RIDESHARE

Mini Project submitted in partial fulfillment of the requirement for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

Under the esteemed guidance of

Dr K. Krishna Jyothi
Associate Professor

By

ABHIRAM VENKATA DAITA (21R11A05A6)
MUDDAM SAI MANISH YADAV (21R11A05D5)
RAGI RAKSHITHA REDDY (21R11A05E1)

Department of Computer Science and Engineering
Accredited by NBA

Geethanjali College of Engineering and Technology
(UGC Autonomous)
(Affiliated to J.N.T.U.H, Approved by AICTE, New Delhi)
Cheeryal (V), Keesara (M), Medchal.Dist.-501 301.

August-2024

Geethanjali College of Engineering & Technology

(UGC Autonomous)

(Affiliated to JNTUH, Approved by AICTE, New Delhi)
Cheeryal (V), Keesara(M), Medchal Dist.-501 301.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
Accredited by NBA

CERTIFICATE

This is to certify that the B.Tech Mini Project report entitled **"RIDESHARE"** is a bonafide work done by **Abhiram Venkata Daita(21R11A05A6), Muddam Sai Manish Yadav (21R11A05D5), Ragi Rakshitha Reddy(21R11A05E1),** in partial fulfillment of the requirement of the award for the degree of Bachelor of Technology in "**Computer Science and Engineering**" from Jawaharlal Nehru Technological University, Hyderabad during the year 2023-2024.

Internal Guide HOD - CSE

Dr K. Krishna Jyothi **Dr A SreeLakshmi**

Associate Professor Professor

External Examiner

Geethanjali College of Engineering & Technology

(UGC Autonomous)

(Affiliated to JNTUH Approved by AICTE, New Delhi)
Cheeryal (V), Keesara(M), Medchal Dist.-501 301.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Accredited by NBA

DECLARATION BY THE CANDIDATE

We, **Abhiram Venkata Daita, Muddam Sai Manish Yadav**, **Ragi Rakshitha Reddy**, bearing Roll Nos. **21R11A05A6, 21R11A05D5, 21R11A05E1**, hereby declare that the project report entitled **"RIDESHARE"** is done under the guidance of **Dr K. Krishna Jyothi,Associate Professor** Department of Computer Science and Engineering, Geethanjali College of Engineering and Technology, is submitted in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology in Computer Science and Engineering**.

This is a record of bonafide work carried out by me/us in **Geethanjali College of Engineering and Technology** and the results embodied in this project have not been reproduced or copied from any source. The results embodied in this project report have not been submitted to any other University or Institute for the award of any other degree or diploma.

Abhiram Venkata Daita(21R11A05A6)
Muddam Sai Manish Yadav(21R11A05D5)
Ragi Rakshitha Reddy(21R11A05E1)

Department of CSE,
Geethanjali College of Engineering and Technology,
Cheeryal.

ACKNOWLEDGEMENT

We would like to express our sincere thanks to Dr. A. Sree Lakshmi, Professor, Head of Department of Computer Science, Geethanjali College of Engineering and Technology, Cheeryal, whose motivation in the field of software development has made us to overcome all hardships during the course of study and successful completion of project.

We would like to express our profound sense of gratitude to all for having helped us in completing this dissertation. We would like to express our deep-felt gratitude and sincere thanks to our guide Dr. K. Krishna Jyothi, Assosiate Professor, Department of Computer Science, Geethanjali College of Engineering and Technology, Cheeryal, for his skillful guidance, timely suggestions and encouragement in completing this project sucessfully.

would like to express our sincere gratitude to our Principal Prof. Dr. S. Udaya Kumar for providing the necessary infrastructure to complete our project. We are also thankful to our Secretary Mr. G.R. Ravinder Reddy for providing an interdisciplinary & progressive environment.

Finally, we would like to express our heartfelt thanks to our parents who were very supportive both financially and mentally and for their encouragement to achieve our set goals.

Abhiram Venkata Daita(21R11A05A6)

Muddam Sai Manish Yadav(21R11A05D5)

Ragi Rakshitha Reddy(21R11A05E1)

ABSTRACT

In urban settings, transportation poses a significant challenge, often leading to inefficiencies and increased costs for commuters. To address this issue, we propose RideShare, a cab booking system that enables users traveling from one location to another to share the cost of their journey with fellow travelers. This project aims to enhance convenience, affordability, and sustainability in transportation by leveraging technology to facilitate ridesharing among users.

RideShare will be developed using modern web development technologies and APIs. The backend will be built using Django, a Python-based framework, coupled with PostgreSQL for data storage. Authentication and authorization will be implemented to ensure secure access to the platform. Mapping functionalities will be integrated using the Mapbox API to enable users to specify pickup and drop-off locations accurately. Real-time communication between users will be facilitated through ajax technology, allowing for seamless coordination of ridesharing arrangements.

The implementation of RideShare will result in a user-friendly platform that empowers users to find and offer rides conveniently. By enabling users to share the cost of their journeys, RideShare will promote cost savings and reduce the environmental impact of transportation. Real-time communication features will enhance user experience.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

- **HTML**: HyperText Markup Language
- **CSS**: Cascading Style Sheets
- **UML**: Unified Modeling Language
- **CSRF**: Cross-Site Request Forgery
- **AJAX**: Asynchronous JavaScript and XML
- **JS**: Java Script

TABLE OF CONTENTS

1. INTRODUCTION

1.1 ABOUT THE PROJECT

Urban transportation often presents significant challenges, including high travel costs, traffic congestion, and environmental concerns due to the large number of vehicles on the road. Traditional cab services are expensive. Additionally, there is a lack of affordable and efficient options for people who wish to share rides with others traveling along the same destination. This creates inefficiencies in transportation, leading to wasted resources and unnecessary expenses for commuters.

To address these challenges, there is a need for a platform that enables users to share rides easily, reducing the number of vehicles on the road, cutting down on travel costs, and promoting environmental sustainability. The platform will facilitate seamless communication between users, ensure safety, transparency, and privacy, and provide a reliable system for coordinating ride-sharing arrangements.

RideShare is designed to solve these problems by offering a web-based solution that allows users to publish their travel plans and share rides with others. By leveraging technology, RideShare aims to make urban transportation more affordable, efficient, and eco-friendly.

1.2 OBJECTIVE

- **Enable Users to Share Rides:** Allow users to easily publish their travel details, such as source, destination, date, time, and vehicle information, and provide a system for others to search for and join these rides.

- **Enhance Communication and Coordination:** Integrate real-time communication features, including chat functionality, to enable users to coordinate ride-sharing details effectively.

- **Ensure Safety and Trust: Implement** user authentication, profile management, and a rating system to build a trustworthy community where users can provide feedback on their experiences.

- **Provide Timely Notifications:** Use email notifications to keep users informed about ride confirmations, cancellations, and other important updates.

- **Support Scalability and Future Development:** Build the platform with scalability in mind, allowing for future enhancements such as mobile application development and the integration of additional features to meet evolving user needs.

2. SYSTEM ANALYSIS

2.1 EXISTING SYSTEM

Carpooling platforms like BlaBlaCar, Uber, Ola has revolutionized how people share rides, offering a cost-effective and eco-friendly alternative to traditional transportation methods. By connecting drivers and passengers traveling along similar routes, these platforms aim to make travel more accessible and affordable. However, despite their popularity, these systems come with certain limitations and challenges that affect user experience, safety, and cost efficiency.

Existing System Challenges

While platforms like BlaBlaCar have made significant strides in promoting carpooling, several drawbacks still hinder their overall effectiveness

- **Safety Concerns:** Although safety features are integrated into these platforms, the inherent risk of traveling with strangers remains. Users must rely heavily on trust and personal judgment, as the platforms often provide limited transparency regarding passenger behavior and demographics.
- **Flexibility Limitations for Drivers:** Drivers may find their flexibility constrained as they need to accommodate passengers' needs. This can limit their ability to adjust routes or departure times, potentially leading to inconveniences for both parties.
- **Limited Transparency:** The lack of detailed information about fellow passengers can create safety and trust concerns. Users often have minimal insight into the backgrounds of the individuals they are sharing rides with, which can be unsettling.
- **User Experience Issues:** Users may experience difficulties in communication with drivers and passengers, leading to misunderstandings or unmet expectations. Additionally, the limited control over ride arrangements can result in frustration and a less-than-optimal experience.

