Cross-Site Request Forgery (CSRF) protection and SQL injection prevention. This helps developers to create secure web applications without having to spend a lot of time and effort on implementing these features themselves.

Overall, Django is a powerful and flexible web framework that makes it easy to create and maintain websites and web applications. Its built-in support for models, views, templates, and security features, along with its focus on maintainability and reusability, make it a great choice for developers looking to build robust and scalable web applications

2.3.2 MySQL

MySQL is a popular open-source relational database management system (RDBMS) that is widely used for web applications and data warehousing. It is developed, distributed, and supported by Oracle Corporation. MySQL uses the SQL (Structured Query Language) to manage and manipulate the data stored in its databases.

MySQL is known for its speed, reliability, and ease of use. It is also highly customizable and can be easily integrated with other software and programming languages such as PHP, Python, and Java. MySQL supports various storage engines, which allow users to choose the storage method that best suits their needs. Some of the most commonly used storage engines are InnoDB, MyISAM, and Memory.

MySQL is also highly scalable and can handle large amounts of data and concurrent connections. This makes it well suited for use in high-traffic websites, online transaction processing (OLTP) systems, and data warehousing. MySQL also supports advanced features such as replication and partitioning, which help users to improve the performance and availability of their databases.

Additionally, MySQL provides a wide range of tools for managing and maintaining databases, including the MySQL Command Line Client, the MySQL Workbench, and the MySQL Administrator. These tools enable users to create and modify databases, manage users and permissions, and perform backups and restores.

Overall, MySQL is a powerful and widely used RDBMS that is well suited for a wide range of applications. Its open-source nature, scalability, and support for advanced features make it a popular choice among developers and system administrators.

2.3.3 HTML and CSS

HTML (Hypertext Markup Language) and CSS (Cascading Style Sheets) are two essential technologies used for creating and designing websites. HTML is used to structure and organize the content of a website, while CSS is used to control the layout and appearance of that content.

HTML is a markup language that uses tags to define the structure of a web page. These tags include headings, paragraphs, lists, links, images, and more. By using these tags, developers can create a hierarchical structure for their content, making it easy for both humans and machines to understand the organization of the page.

CSS is used to apply styles to the HTML elements on a web page. Styles include things like color, font, size, spacing, and positioning. With CSS, developers can create a consistent look and feel across all pages of a website, and can also easily make changes to the layout and design without having to alter the HTML code.

CSS also allows for the separation of presentation and content, which makes it easier to maintain and update a website over time. It also enables responsive design, which allows the layout and design of a website to adapt to the size of the device or screen on which it is viewed. This is becoming increasingly important as more and more people access the internet on mobile devices.

CSS also has some advanced features like CSS Grid and Flexbox which enables the developers to create complex and responsive layouts easily. With the help of these features, developers can create grid-based layouts and align elements on a web page in a flexible and responsive way.

In conclusion, HTML and CSS are essential building blocks of the modern web. HTML provides the structure and organization of a web page, while CSS provides the layout and design. Together, they make it possible to create beautiful and functional websites that are easy to navigate and maintain.

Chapter 3

Design Methodology

3.1 Entities, Attributes and Relations

- Metro User

  User name - primary key

  Password

  Mobile number

  Email

- Station

  Station name - primary key

  Station ID

- Feedback

  Name - foreign key references to User name (Metro User)

  Review Stars

  Message

- Help

  Name - foreign key references to User name (Metro User)

  Email

  Contact

  Message

- Ticket Details

  Ticket ID - primary key

  Source ID - foreign key reference to Station ID (Station)

  Destination ID - foreign key references to Station ID (Station)

  User name - foreign key references to User name (Metro User)

  Date

  Trip

  Cost

  Time slot

3.2 ER Diagram

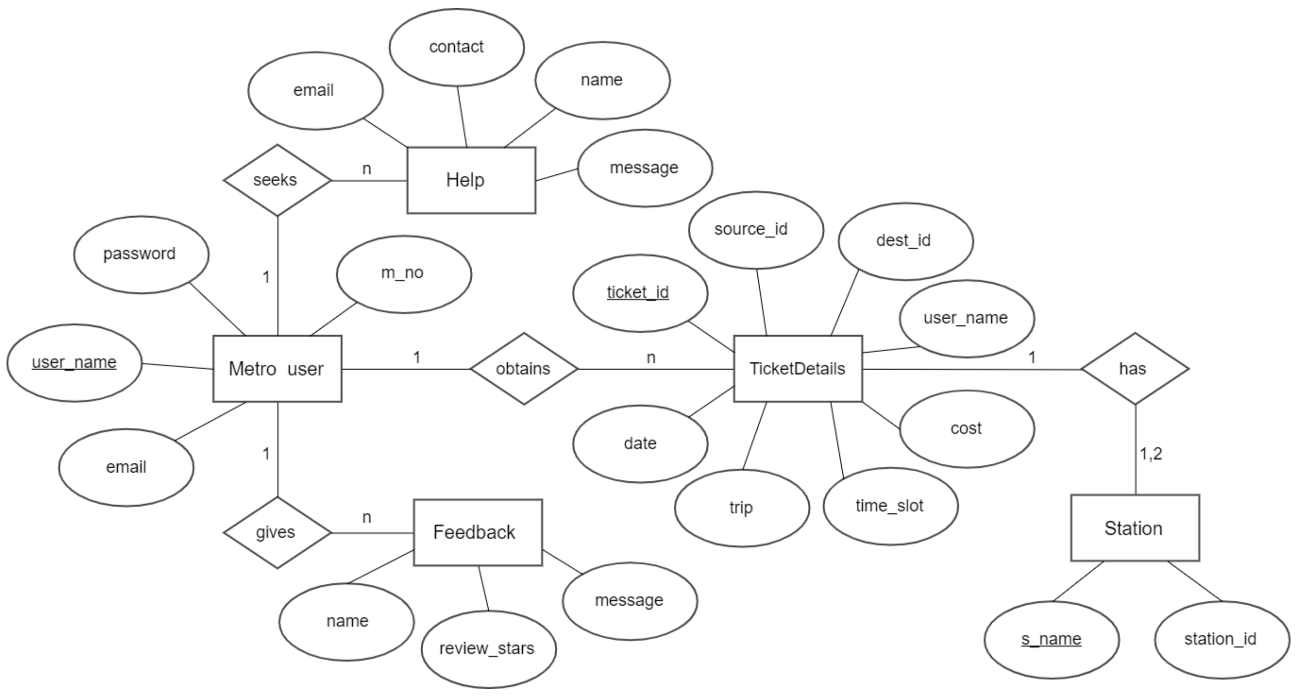


Figure 3.1: ER Diagram

3.3 Schema Diagram

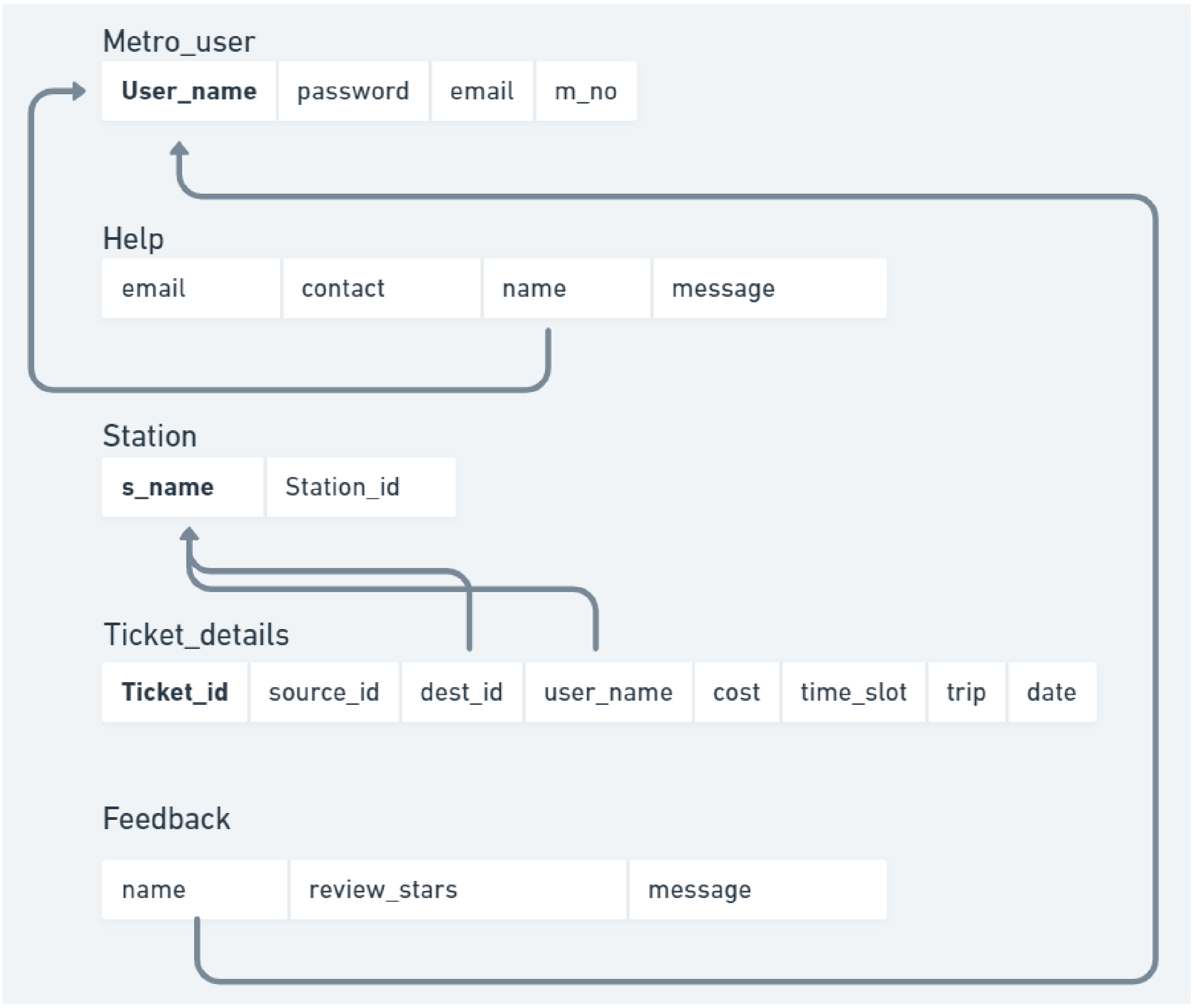


Figure 3.2: Schema Diagram

Chapter 4

Implementation

4.1 Database Connectivity

```
class Feedback(models.Model):
    email = models.CharField(max_length=20, blank=True, null=True)
    cno = models.BigIntegerField(primary_key=True)
    uname = models.ForeignKey('Metrouser', models.DO_NOTHING, db_column='uname',
    blank=True, null=True)
    detail_id = models.IntegerField(blank=True, null=True)
    class Meta:
        managed = False
        db_table = 'feedback'

class Metrouser(models.Model):
    username = models.CharField(primary_key=True, max_length=20)
    password = models.CharField(max_length=20, blank=True, null=True)
    mno = models.BigIntegerField(blank=True, null=True)
    email = models.CharField(max_length=30, blank=True, null=True)
    class Meta:
        managed = False
        db_table = 'Metrouser'

class Station(models.Model):
    s_name = models.CharField(max_length=20, blank=True, null=True)
    station_id = models.CharField(primary_key=True, max_length=20)
    class Meta:
        managed = False
        db_table = 'station'

class Ticketdetails(models.Model):
    ticket_id = models.CharField(primary_key=True, max_length=20)
    sourceid = models.ForeignKey(Station,models.DO_NOTHING,db_column='sourceid',
    blank=True, null=True,related_name='src')
    destid = models.ForeignKey(Station, models.DO_NOTHING, db_column='destid',
    blank=True, null=True,related_name='dest')
    uname = models.ForeignKey(Metrouser, models.DO_NOTHING, db_column='uname',
    blank=True, null=True)
    date = models.DateField(db_column='Date', blank=True,
    null=True)  # Field name made lowercase.
    cost = models.IntegerField(blank=True, null=True)
    route = models.ForeignKey('Traindetails', models.DO_NOTHING, blank=True,
    null=True)
    trip = models.CharField(db_column='TRIP', max_length=30, blank=True,
    null=True)  # Field name made lowercase.
    time_slot = models.CharField(max_length=20, blank=True, null=True)
    class Meta:
        managed = False
        db_table = 'ticketdetails'

class help(models.Model):
    email = models.CharField(max_length=20, blank=True, null=True)
    contact = models.BigIntegerField(blank=True, null=True)
    name = models.ForeignKey('Metrouser', models.DO_NOTHING, db_column='uname',
    blank=True, null=True)
    message = models.IntegerField(blank=True, null=True)
    class Meta:
        managed = False
        db_table = 'help'
```

4.2 Pseudo code for Major Functionalities

4.2.1 Login

```
def login_check(request):
    if request.method == "POST":
         if request.POST.get('username')  and request.POST.get('password') :
            global username
            username = request.POST.get('username')
            global password
            password =  request.POST.get('password')
            print(username)
            print(password)
            userr = authenticate(request,username=username,password=password)
            if userr is not None:
                login(request,userr)
                messages.success(request,"Login successful")
                return redirect('/metro/user_check')
            else:
                messages.error(request, 'Login unsuccessful,try again.')
    return render(request,'templates/login.html')
```

4.2.2 Registration

```
def register_check(request):
    if request.method == "POST":
        print("Welcome to post reg")
        form = UserCreationForm(request.POST)
        if request.POST.get('username') and  request.POST.get('email') and
        request.POST.get('mno') and request.POST.get('password1') and
        request.POST.get('password2') :
            red2 = Metrouser()
            red2.username = request.POST.get('username')
            red2.email = request.POST.get('email')
            red2.mno =  request.POST.get('mno')
            red2.password =  request.POST.get('password1')
            red2.save()
            user = User.objects.create_user(username=
            request.POST.get('username'),password=
            request.POST.get('password1'),
            email =request.POST.get('email'))
            user.save()
            messages.success(request,"Registeration successful")
            return redirect('/metro/login_check')
        else:
            messages.error(request, 'Registeration unsuccessful,try again.')
            return render(request,'templates/reg.html')
     else:
        messages.error(request, 'Registeration unsuccessful,try again.')
        return render(request,'templates/reg.html')
```