2.2 PROPOSED SYSTEM

In response to the challenges faced by existing carpooling platforms, our proposed system, RideShare, is designed to overcome these limitations by offering a more user-friendly, secure, and flexible experience. RideShare integrates several key features aimed at addressing the shortcomings of traditional carpooling services.

Proposed System advantages

- **Rating System for Publishers and Passengers:** To enhance trust and accountability, RideShare implements a robust rating system that allows both publishers (drivers) and passengers to rate each other after a ride. This feedback mechanism ensures that users can make informed decisions based on previous experiences, fostering a safer and more reliable community.
- **Accept/Reject Mechanism for Publishers:** RideShare empowers publishers with the ability to accept or reject ride requests from passengers. This feature gives publishers greater control over who they share their ride with, ensuring a more comfortable and secure experience. Publishers can make decisions based on passenger details, ratings, and travel preferences.
- **Flexible Scheduling for Publishers:** Unlike traditional platforms where drivers might feel compelled to adjust their schedules for passengers, RideShare allows publishers to set their preferred departure times. This flexibility ensures that publishers can plan their journeys according to their own needs, with passengers joining rides that fit their schedules.
- **Chat Option for Communication:** Effective communication is vital for a smooth ride-sharing experience. RideShare includes an in-app chat feature that allows publishers and passengers to communicate directly before and during the ride. This feature helps coordinate details such as pickup locations, luggage requirements, and other preferences, reducing the likelihood of misunderstandings.

By incorporating these features, RideShare aims to deliver a more secure, flexible, and user-centric carpooling platform, addressing the key issues present in existing systems while enhancing overall user satisfaction.

2.3 FEASIBILITY STUDY

2.3.1 DETAILS

The RideShare project is a web-based carpooling service developed using Django for the backend and PostgreSQL for data storage, HTML, CSS, and JavaScript for the front end, with integration of the Mapbox API to provide location suggestions. The primary goal is to provide a platform where users can share rides, reducing transportation costs and environmental impact. The feasibility of this project is high due to the increasing demand for cost-effective and environmentally friendly transportation solutions in urban areas.

2.3.2 IMPACT ON ENVIRONMENT

RideShare has a positive impact on the environment in several ways

- **Reduction in Pollution:** By encouraging carpooling, the number of vehicles on the road is reduced, leading to lower emissions and decreased air pollution.
- **Reduction in Global Warming:** Fewer vehicles mean reduced greenhouse gas emissions, contributing to the fight against global warming.
- **Resource Optimization:** Sharing rides optimizes fuel consumption and vehicle usage, reducing the need for additional vehicles and the associated environmental impact.
- **Decreased Traffic Congestion:** By promoting shared rides, RideShare helps decrease traffic congestion, which further reduces fuel consumption and travel time.
- **Reduction in Noise Pollution:** Fewer vehicles on the road lead to a decrease in noise pollution, contributing to a more peaceful urban environment.

2.3.3 SAFETY

Safety is a crucial aspect of the RideShare platform:

- **Data Security:** User information is stored securely in the PostgreSQL database, with encryption techniques applied to sensitive data such as passwords.
- **Privacy:** User data, including personal details , is protected and only accessible to the user and relevant parties (e.g., ride participants).
- **User Verification:** The platform ensures that users are authenticated before accessing the service, protecting against unauthorized access.
- **CSRF Protection:** Django's built-in CSRF protection helps prevent cross-site request forgery attacks by ensuring that requests made to the server are authorized by the user.

2.3.4 ETHICS

Ethical considerations in the development and use of RideShare include:

- **Non-Harmful:** The platform is designed to ensure that no user is harmed, either physically or virtually, through the use of the service.
- **Privacy Protection:** Users personal information, including contact details and travel history, is kept private and not exposed to other users.

2.3.5 COST

- **Development Cost:** The development cost of RideShare was kept minimal by using open-source technologies such as Django and PostgreSQL. The only significant cost involved was the usage of the Mapbox API, which was within the free tier limits.
- **Usage Cost:** There is no cost for users to use the platform.
- **Maintenance Cost:** The ongoing maintenance cost is minimal, primarily involving periodic updates to the software.

- **Cost Reduction:** The implementation of RideShare leads to cost savings for users by reducing the need for individual transportation, which lowers fuel and maintenance expenses.

2.3.6 TYPE

RideShare is a web application designed for desktop browsers, with plans for future development of a mobile application. It is a product intended to facilitate carpooling among users in urban areas.

2.4 SCOPE OF THE PROJECT

The scope of the RideShare project encompasses the development of a comprehensive web-based carpooling platform designed to connect users traveling in the same destination. The project includes the creation of user-friendly interfaces for both ride publishers and passengers, along with secure authentication and authorization mechanisms to ensure user privacy and data security. The platform integrates geolocation services through the Mapbox API, allowing users to specify accurate pickup and drop-off locations through location suggestions. RideShare also includes real-time communication features, enabling seamless interaction between users to coordinate ride-sharing arrangements. The platform's scalability allows for future enhancements, such as the development of a mobile application and the integration of additional features like payment processing and advanced ride scheduling. Overall, RideShare aims to provide a sustainable and cost-effective transportation solution, reducing environmental impact by encouraging carpooling and optimizing resource usage. The project is built on a robust backend infrastructure using Django and PostgreSQL, ensuring reliability, security, and ease of maintenance.

2.5 SYSTEM CONFIGURATION

Software Requirements:

- WEB BROWSER GOOGLE OR ANY COMPACTIBLE BROWSER
- **Operating System:** Windows 11
- **Development Tools:** Visual Studio Code
- **Development Framework:** Django
- **Database:** PostgreSQL
- **Frontend Technologies:** HTML, CSS, JavaScript
- **API:** Mapbox API

Hardware Requirements:

- **System:** Intel i5
- **Memory:** Minimum 4 GB RAM.
- **Internet:** Stable internet connection.

3. LITERATURE OVERVIEW

3.1 LITERATURE REVIEW

1. PoliUniPool: a carpooling system for universities

The article discusses the PoliUniPool carpooling system designed for students and staff at two universities in Milan. The system matches users for shared rides, optimizes routes, and schedules based on user preferences and constraints. Key features include selecting environmentally friendly destinations, immediate notifications for delays, cost-sharing estimates, and social networking functionalities. An algorithm is used to maximize user satisfaction by considering factors like route length and user preferences. A trial of the system was planned to assess its implementation and promotion strategies

2. Environmental impacts of shared mobility: a systematic literature review of life-cycle assessments focusing on car sharing, carpooling, bikesharing, scooters and moped sharing.

Shared mobility services such as car sharing, carpooling, bikesharing, and scooter/moped sharing have been the subject of numerous Life-Cycle Assessment (LCA) studies aimed at evaluating their environmental impacts. Car sharing, which reduces the number of private vehicles, can potentially lower emissions but shows mixed results depending on user behavior and vehicle types (Shaheen & Cohen, 2013). Carpooling generally reduces per capita emissions by sharing rides, though its effectiveness is influenced by vehicle efficiency and passenger numbers (Noland et al., 2008). Bikesharing programs are found to be environmentally beneficial when they replace car trips, although the benefits are affected by bike production and disposal (Fishman et al., 2013). Scooter and moped sharing, while potentially reducing emissions compared to gasoline vehicles, faces challenges related to battery production and disposal (Gössling & Choi, 2020; He et al., 2021). Overall, the environmental impact of these shared mobility modes is context-dependent, influenced by factors such as travel behavior, design, and implementation strategies (Martens, 2007; Kietzmann et al., 2019).

3. What Encourages People to Carpool? A Conceptual Framework of Carpooling Psychological Factors and Research Propositions.

This paper investigates the psychological factors that influence carpooling behavior, using the Theory of Planned Behaviour and the Norm-Activation Model to guide a systematic literature review. The study identifies eighteen factors affecting both drivers and passengers, proposing a conceptual framework and six research propositions. It also highlights the relevance of Consumer Perceived Value, Social Capital, and the Technology Acceptance Model. The paper suggests further research directions, including diverse data collection and the impact of COVID-19. Overall, it offers valuable insights for enhancing carpooling practices through a better understanding of psychological motivations.

4. Review of Carpooling System Abstract

The abstract describes a carpooling system designed to address urban transportation challenges by connecting drivers with passengers through a user-friendly mobile application. This system targets issues like traffic congestion and environmental impact by facilitating shared mobility, thus reducing vehicle miles traveled and carbon emissions.