4.2.3 Cost Generation

```
def User(request):
    if request.method == "POST":
        if request.POST.get('TRIP') and
          request.POST.get('source_id') and
          request.POST.get('dest_id') and
          request.POST.get('date') and
          request.POST.get('time_slot') and
          request.POST.get('gender') and request.POST.get('grp') :
            res =Ticketdetails.objects.latest("ticket_id")
            num = 1
            num = num + int(res.ticket_id)
            global cos
            cos = (int(request.POST.get('source_id')) -
                int(request.POST.get('dest_id')))*10
            if( cos <0):
                cos= cos-(cos *2)
            v= Station()
            w =Station()
            v.station_id = request.POST.get('source_id')
            w.station_id =request.POST.get('dest_id')
            m  = Metrouser()
            m.username = username
            u = Ticketdetails()
            u.ticket_id = num
            u.uname = m
            u.sourceid=v
            u.destid=w
            u.date=request.POST.get('date')
            u.trip = request.POST.get('TRIP')
            u.time_slot = request.POST.get('time_slot')
            g = request.POST.get('gender')
            gr = request.POST.get('grp')
            num= num+1
            u.cost = cos
            print(cos)
            print(u.cost)
            u.save()
        else:
            print("failure")
            return render(request,'templates/User.html')
        return redirect('/metro/payment_check')
    return render(request,'templates/User.html')
```

4.3 Database Connectivity

```
class Feedback(models.Model):
    email = models.CharField(max_length=20, blank=True, null=True)
    cno = models.BigIntegerField(primary_key=True)
    uname = models.ForeignKey('Metrouser', models.DO_NOTHING, db_column='uname',
    blank=True, null=True)
    detail_id = models.IntegerField(blank=True, null=True)
    class Meta:
        managed = False
        db_table = 'feedback'

class Metrouser(models.Model):
    username = models.CharField(primary_key=True, max_length=20)
    password = models.CharField(max_length=20, blank=True, null=True)
    mno = models.BigIntegerField(blank=True, null=True)
    email = models.CharField(max_length=30, blank=True, null=True)
    class Meta:
        managed = False
        db_table = 'Metrouser'

class Station(models.Model):
    s_name = models.CharField(max_length=20, blank=True, null=True)
    station_id = models.CharField(primary_key=True, max_length=20)
    class Meta:
        managed = False
        db_table = 'station'

class Ticketdetails(models.Model):
    ticket_id = models.CharField(primary_key=True, max_length=20)
    sourceid = models.ForeignKey(Station,models.DO_NOTHING,db_column='sourceid',
    blank=True, null=True,related_name='src')
    destid = models.ForeignKey(Station, models.DO_NOTHING, db_column='destid',
    blank=True, null=True,related_name='dest')
    uname = models.ForeignKey(Metrouser, models.DO_NOTHING, db_column='uname',
    blank=True, null=True)
    date = models.DateField(db_column='Date', blank=True,
    null=True)  # Field name made lowercase.
    cost = models.IntegerField(blank=True, null=True)
    route = models.ForeignKey('Traindetails', models.DO_NOTHING, blank=True,
    null=True)
    trip = models.CharField(db_column='TRIP', max_length=30, blank=True,
    null=True)  # Field name made lowercase.
    time_slot = models.CharField(max_length=20, blank=True, null=True)
    class Meta:
        managed = False
        db_table = 'ticketdetails'

class help(models.Model):
    email = models.CharField(max_length=20, blank=True, null=True)
    contact = models.BigIntegerField(blank=True, null=True)
    name = models.ForeignKey('Metrouser', models.DO_NOTHING, db_column='uname',
    blank=True, null=True)
    message = models.IntegerField(blank=True, null=True)
    class Meta:
        managed = False
        db_table = 'help'
```

4.4 Pseudo code for Major Functionalities

4.4.1 Login

```
def login_check(request):
    if request.method == "POST":
         if request.POST.get('username')  and request.POST.get('password') :
            global username
            username = request.POST.get('username')
            global password
            password =  request.POST.get('password')
            print(username)
            print(password)
            userr = authenticate(request,username=username,password=password)
            if userr is not None:
                login(request,userr)
                messages.success(request,"Login successful")
                return redirect('/metro/user_check')
            else:
                messages.error(request, 'Login unsuccessful,try again.')
    return render(request,'templates/login.html')
```

4.4.2 Registration

```
def register_check(request):
    if request.method == "POST":
        print("Welcome to post reg")
        form = UserCreationForm(request.POST)
        if request.POST.get('username') and  request.POST.get('email') and
        request.POST.get('mno') and request.POST.get('password1') and
        request.POST.get('password2') :
            red2 = Metrouser()
            red2.username = request.POST.get('username')
            red2.email = request.POST.get('email')
            red2.mno =  request.POST.get('mno')
            red2.password =  request.POST.get('password1')
            red2.save()
            user = User.objects.create_user(username=
            request.POST.get('username'),password=
            request.POST.get('password1'),
            email =request.POST.get('email'))
            user.save()
            messages.success(request,"Registeration successful")
            return redirect('/metro/login_check')
        else:
            messages.error(request, 'Registeration unsuccessful,try again.')
            return render(request,'templates/reg.html')
     else:
        messages.error(request, 'Registeration unsuccessful,try again.')
        return render(request,'templates/reg.html')
```

4.4.3 Cost Generation

```
def User(request):
    if request.method == "POST":
        if request.POST.get('TRIP') and
          request.POST.get('source_id') and
          request.POST.get('dest_id') and
          request.POST.get('date') and
          request.POST.get('time_slot') and
          request.POST.get('gender') and request.POST.get('grp') :
            res =Ticketdetails.objects.latest("ticket_id")
            num = 1
            num = num + int(res.ticket_id)
            global cos
            cos = (int(request.POST.get('source_id')) -
                int(request.POST.get('dest_id')))*10
            if( cos <0):
                cos= cos-(cos *2)
            v= Station()
            w =Station()
            v.station_id = request.POST.get('source_id')
            w.station_id =request.POST.get('dest_id')
            m  = Metrouser()
            m.username = username
            u = Ticketdetails()
            u.ticket_id = num
            u.uname = m
            u.sourceid=v
            u.destid=w
            u.date=request.POST.get('date')
            u.trip = request.POST.get('TRIP')
            u.time_slot = request.POST.get('time_slot')
            g = request.POST.get('gender')
            gr = request.POST.get('grp')
            num= num+1
            u.cost = cos
            print(cos)
            print(u.cost)
            u.save()
        else:
            print("failure")
            return render(request,'templates/User.html')
        return redirect('/metro/payment_check')
    return render(request,'templates/User.html')
```

4.5 Database Connectivity

```
class Feedback(models.Model):
    email = models.CharField(max_length=20, blank=True, null=True)
    cno = models.BigIntegerField(primary_key=True)
    uname = models.ForeignKey('Metrouser', models.DO_NOTHING, db_column='uname',
    blank=True, null=True)
    detail_id = models.IntegerField(blank=True, null=True)
    class Meta:
        managed = False
        db_table = 'feedback'

class Metrouser(models.Model):
    username = models.CharField(primary_key=True, max_length=20)
    password = models.CharField(max_length=20, blank=True, null=True)
    mno = models.BigIntegerField(blank=True, null=True)
    email = models.CharField(max_length=30, blank=True, null=True)
    class Meta:
        managed = False
        db_table = 'Metrouser'

class Station(models.Model):
    s_name = models.CharField(max_length=20, blank=True, null=True)
    station_id = models.CharField(primary_key=True, max_length=20)
    class Meta:
        managed = False
        db_table = 'station'

class Ticketdetails(models.Model):
    ticket_id = models.CharField(primary_key=True, max_length=20)
    sourceid = models.ForeignKey(Station,models.DO_NOTHING,db_column='sourceid',
    blank=True, null=True,related_name='src')
    destid = models.ForeignKey(Station, models.DO_NOTHING, db_column='destid',
    blank=True, null=True,related_name='dest')
    uname = models.ForeignKey(Metrouser, models.DO_NOTHING, db_column='uname',
    blank=True, null=True)
    date = models.DateField(db_column='Date', blank=True,
    null=True)  # Field name made lowercase.
    cost = models.IntegerField(blank=True, null=True)
    route = models.ForeignKey('Traindetails', models.DO_NOTHING, blank=True,
    null=True)
    trip = models.CharField(db_column='TRIP', max_length=30, blank=True,
    null=True)  # Field name made lowercase.
    time_slot = models.CharField(max_length=20, blank=True, null=True)
    class Meta:
        managed = False
        db_table = 'ticketdetails'

class help(models.Model):
    email = models.CharField(max_length=20, blank=True, null=True)
    contact = models.BigIntegerField(blank=True, null=True)
    name = models.ForeignKey('Metrouser', models.DO_NOTHING, db_column='uname',
    blank=True, null=True)
    message = models.IntegerField(blank=True, null=True)
    class Meta:
        managed = False
        db_table = 'help'
```

4.6 Pseudo code for Major Functionalities

4.6.1 Login

```
def login_check(request):
    if request.method == "POST":
        username = request.POST.get('username')
        password =  request.POST.get('password')
        product = Metrouser.objects.get(username = username)
        userr = authenticate(request,username = username ,password = password)
        if userr is not None and product.admin is True:
            login(request,userr)
            messages.success(request,"Login successful")
            return redirect('/metro/admin_check')
        elif userr is not None :
            login(request,userr)
            print(request.user.username)
            return redirect('/metro/user_check')
        else:
            messages.error(request, 'Login unsuccessful,try again.')
    return render(request,'templates/login.html')
```

4.6.2 Registration

```
def register_check(request):
    if request.method == "POST":
        print("Welcome to post reg")
        red2 = Metrouser()
        red2.username = request.POST.get('username')
        red2.email = request.POST.get('email')
        red2.mno =  request.POST.get('mno')
        red2.password =  request.POST.get('password1')
        red2.save()
        user = User.objects.create_user(username=request.POST.get('username'), pas
        user.save()
```

```
            messages.success(request,"Registeration successful")
            return redirect('/metro/login_check')
        return render(request,'templates/reg.html')
```

4.6.3 Cost Generation

```
def User(request):
    if request.method == "POST":
        if request.POST.get('TRIP') and
          request.POST.get('source_id') and
          request.POST.get('dest_id') and
          request.POST.get('date') and
          request.POST.get('time_slot') and
          request.POST.get('gender') and request.POST.get('grp') :
            res =Ticketdetails.objects.latest("ticket_id")
            num = 1
            num = num + int(res.ticket_id)
            global cos
            cos = (int(request.POST.get('source_id')) -
                int(request.POST.get('dest_id')))*10
            if( cos <0):
                cos= cos-(cos *2)
            v= Station()
            w =Station()
            v.station_id = request.POST.get('source_id')
            w.station_id =request.POST.get('dest_id')
            m  = Metrouser()
            m.username = username
            u = Ticketdetails()
            u.ticket_id = num
            u.uname = m
            u.sourceid=v
            u.destid=w
            u.date=request.POST.get('date')
            u.trip = request.POST.get('TRIP')
            u.time_slot = request.POST.get('time_slot')
            g = request.POST.get('gender')
            gr = request.POST.get('grp')
            num= num+1
            u.cost = cos
            print(cos)
            print(u.cost)
            u.save()
        else:
            print("failure")
            return render(request,'templates/User.html')
        return redirect('/metro/payment_check')
    return render(request,'templates/User.html')
```

Chapter 5

Testing

5.1 Testing Implementation

We have carried out unit testing and also have carried out integration testing and full end to end system testing to ensure that few or no bugs found a place in the deployed application.

5.1.1 Unit testing:

This was done at the module level where basic components of the software were tested to verify its functionality.

5.1.2 Integration testing:

This was used to test the combined modules as a group to identify defects in the interfaces between integrated components.

5.1.3 System testing:

The complete integrated system was tested to verify whether all components work efficiently and effectively as a whole.

5.2 Results of Implementation:

It demonstrates that it will have a web interface where user of the system can logon and register, where clients can book a ticket, can view route map. Interface where the admin can delete a user System admin can manage all users of the system such us users and other admins.

5.3 Black Box Testing

5.3.1 Admin

- When the application is made to run on the server home page of the admin will open then click on the login in the home page then give valid admin Id and password then click on login button.
- Once the admin login is successful, it will show dashboard page there we have a plenty of options including the Number of stations, add/remove user, add/remove admin ,add/remove stations.

5.3.2 User

- Open the webpage there will be login button on the right top corner, click on that and select signup fill out the required columns, then click on signup. Again click on login/register button fill the email id and password then click on login.
- After signing in ,fill the ticket form by entering starting and ending stations click on submit.
- To change the password then click on update password on your profile, type new password and click save.
- To send any feedback then click on feedback in contact option and type your email and message then click on send.
- To ask for any help then click on help in contact option and type your email and your query then click on send.

5.4 Results of Testing:

Cross browser testing was done me to ensure that the web application looks the same in major browsers that is Google chrome, Mozilla Firefox, Opera and Internet explorer. The web project is consistent (looks exactly the same) in Google chrome, Mozilla Firefox and Opera but the looks vary slightly in internet explorer.