Key features include a mobile app for ride matching, dynamic routing algorithms for trip optimization, and incentive structures to boost user engagement. The project uses both quantitative metrics (such as reductions in vehicle miles and greenhouse gas emissions) and qualitative feedback to evaluate its effectiveness.

The paper aims to contribute to sustainable transportation solutions by leveraging modern technology and data analytics. It offers valuable insights for policymakers, urban planners, and transportation stakeholders interested in promoting carpooling. The use of technologies like HTML, CSS, JavaScript, Django, Bootstrap, MySQL, and Python underscores the project's technical depth. Overall, the abstract effectively communicates the project's goals, methodologies, and anticipated impacts.

4. SYSTEM DESIGN

4.1 SYSTEM ARCHITECTURE

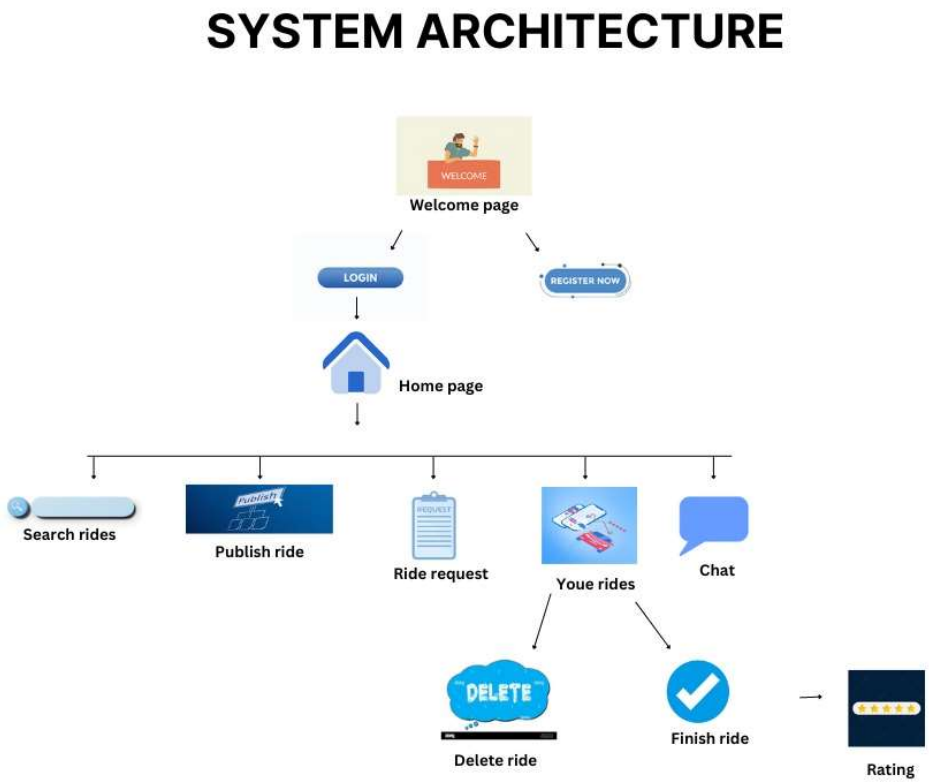

Fig 4.1 System Architecture

- **Login/Register:** The authentication page where users can either log in with existing credentials or create a new account, providing access to the system.
- **Search Rides:** A feature that allows users to search for available rides by entering specific criteria like source, destination.
- **Publish Rides:** A module that enables users (publishers) to create and list a new ride, including details such as departure point, destination, time, vehicle details, and available seats.

- **Ride Request:** A system component where users can request to join a published ride, and publishers can accept or reject these requests.
- **Your Rides:** A dashboard where users can view rides they have published as well as rides they are participating in as passengers. This section may also include options for rating rides and marking them as completed.
- **Chat:** A real-time communication feature allowing users to exchange messages, typically within a specific ride's context, enhancing coordination and interaction between publishers and passengers**.**

4.1.1MODULE DESCRIPTION

1. User Authentication and Authorization

The User Authentication and Authorization module secures access to the RideShare platform by managing the login and registration processes. The Login Module is responsible for handling user authentication by validating credentials against the platform's database. It establishes secure sessions for users, ensuring that only authorized individuals can access the platform's features. The Registration Module facilitates the onboarding of new users by allowing them to sign up on the platform. It securely collects and stores user details such as name, date of birth, contact information, and other relevant data.

2. Ride Management

The Ride Management module comprises the Publisher Module and the Passenger Module, which together form the core of the RideShare platform's functionality. The Publisher Module empowers users to offer carpooling services by allowing them to publish ride offers. Users can specify critical ride details such as the pickup location, destination, departure time, vehicle information, and the number of available seats. This module also enables publishers to manage their published rides, with options to cancel ride. On the other hand, the Passenger Module enables users to search for and join rides that align with their travel preferences. Users can search for rides using source and destination, and can view detailed ride information before requesting to join. Once a request is made, the module

handles the confirmation process, allowing passengers to book rides and manage their bookings and cancellations effectively.

3. Communication Module

The Communication Module is designed to enhance interaction between users through a real-time chat platform, ensuring smooth coordination for ride-sharing. Implemented using AJAX, this module facilitates instant messaging between ride publishers and passengers, enabling them to coordinate details such as pickup points, timing, and costs seamlessly. The real-time communication capability helps users make on-the-fly adjustments and ensures that all parties are kept informed about the ride. Additionally, the module preserves chat history for future reference, allowing users to revisit past conversations if needed.

4. Location Suggestion Module

The Location Suggestion Module leverages the Mapbox API to provide users with accurate and relevant location suggestions, enhancing the overall experience on the platform. This module assists users in easily searching for and selecting specific pickup and drop-off points. By offering precise location suggestions, it ensures that both publishers and passengers can identify and choose the most appropriate locations for their journeys.

5. Rating System

The Rating System module fosters accountability and trust on the RideShare platform by allowing users to rate each other. Passenger Ratings let passengers rate publishers and co-passengers on ride quality, comfort, and overall experience, with these ratings displayed in available rides during searches to guide future choices. Publisher Ratings enable publishers to rate passengers based on behavior and punctuality, visible to other publishers when a passenger requests a ride. This mutual feedback system helps maintain a high standard of service and user reliability.

6. Email Notification Module

The Email Notification Module is integral to keeping users informed and engaged by sending timely and relevant notifications throughout their interaction with the platform. Users receive notifications when they receive ride requests from passengers, when their requests are accepted or rejected, and when a ride they are involved in is canceled. Additionally, the module sends confirmation emails upon successful user registration, ensuring that users are kept informed from the moment they join the platform.

7. Ride Details Module

The Ride Details Module provides a comprehensive overview of rides, offering both publishers and passengers detailed information about their journeys. For publishers, the module displays all relevant details of their published rides, including the origin, destination, timing, and a list of passengers. This allows publishers to manage their rides efficiently, with options to cancel rides, rating the ride upon completion, marking the ride as completed. For passengers, the module provides access to detailed information about the rides they are participating in, such as ride schedules, pickup points, and fellow passengers. It also includes features for rating the ride upon completion, contributing to the platform's feedback system, and marking the ride as completed.

8. User Profile Management Module

The User Profile Management Module empowers users to manage their personal information and customize their profiles within the RideShare platform. This module provides users with the ability to view and update their personal details, including name, contact information etc.

9. Admin Dashboard Module

The Admin Dashboard Module is a powerful tool for platform management, offering administrators a comprehensive interface to monitor and control all aspects of the RideShare system. Built on Django's robust admin interface, this module allows administrators to manage users, rides, ratings, and feedback effectively. The centralized

dashboard provides administrators with a bird's-eye view of platform operations, enabling efficient management and timely interventions to maintain the quality and security of the RideShare platform.

4.2 UML DIAGRAMS

USECASE DIAGRAM

This diagram captures the functional requirements of the RideShare system by illustrating the interactions between different types of users (actors) and the various use cases (functionalities) they can perform. It provides a high-level overview of the system's capabilities and the key interactions that occur.