5.5 Achievements of the project:

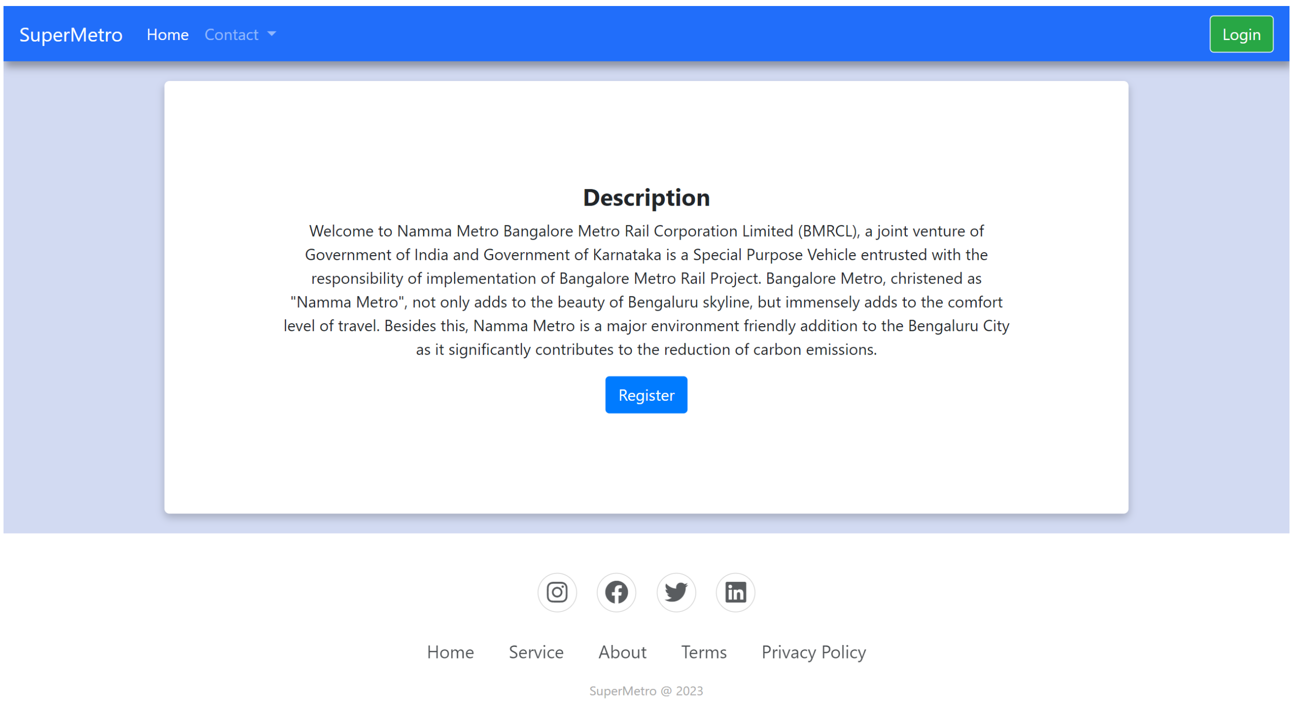
- Clients can create accounts on the system through registration.

- System Administrator can manage the Customers/Subscribers by creating for them the account.
- Customers are able to book Tickets for their usage.
- System Administrators can manage Routes and also he can manage the Customers.
- System Administrator can efficiently manage all the users on the application and roles.

Chapter 6

Results & Snapshots

This page provides a brief description about the website.



Description

Welcome to Namma Metro Bangalore Metro Rail Corporation Limited (BMRCL), a joint venture of Government of India and Government of Karnataka is a Special Purpose Vehicle entrusted with the responsibility of implementation of Bangalore Metro Rail Project. Bangalore Metro, christened as "Namma Metro", not only adds to the beauty of Bengaluru skyline, but immensely adds to the comfort level of travel. Besides this, Namma Metro is a major environment friendly addition to the Bengaluru City as it significantly contributes to the reduction of carbon emissions.

Figure 6.1: Home

Here user or admin can log-in using their credentials.

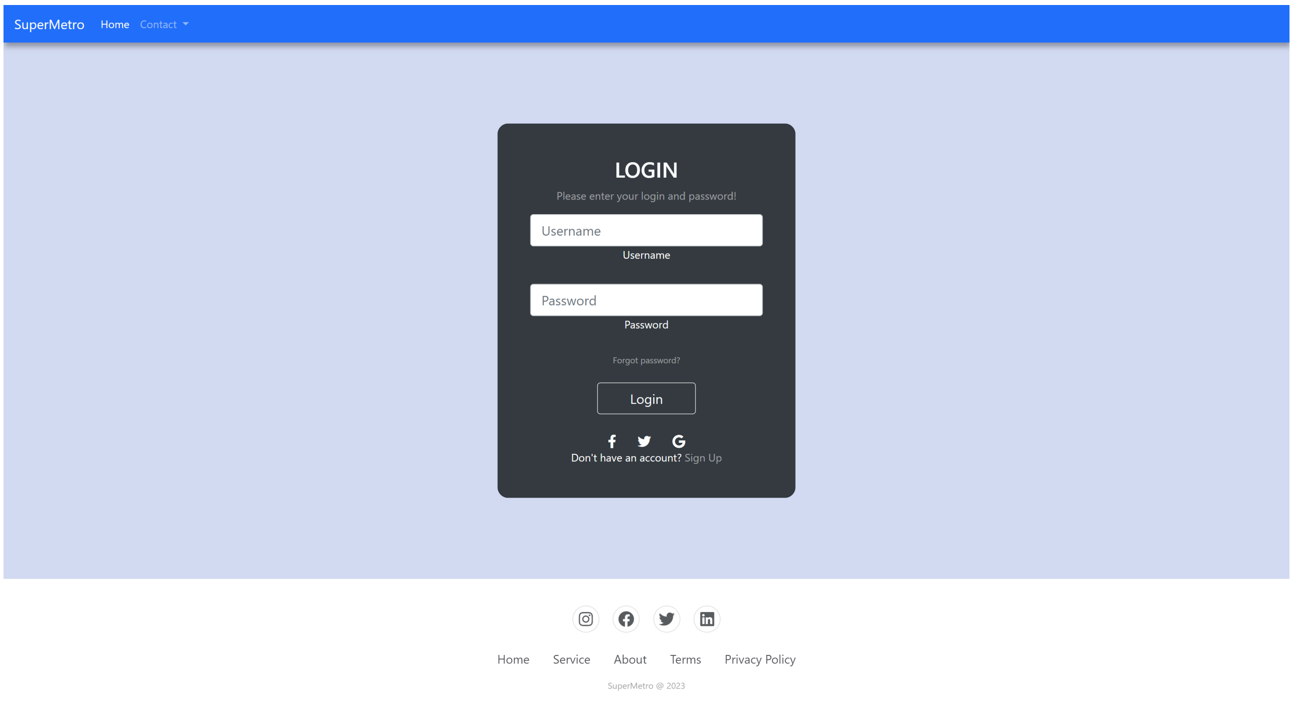


Figure 6.2: Login

New users can register here to access the website.