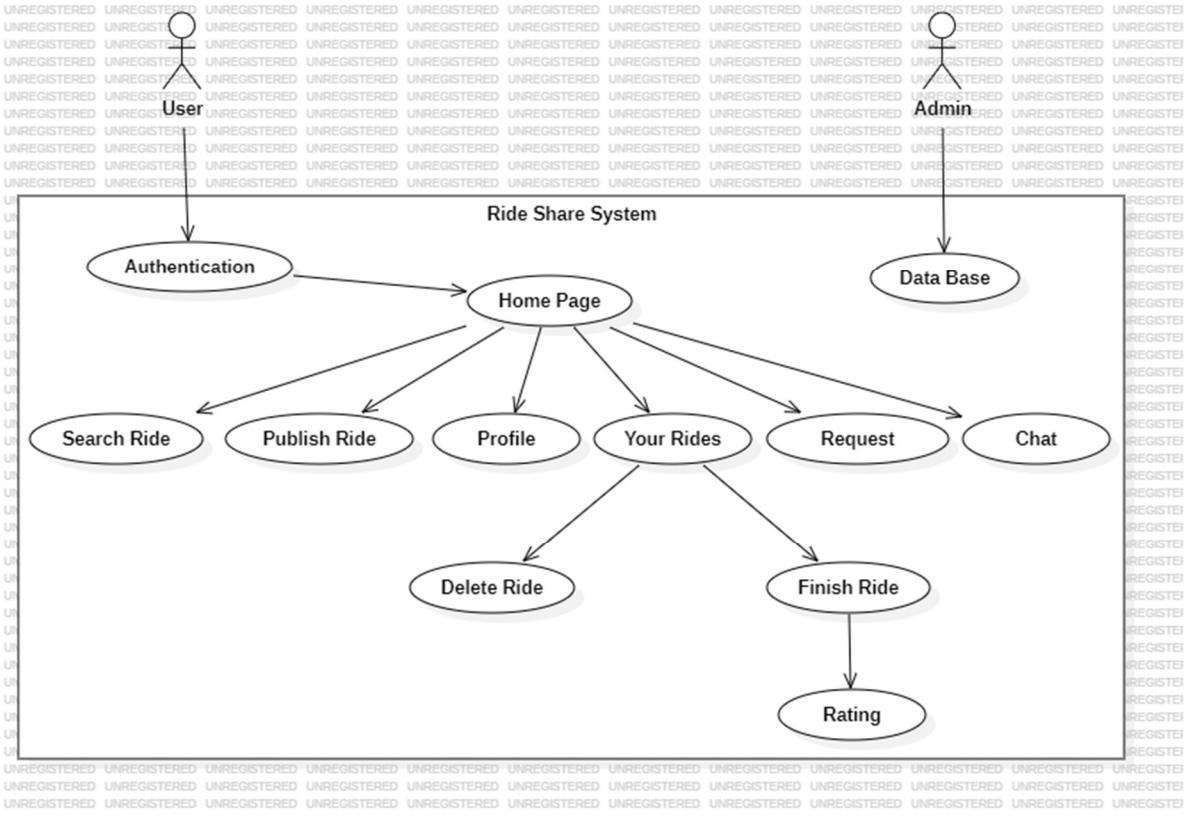

Fig 4.2.1 Usecase diagram

CLASS DIAGRAM

The class diagram presents the static structure of the system by detailing the system's classes, their attributes, methods, and the relationships between them. This diagram is essential for understanding the design of the RideShare system, showing how different entities (like Users, Rides, and Requests) are organized and connected

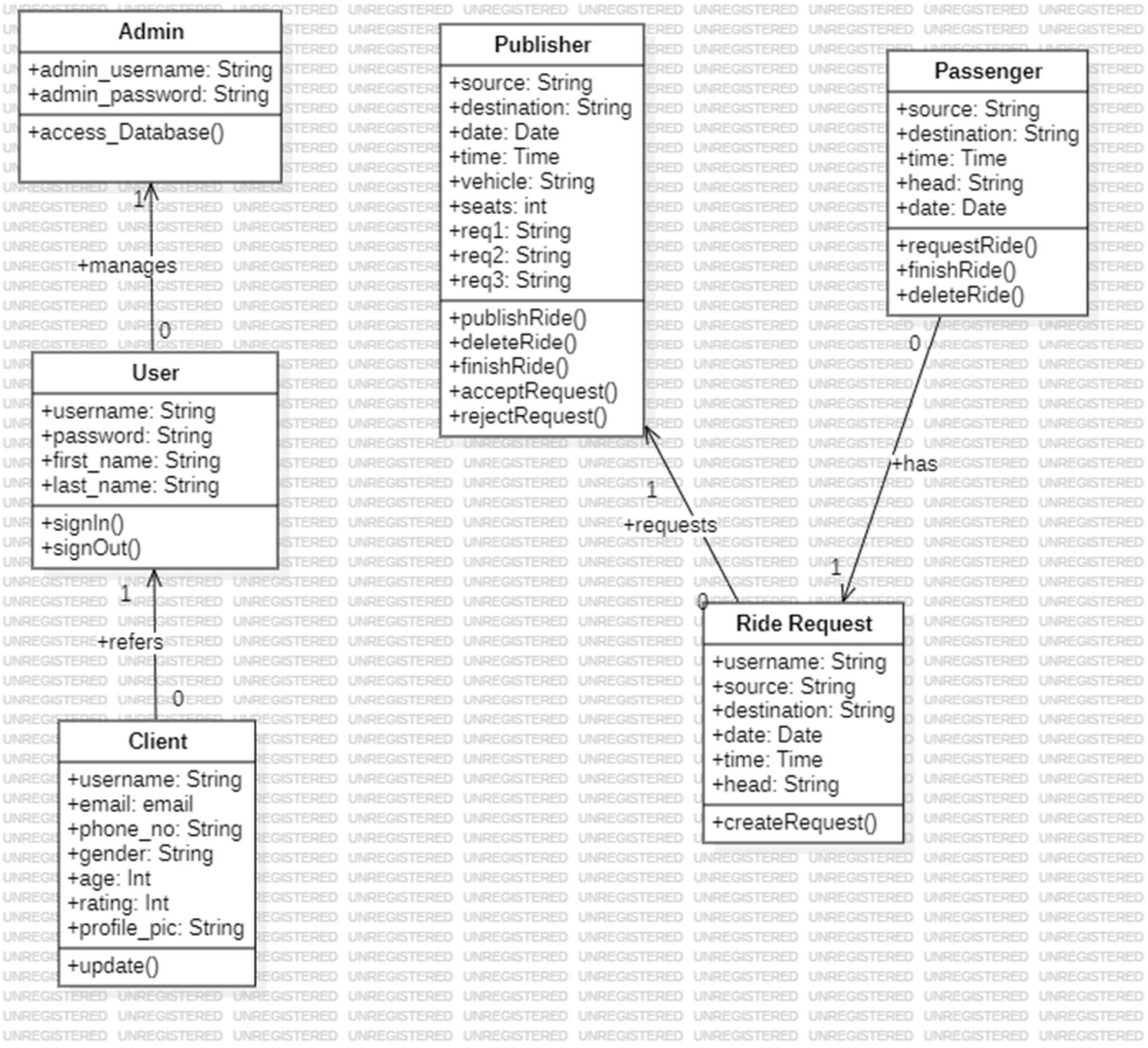

Fig 4.2.2 Class diagram

SEQUENCE DIAGRAM

The sequence diagram offers a dynamic view of the system, depicting the sequence of interactions between objects in a particular scenario or use case. It illustrates how messages are exchanged between objects over time, providing a clear understanding of the flow of operations in specific functionalities, such as publishing a ride or making a ride request.

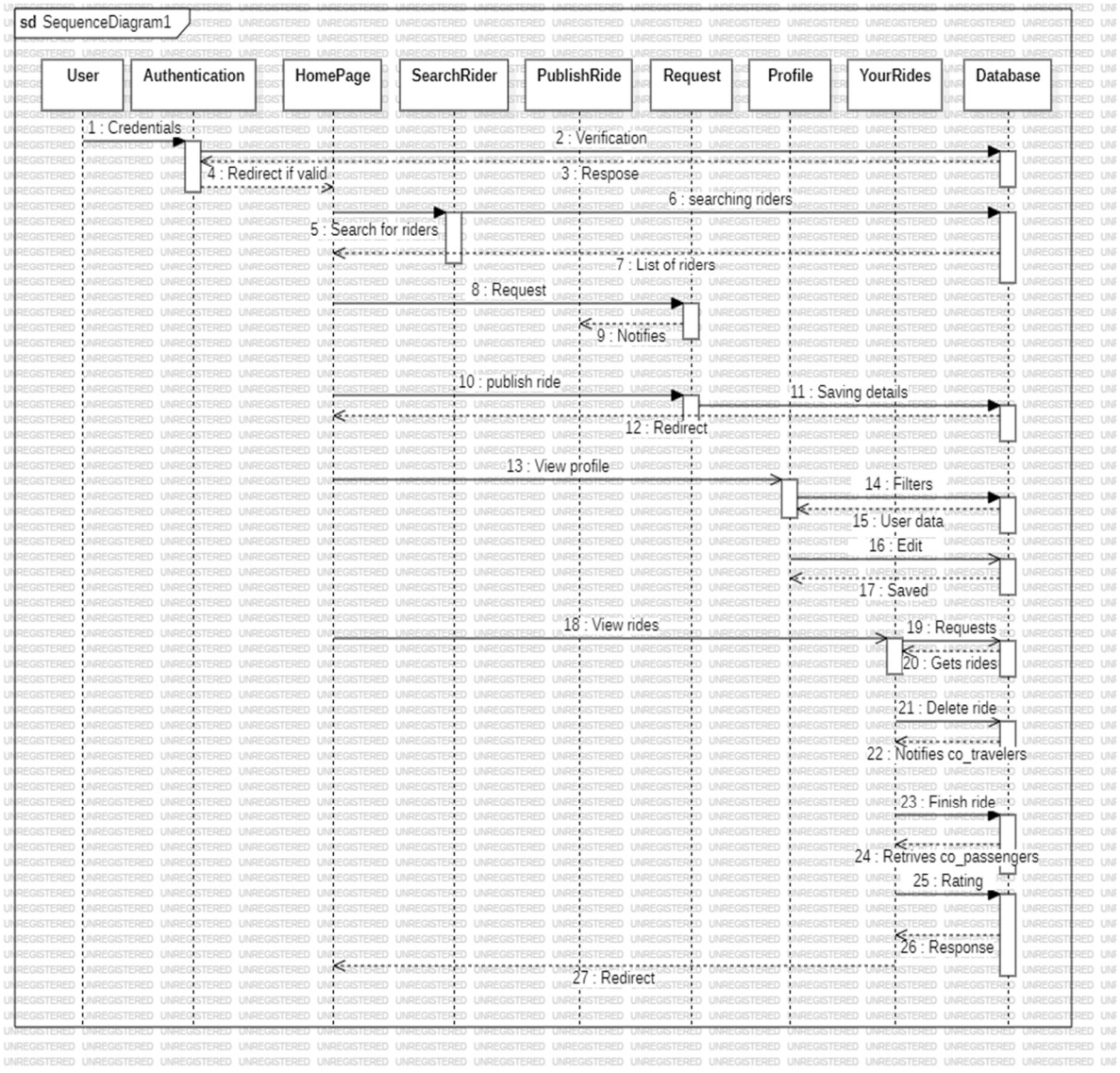

Fig 4.2.3 Sequence diagram

STATECHART DIAGRAM

The statechart diagram describes the different states an object within the system can be in and the transitions between those states. This diagram is particularly useful for understanding the lifecycle of key entities, such as a Ride or a Request, showing how they change states in response to different events.

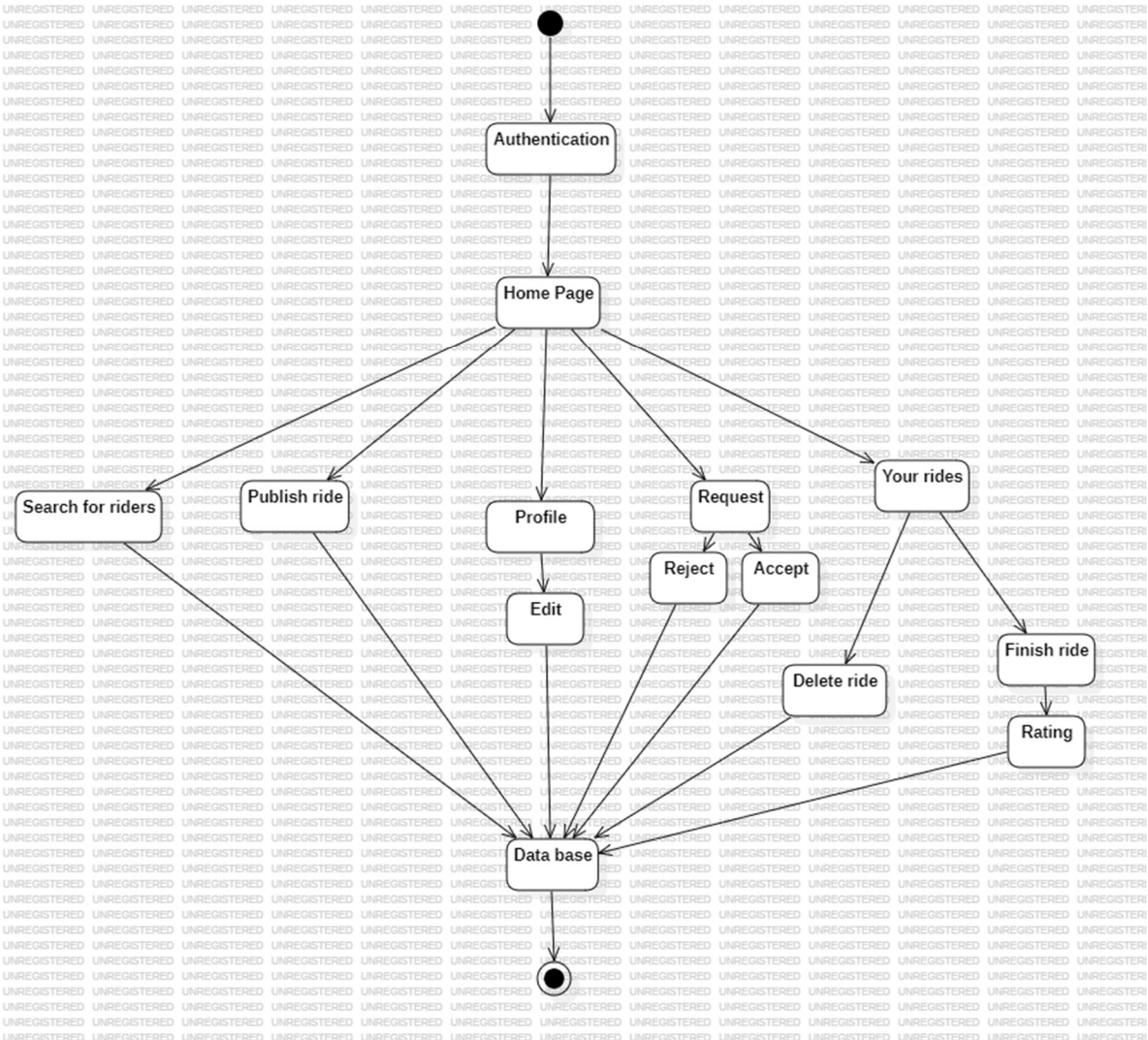

Fig 4.2.4 Statechart diagram

5. IMPLEMENTATION

5.1 IMPLEMENTATION

- **Set Up Development Environment:**
    - **Install Python and Django:** Ensure Python is installed on your system. Install Django using pip install django. Set up a new Django project using django-admin startproject RideShare.
    - **Install Required Libraries:** Use pip to install any additional libraries needed, such as PostgreSQL for the database.
- **Database Configuration:**
    - **Set Up PostgreSQL:** Install PostgreSQL and create a new database for the RideShare project. Configure the database settings in settings.py to connect your Django project to PostgreSQL.
    - **Create Models:** Define models for your application in models.py, such as PublisherTable, PassengerTable, ClientTable, and others as required. These models will represent the database structure.
- **Design and Implement User Authentication:**
    - **User Registration:** Create views and forms for user registration. Collect details like username, password, email, and additional personal information. Ensure that passwords are hashed using Django's built-in authentication system.
        - **User Login and Session Management:** Implement login functionality where users can authenticate themselves.
- **Implement Ride Management System:**
    - **Publish a Ride:** Develop a view where users can enter ride details such as source, destination, date, time, and vehicle information. Save this data to the database and display the ride as available for other users to join.