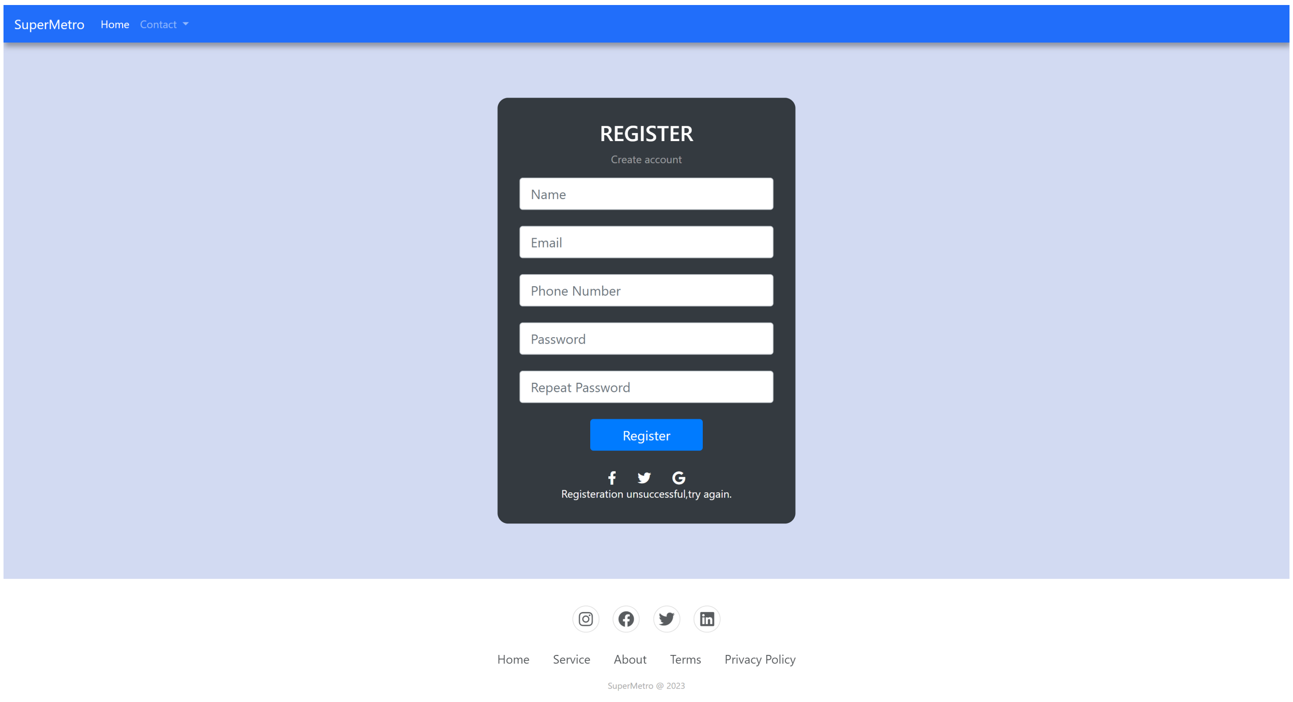


Figure 6.3: Registration

This main page allows users to enter their ticket details.

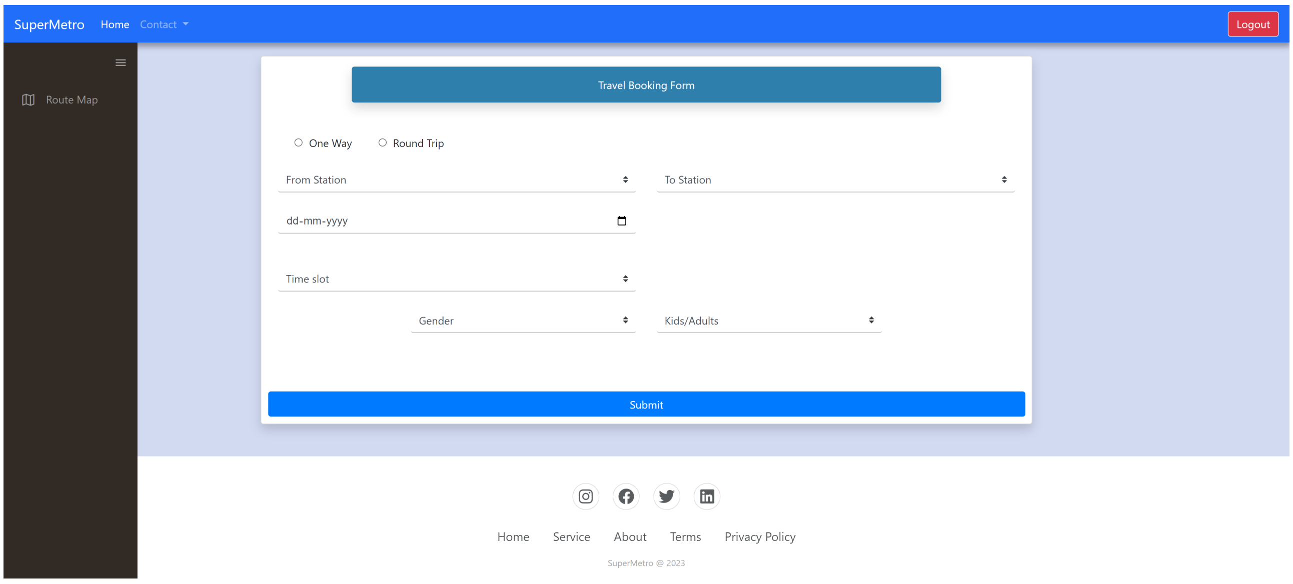


Figure 6.4: User

In case of any issues users can seek help from admin through this page.

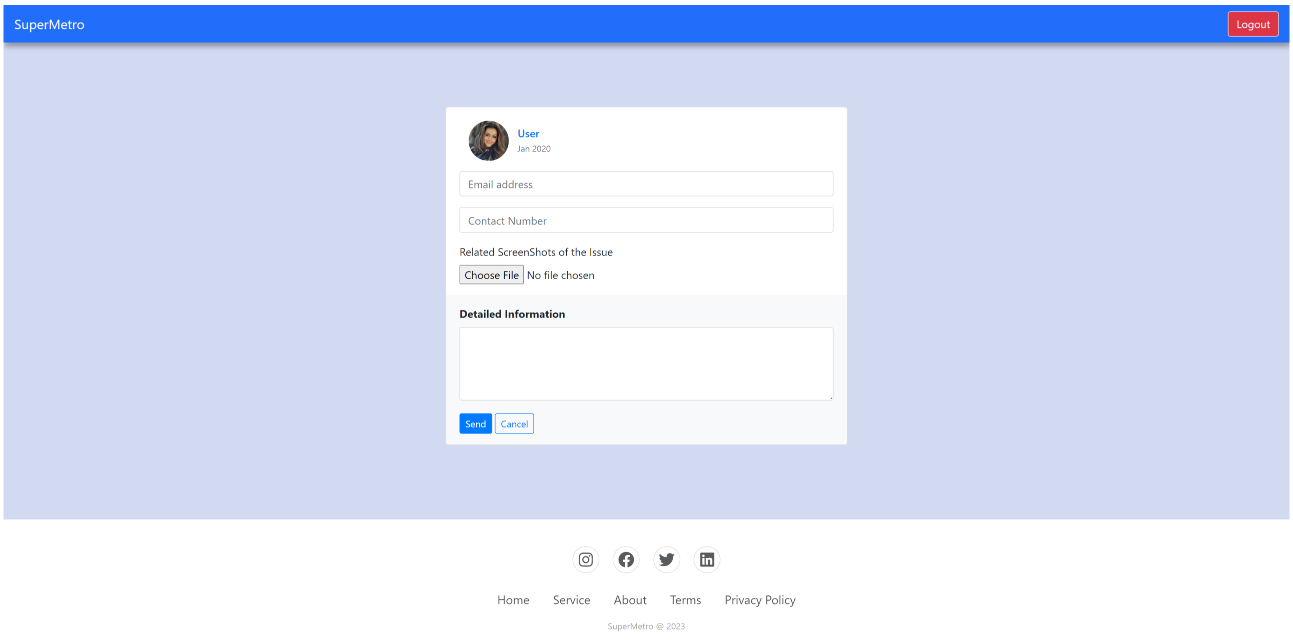


Figure 6.5: Help

Feedback page allows users to share their experience with the website to the admin.