- **Search and Join Rides:** Implement search functionality where users can search for available rides based on their travel requirements. Allow users to send request for the ride and notify the publisher.
- **Manage "Your Rides":** Create a section where users can view their published rides and the rides they are participating in as passengers. Include options to "Finish Ride" and rate co-passengers and publishers.

- **Integrate Geolocation Services:**
  - **Mapbox API Integration:** Set up the Mapbox API to provide location suggestions for users when entering their source and destination. This will enhance the accuracy of location entries and improve the user experience.
- **Implement Notification System:**
  - **Email Notifications:** Configure Django to send emails to users for ride-related updates, such as confirmations, cancellations, or any other important information. Use the send_mail function in Django for this purpose.
- **Chat and Communication System**
  - **Chat Integration:** A chat feature was implemented to allow users to communicate with each other, particularly for coordinating ride details. AJAX technology was used to enable real-time messaging, allowing for seamless and instant communication between users without needing to reload the page. This approach ensures that the chat experience is smooth and responsive, enhancing the overall user experience in the RideShare application.
- **Rating and Feedback System**
  - **User Feedback:** After completing a ride, users can rate and provide feedback on other participants. Publishers can rate passengers, and passengers can rate both the publisher and other passengers. The ratings are stored in the database and displayed as follows
    - **For Passengers:** When passengers search for rides, the ratings of the publishers are shown alongside the available rides, helping them make informed decisions.

        - **For Publishers:** When passengers send a ride request, their ratings are shown to the publisher, allowing the publisher to assess the passengers before accepting or rejecting the request.
- **Profile Management:**
    - **User Profile:** Create a profile page where users can view and update their personal information, and see their ratings.
- **Frontend Development:**
    - **HTML, CSS, JavaScript:** Develop the frontend of the application using HTML, CSS, and JavaScript. Ensure that the design is user-friendly and responsive, primarily for desktop browsers.
    - **Templates in Django:** Use Django's templating system to render dynamic content on the frontend. Create templates for various pages such as home, login, registration, ride publishing, ride search, profile, chat, and Your Rides pages.