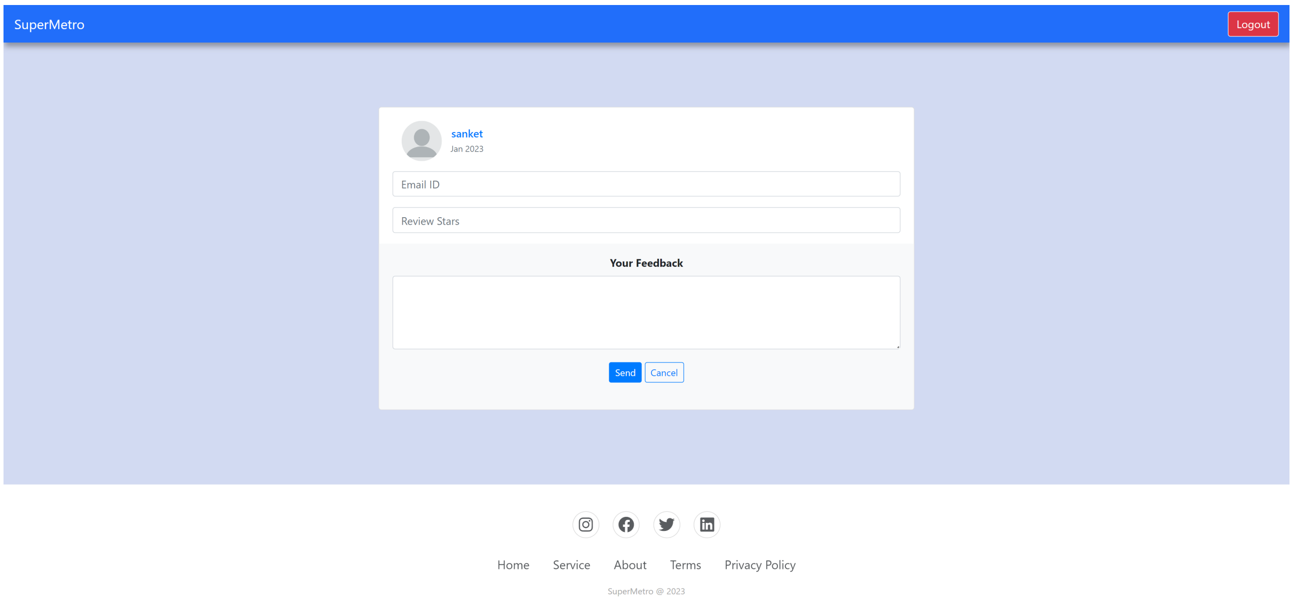


Figure 6.6: Feedback

User can make the payment here.

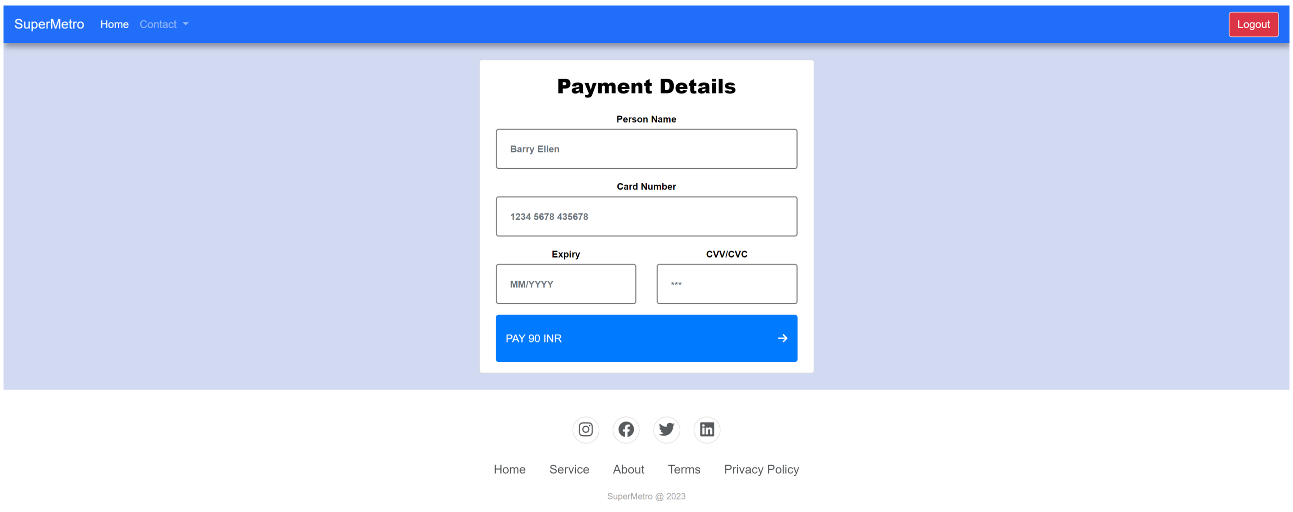
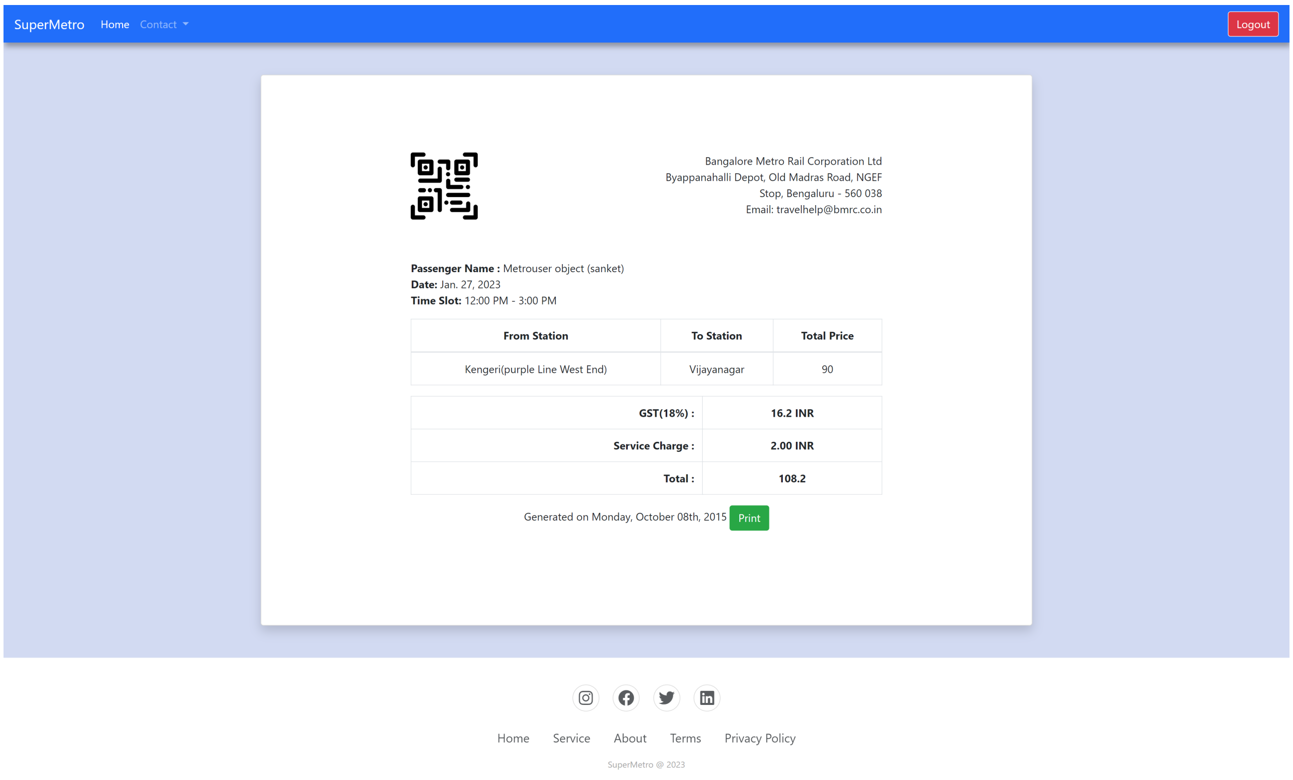


Figure 6.7: Payment

The ticket based on the information given by the user will be generated here.



Bangalore Metro Rail Corporation Ltd
Byappanahalli Depot, Old Madras Road, NGEF
Stop, Bengaluru - 560 038
Email: travelhelp@bmrc.co.in

Passenger Name : Metrouser object (sanket)
Date: Jan. 27, 2023
Time Slot: 12:00 PM - 3:00 PM

From Station	To Station	Total Price
Kengeri(purple Line West End)	Vijayanagar	90

GST(18%) :	16.2 INR
Service Charge :	2.00 INR
Total :	108.2

Generated on Monday, October 08th, 2015

Figure 6.8: Ticket

This Admin interface allows the admin to handle the station information, add and delete the user and subordinate admin accounts and also to inspect on the complaint and feedback given by the users.

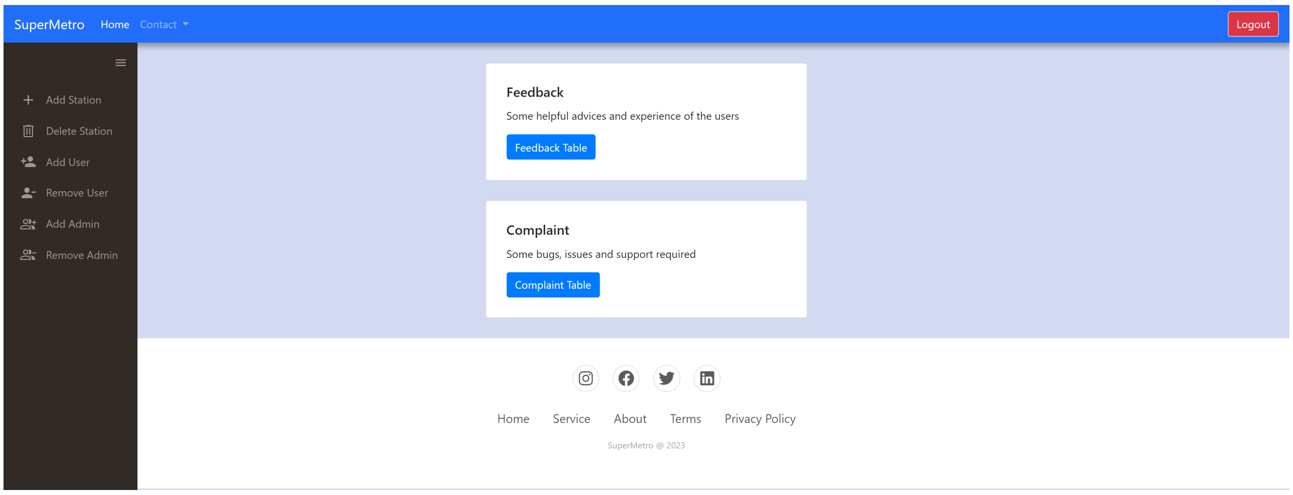


Figure 6.9: Admin

Chapter 7

Conclusion & Future Enhancements

7.1 Conclusion

the implementation of a metro management system is a complex and challenging task that requires careful planning, design, and execution. The system is very much required plays a important role in the smooth and efficient operation of a metro rail system, and it provides a wide range of benefits such as improved resource management, real-time monitoring and control, enhanced passenger experience and more. The project is be a great opportunity for us to learn about the different aspects of metro management and gain practical experience in system development and implementation. The successful completion of this project demonstrates our ability to work in a team, manage a project, and apply the knowledge and skills acquired during their studies.

7.2 Future Enhancements

A basic metro management database project typically includes features such as schedule and route information, ticketing and fare management, and real-time train tracking. However, there are several ways in which this type of project could be enhanced in the future to improve the overall experience for both metro operators and riders.

One potential enhancement would be the integration of predictive analytics. By analyzing historical data, predictive analytics algorithms can forecast future demand for certain routes and trains, allowing metro operators to optimize their schedules and resources accordingly. This could lead to more efficient use of resources and a reduction in overcrowding and delays.

Another enhancement could be the integration of artificial intelligence (AI) and machine learning (ML) technologies. For example, AI-powered chatbots could be used to assist riders with trip planning and customer service inquiries. ML-powered algorithms could also be used to optimize the routing and scheduling of trains, taking into account factors such as traffic, weather, and special events.

The project could also be enhanced with the integration of the Internet of Things (IoT) technology. This would enable the collection of real-time data from sensors on trains and at stations, providing valuable insights into the performance and condition of metro infrastructure. This could be used to identify potential issues and maintenance needs, allowing operators to take preventative measures and avoid disruptions.

Another feature that could be added is integration with other transportation modes like bus, bike-sharing, and car-sharing services, enabling the users to plan their journey across multiple modes and purchase tickets for the entire journey. This would improve the overall user experience and encourage the use of public transportation.

In conclusion, there are many ways to enhance a basic metro management database project. By integrating predictive analytics, AI/ML, IoT, and other emerging technologies, metro operators can improve the efficiency and reliability of their service, while riders can benefit from more convenient and personalized experiences.

References

- Django documentation ://docs.djangoproject.com/en/4.1/
- getbootstrap ://getbootstrap.com/
- CodeWithHarry-Youtube Playlist
- W3Schools ://www.w3schools.com/
- GeekforGeeks ://www.geeksforgeeks.org/