5.2 SAMPLE CODE

views.py

```
# Import necessary modules and models
from django.shortcuts import render, redirect
from django.contrib import messages
from django.contrib.auth import authenticate, login
from django.contrib.auth.decorators import login_required
from django.contrib.auth.models import User
from datetime import datetime
from .models import PublisherTable as Publisher
from .models import PassengerTable as Passenger
from .models import ClientTable as Client
from .models import PassengerDTable
from .models import PublisherDTable
from django.core.mail import send_mail
# Define a view function for the home page
def home(request):
    return render(request, 'home.html')
# Define a view function for the login page
def login_page(request):
    if request.method == "POST":
        username = request.POST.get('username')
        password = request.POST.get('password')
        user = authenticate(username=username, password=password)
        if user is not None:
            login(request, user)
```

```
            request.session['user_name'] = user.username
            request.session['user_id'] = user.id
            request.session['user_email'] = user.email
            return redirect('/mainpage/')
        else:
            messages.error(request, "Invalid Username or Password")
            return redirect('/login/')
    return render(request, 'login.html')
# Define a view function for the registration page
def calculate_age(birthdate):
    today = datetime.today()
     return today.year - birthdate.year - ((today.month, today.day) < (birthdate.month, birthdate.day))
def register_page(request):
    if request.method == 'POST':
        first_name = request.POST.get('first_name')
        last_name = request.POST.get('last_name')
        username = request.POST.get('username')
        password = request.POST.get('password')
        dob = request.POST.get('dateofbirth')
        gender = request.POST.get('gender')
        phone_number = request.POST.get('phonenumber')
        email = request.POST.get('email')
        e=str(email)
        #print(e)
        p=str(phone_number)
```

```
profile_pic = request.FILES.get('profile_pic')  # Use FILES to get file uploads
if User.objects.filter(username=username).exists():
    messages.info(request, "Username already taken!")
    return redirect('/register/')
# Parse the date of birth and calculate age
birthdate = datetime.strptime(dob, '%Y-%m-%d')
age = calculate_age(birthdate)
# Check if profile picture is uploaded; if not, use default
if not profile_pic:
    default_profile_pic = 'profile_pics/download_1TlHrl9.png'
else:
    default_profile_pic = profile_pic
# Create client profile
client = Client.objects.create(
    username=username,
    age=age,
    gender=gender,
    phone_number=p,
    email=e,
    profile_pic=default_profile_pic,
)
client.save()
user = User.objects.create_user(
    first_name=first_name,
    last_name=last_name,
    username=username
```

```
        )
        user.set_password(password)
        user.save()
        # Send welcome email
        subject = 'Welcome to RideShare!'
         message = f'Dear {first_name},\n\nWelcome to RideShare! We are excited to have you as a part of our community. ' \
                   f'Feel free to explore and enjoy our services.\n\nBest Regards,\nThe RideShare Team'
        from_email = settings.DEFAULT_FROM_EMAIL
        recipient_list = [email]
        try:
            send_mail(subject, message, from_email, recipient_list)
            messages.info(request, "Account created successfully! A welcome email has been sent.")
        except Exception as e:
            messages.warning(request, f"Account created successfully, but there was an error sending the welcome email: {e}")
        return redirect('/login/')
    return render(request, 'register.html')


# Custom decorator to check if the user is logged in
def login_required_decorator(view_func):
    def wrapper(request, *args, **kwargs):
        if 'user_name' not in request.session:
            messages.error(request, "You need to log in first.")
            return redirect('/login/')
        return view_func(request, *args, **kwargs)
```

```
    return wrapper
@login_required_decorator
def main_page(request):
    user_name = request.session.get('user_name')
    return render(request, 'mainpage.html', {'user_name': user_name})
@login_required_decorator
def publisher_page(request):
    user_name = request.session.get('user_name')
    #print(user_name)
    return render(request, 'publisher.html')
@login_required_decorator
def publisher_database(request):
    if request.method == "POST":
        username = request.session['user_name']
        c=Client.objects.get(username__iexact=username)
        phonenumber =c.phone_number
        source = request.POST.get("source")
        destination = request.POST.get("destination")
        date = request.POST.get("date")
        time = request.POST.get("time")
        seats = request.POST.get("seats")
        wheeler = request.POST.get("wheeler")
        vehicle = request.POST.get("vehicle")
        discription = request.POST.get("discription")
        pub = Publisher(
            username=username,
```

```
            phoneNumber=phonenumber,
            source=source,
            destination=destination,
            date=date,
            time=time,
            seats=seats,
            wheele=wheeler,
            vehicle=vehicle,
            discription=discription,
            requested_by="EMO"
        )
        pub.save()
        return redirect("/mainpage/")
@login_required_decorator
def request_page(request):
    return render(request, "Request.html")
def no_rides(request):
    return render(request, "no_rides.html")
@login_required_decorator
def search_for_publisher(request):
    name = request.session['user_name'].strip()
    if request.method == "POST":
        source = request.POST.get("source")
        destination = request.POST.get("destination")
        print(source,destination,"jijimm")
        pub = Publisher.objects.filter(source=source, destination=destination,seats__gt=0)
```

```
a=[]
if not pub:
    messages.error(request, "No publishers found")
    return render(request, "no_rides.html")
for p in pub:
    if not p.username==name:
        if p.req1!=name and p.req2!=name and p.req3!=name:
            b=[]
            name1=p.username
            c=Client.objects.filter(username__iexact=name1)
            for i in c:
                b.append(i.username)
                b.append(i.gender)
                b.append(i.age)
                b.append(i.rating)
            for req in [p.req1,p.req2,p.req3]:
                if  "None" in  req or "NONE" in req:
                    b.append("@")
                    b.append("@")
                    b.append("@")
                    b.append("@")
                else:
                    name1=req
                    c=Client.objects.filter(username__iexact=name1)
                    for i in c:
                        b.append(i.username)
```

```
                    b.append(i.gender)
                    b.append(i.age)
                    b.append(i.rating)
                b.append(p.source)
                b.append(p.destination)
                b.append(p.date)
                b.append(p.time)
                b.append(p.seats)
                b.append(p.wheele)
                b.append(p.vehicle)
                a.append(b)
        context = {'a': a}
        return render(request, "Request.html", context)
    return redirect("/mainpage/")
@login_required_decorator
def chat_page(request):
    name = request.session['user_name'].strip()
    pub = Publisher.objects.filter(username__iexact=name)
    #print("Queryset:", pub)
    if pub.exists():
        users = []
        for publisher in pub:
            s = int(publisher.seats)
            if s > 0:
                for req in [publisher.req1, publisher.req2, publisher.req3]:
                    if "None" in req:
```

```
                a = req.replace("None", "")
                c=Client.objects.filter(username__iexact=a)
                b=[]
                for i in c:
                    b.append(i.username)
                    b.append(i.rating)
                    b.append(i.gender)
                    b.append(i.age)
                b.append(publisher.date)
                b.append(publisher.time)
                users.append(b)
        #print(users)
        return render(request, 'chat.html', {'users': users})
    else:
        messages.error(request, "Publisher not found")
        return redirect("/chat/")
@login_required_decorator
def sendingrequest_page(request):
    if request.method == "POST":
        name = request.session['user_name']
        publishername = request.POST.get("publishername").strip()
        pub = Publisher.objects.filter(username__iexact=publishername)
        if pub.exists():
            for publisher in pub:
                s = int(publisher.seats)
                if publisher.req1 == "NONE" and s > 0:
```

```
            publisher.req1 = name + "None"
        elif publisher.req2 == "NONE" and s > 0:
            publisher.req2 = name + "None"
        elif publisher.req3 == "NONE" and s > 0:
            publisher.req3 = name + "None"
        else:
            messages.error(request, "Seats are full")
            return redirect("/mainpage/")
        publisher.save()
        # Send email notification to publisher
        try:
            client = Client.objects.get(username__iexact=publishername)
            if client.email:
                subject = 'New Ride Request'
                message = f'Hello {publishername},\n\nYou have received a new ride request from {name}.'
                send_mail(subject, message, settings.DEFAULT_FROM_EMAIL, [client.email])
                messages.info(request, "Request sent successfully and notification email sent to the publisher.")
            else:
                messages.info(request, "Request sent successfully, but no email found for the publisher to notify.")
        except Client.DoesNotExist:
            messages.error(request, "Request sent successfully, but publisher email not found.")
        except Exception as e:
            messages.error(request, f"Error sending email: {e}")
```

```
            return redirect("/mainpage/")
        else:
            messages.error(request, "Something went wrong")
            return redirect("/mainpage/")
    else:
        messages.error(request, "Something went wrong")
        return redirect("/mainpage/")
from django.conf import settings
@login_required_decorator
def acceptrequest_page(request):
    if request.method == "POST":
        name = request.session['user_name'].strip()
        date=request.POST.get("date")
        time=request.POST.get("time")
        username = request.POST.get("username").strip()
        print(date,"datuu ")
        clean_time = time.replace('.', '').lower()
        date_iso = time.replace('.', '').lower()
        # Clean up the time string
        clean_time = time.replace('.', '').lower()

            # Strip leading/trailing spaces and remove the dot in the month abbreviation
        date_iso = date.strip().replace('.', '')
            # Handle time conversion
        if clean_time == "midnight":
            clean_time = "00:00"
```

```
else:
    try:
        clean_time = datetime.strptime(clean_time, '%I:%M %p').strftime('%H:%M')
    except ValueError as e:
        print(f"Error parsing time: {e}")
        messages.error(request, "Invalid time format.")
        return
    # Handle date conversion
try:
    date_iso = datetime.strptime(date_iso, '%b %d, %Y').strftime('%Y-%m-%d')
except ValueError as e:
    print(f"Error parsing date: {e}")
    messages.error(request, "Invalid date format.")
    return
t = str(clean_time)
d = str(date_iso)
print("time:", t, "date:", d)
pub= Publisher.objects.filter(username__iexact=name,date=d)
for publisher in pub:
        dd=str(publisher.date)
        if d in dd:
            print(d,dd)
            s = int(publisher.seats)
            if s > 0:
                if username + "None" in publisher.req1:
                    publisher.req1 = username
```

```
                s -= 1
            elif username + "None" in publisher.req2:
                publisher.req2 = username
                s -= 1
            elif username + "None" in publisher.req3:
                publisher.req3 = username
                s -= 1
            publisher.seats = str(s)
            publisher.save()
            break
        else:
            messages.error(request, "No available seats")
            return redirect("/chat/")

    # Save the accepted passenger information
pas = Passenger(
        username1=username,
        username2=name,
        source=publisher.source,
        destination=publisher.destination,
        date=publisher.date,
        time=publisher.time,
    )
pas.save()
# Send email notification
try:
```

```
            client = Client.objects.get(username__iexact=username)
            if client.email:
                subject = 'Ride Request Accepted'
                    message = f'Hello {username},\n\nYour ride request has been accepted by {name}. \n\n Wanna chat with you co-passengers and publisher here is the room id {name} '
                send_mail(subject, message, settings.DEFAULT_FROM_EMAIL, [client.email])
                messages.info(request, f"Accepted {username} and notification email sent.")
            else:
                messages.info(request, f"Accepted {username}, but no email found to notify.")
        except Client.DoesNotExist:
            messages.error(request, f"Accepted {username}, but user email not found.")
        except Exception as e:
            messages.error(request, f"Error sending email: {e}")
        return redirect("/chat/")
    else:
        messages.error(request, "Something went wrong")
        return redirect("/chat/")



@login_required_decorator
def rejectrequest_page(request):
    if request.method == "POST":
        name = request.session['user_name'].strip()
        pub = Publisher.objects.filter(username__iexact=name)
```

```
        username = request.POST.get("username").strip()

        request_rejected = False
        for publisher in pub:
            if username + "None" in publisher.req1:
                publisher.req1 = "NONE"
                request_rejected = True
            elif username + "None" in publisher.req2:
                publisher.req2 = "NONE"
                request_rejected = True
            elif username + "None" in publisher.req3:
                publisher.req3 = "NONE"
                request_rejected = True
            publisher.save()

        if request_rejected:
            # Send email notification
            try:
                client = Client.objects.get(username__iexact=username)
                if client.email:
                    subject = 'Ride Request Rejected'
                        message = f'Hello {username},\n\nYour ride request has been rejected by {name}.'
                            send_mail(subject, message, settings.DEFAULT_FROM_EMAIL, [client.email])
                    messages.info(request, f"Rejected {username} and notification email sent.")
                else:
```

```
                messages.info(request, f"Rejected {username}, but no email found to notify.")
            except Client.DoesNotExist:
                messages.error(request, f"Rejected {username}, but user email not found.")
            except Exception as e:
                messages.error(request, f"Error sending email: {e}")
        else:
            messages.info(request, f"No pending request from {username} found to reject.")


        return redirect("/mainpage/")
    else:
        messages.error(request, "Something went wrong")
        return redirect("/mainpage/")
@login_required_decorator
def your_rides(request):
    name = request.session['user_name'].strip()
    source = request.POST.get('source')
    destination = request.POST.get('destination')
    time = request.POST.get('time')
    date=request.POST.get("date")
    print(source,destination,date)
    # Fetch unique rides where the current user is the publisher
    publishers = Publisher.objects.filter(username__iexact=name).distinct()
    pub_details = []
    for publisher in publishers:
        p = [name]
        for req in [publisher.req1, publisher.req2, publisher.req3]:
```

```
            if not "none" in req.lower():
                p.append(req)
            ride_info = (publisher.source, publisher.destination, publisher.date, publisher.time, publisher.vehicle, p)
            if ride_info not in pub_details:  # Check if ride info is already added
                pub_details.append(ride_info)
    # Fetch unique rides where the current user is the passenger
    passengers = Passenger.objects.filter(username1__iexact=name).distinct()
    print(passengers)
    pass_details = []
    for passenger in passengers:
                                                    associated_publishers = Publisher.objects.filter(username__iexact=passenger.username2,source=passenger.source,destination=passenger.destination,date=passenger.date)
        print(associated_publishers)
        for publisher in associated_publishers:
            pp = [passenger.username2]
            for req in [publisher.req1, publisher.req2, publisher.req3]:
                if not "none" in req.lower():
                    pp.append(req)
            ride_info = (publisher.source, publisher.destination, publisher.date, publisher.time, publisher.vehicle, pp)
            if ride_info not in pass_details:  # Check if ride info is already added
                pass_details.append(ride_info)
    context = {
        'pub_details': pub_details,
        'pass_details': pass_details,
    }
```

```
    print("pass pun deb",pub_details)
    return render(request, 'your_rides.html', context)
from django.contrib.auth.models import User
from django.db.models import TextField
@login_required_decorator
def profile_page(request):
    try:
        name = request.session['user_name'].strip()
        u= User.objects.get(username__iexact=name)
        user_info = []
        user_info.append(u.username)
        user_info.append(u.first_name)
        user_info.append(u.last_name)
        u= Client.objects.get(username__iexact=name)
        user_info.append(u.email)
        user_info.append(u.phone_number)
        user_info.append(u.age)
        user_info.append(u.gender)
        user_info.append(u.rating)
        user_info.append(u.profile_pic)
        #print(u.profile_pic)
    except User.DoesNotExist:
        messages.error(request, "Profile not found.")
        return redirect("/mainpage/")
    context = {
        'user_info': user_info,
```

```
    }
    return render(request, 'profile.html', context)
@login_required_decorator
def edit_page(request):
    user = request.user
    client_data = Client.objects.get(username=user.username)
    context = {
        'user': client_data,
        'first_name': user.first_name,  # Accessing first name from the User model
        'last_name': user.last_name,
    }
    return render(request, 'edit.html', context)
@login_required_decorator
def edited_page(request):
    try:
        name = request.session['user_name'].strip()
        u= User.objects.get(username__iexact=name)
        c=Client.objects.get(username__iexact=name)
        username = request.POST.get("username")
        password = request.POST.get("password")
        first_name = request.POST.get("first_name")
        last_name = request.POST.get("last_name")
        email = request.POST.get("email")
        phonenumber = request.POST.get("phonenumber")
        p=str(phonenumber)
        age = request.POST.get("age")
```

```
gender = request.POST.get("gender")
profile_pic= request.POST.get("profile_pic")
if username:
    # if User.objects.filter(username=username).exists():
    #     messages.info(request, "Username already taken!")
    #     return redirect('/edit/')
    u.username=username
    c.username=username
if first_name:
    u.first_name=first_name
if last_name:
    u.last_name=last_name
if email:
    c.email=email
if phonenumber:
    c.phone_number=p
if age:
    c.age=age
if gender:
    c.gender=gender
if profile_pic:
    c.profile_pic=profile_pic
if password:
    u.password=password
u.save()
c.save()
```

```
        messages.info(request, "succesfull")
        return redirect('/profile/')
    except:
        messages.error(request, "somthing went wrong")
        return redirect('/login/')
from django.shortcuts import render, redirect, get_object_or_404
from django.contrib import messages
from django.views.decorators.http import require_POST
from datetime import datetime
from django.shortcuts import render, redirect, get_object_or_404
from django.contrib import messages
from django.views.decorators.http import require_POST
from datetime import datetime
def delete_published_ride(request):
    name = request.session['user_name'].strip()
    source = request.POST.get('source')
    destination = request.POST.get('destination')
    date = request.POST.get('date')
    time = request.POST.get('time')

        # Clean up the time string
    clean_time = time.replace('.', '').lower()

        # Strip leading/trailing spaces and remove the dot in the month abbreviation
    date_iso = date.strip().replace('.', '')
        # Handle time conversion
```

```
    if clean_time == "midnight":
        clean_time = "00:00"
    else:
        try:
            clean_time = datetime.strptime(clean_time, '%I:%M %p').strftime('%H:%M')
        except ValueError as e:
            print(f"Error parsing time: {e}")
            messages.error(request, "Invalid time format.")
            return
        # Handle date conversion
    try:
        date_iso = datetime.strptime(date_iso, '%b %d, %Y').strftime('%Y-%m-%d')
    except ValueError as e:
        print(f"Error parsing date: {e}")
        messages.error(request, "Invalid date format.")
        return
    t = str(clean_time)
    d = str(date_iso)
    print("time:", t, "date:", d)
                                                                                pub=
Publisher.objects.filter(username__iexact=name,source=source,destination=destination,dat
e=d,time=t)
    print(pub)
    a=[]
    for p in pub:
        tt=str(p.time)
        dd=str(p.date)
```

```
        #print(p.source,source,p.time,tt,p.date,dd)
        if t in tt and d in dd and source==p.source:
            for req in [p.req1,p.req2,p.req3]:
                if req!="NONE" and not "None" in req:
                    c= Client.objects.filter(username__iexact=req)
                    for i in c:
                        a.append(i.email)
            p.delete()
    notify_copassengers(name,a,source,destination,date,time)
    return redirect("/your_rides/")
from django.conf import settings
def notify_copassengers(username,sendMail,source,destination,date,time):
    if sendMail:
        subject = 'Ride Cancellation Notification'
         message = f'The ride from {source} to {destination} on {date} at {time} has been canceled by the {username}'
        send_mail(subject, message, settings.DEFAULT_FROM_EMAIL, list(sendMail))
def delete_passenger_ride(request):
    name = request.session['user_name'].strip()
    source = request.POST.get('source')
    time = request.POST.get('time')
    date=request.POST.get("date")
    head=request.POST.get("head")
    destination=request.POST.get("destination")
        # Clean up the time string
    clean_time = time.replace('.', '').lower()
```

```
    # Strip leading/trailing spaces and remove the dot in the month abbreviation
    date_iso = date.strip().replace('.', '')
    # Handle time conversion
    if clean_time == "midnight":
        clean_time = "00:00"
    else:
        try:
            clean_time = datetime.strptime(clean_time, '%I:%M %p').strftime('%H:%M')
        except ValueError as e:
            print(f"Error parsing time: {e}")
            messages.error(request, "Invalid time format.")
            return
    # Handle date conversion
    try:
        date_iso = datetime.strptime(date_iso, '%b %d, %Y').strftime('%Y-%m-%d')
    except ValueError as e:
        print(f"Error parsing date: {e}")
        messages.error(request, "Invalid date format.")
        return
    t = str(clean_time)
    d = str(date_iso)
    print("time:", t, "date:", d)
                                                                                pub=
Publisher.objects.filter(username__iexact=head,source=source,destination=destination,date
=d)
    for p in pub:
```

```
tt=str(p.time)
dd=str(p.date)
#print(p.source,source,p.time,tt,p.date,dd)
if t in tt and d in dd and source==p.source:
    if p.req1==name:
        pasd=PassengerDTable.objects.create(
        username2=head,
        username1=name,
        )
        pasd.save()
        #print(p.req1)
        p.req1="NONE"
        p.seats=str(int(p.seats)+1)
        p.save()
        #print(p.req1)
        #print("DONE")
    if p.req2==name:
        pasd=PassengerDTable.objects.create(
        username2=head,
        username1=name,
        )
        pasd.save()
        p.req2="NONE"
        p.seats=str(int(p.seats)+1)
        p.save()
    if p.req3==name:
```

```
            pasd=PassengerDTable.objects.create(
            username2=head,
            username1=name,
            )
            pasd.save()
            p.req3="NONE"
            p.seats=str(int(p.seats)+1)
            p.save()
                                                                                pas=
Passenger.objects.filter(username1=name,username2=head,source=source,destination=dest
ination,date=d)
    for p in pas:
        p.delete()
    c=Client.objects.filter(username__iexact=head)
    name1=""
    for i in c:
        name1=i.email
    a=[name1]
    notify_copassengers2(name,a)
    return redirect("your_rides")
def notify_copassengers2(username,sendMail):
    if sendMail:
        subject = 'Ride Cancellation Notification'
        message = f'your co passenger {username} has canceled the ride.'
        send_mail(subject, message, settings.DEFAULT_FROM_EMAIL, list(sendMail))
from datetime import datetime
from django.shortcuts import render, redirect
```

```
from django.contrib import messages
from .models import PublisherTable, PassengerTable, ClientTable
def thank_you(request):
    return render(request, 'thank_you.html')


from datetime import datetime
from django.contrib import messages
from django.contrib.auth.decorators import login_required
@login_required
def finish_ride(request):
    if request.method == 'POST':
        source = request.POST.get('source')
        destination = request.POST.get('destination')
        time = request.POST.get('time')
        date = request.POST.get('date')
        head = request.POST.get('head')
        print("headddddddddddddddd:",head)
        # Clean up the time string
        clean_time = time.replace('.', '').lower()

        # Strip leading/trailing spaces and remove the dot in the month abbreviation
        date_iso = date.strip().replace('.', '')
        # Handle time conversion
        if clean_time == "midnight":
            clean_time = "00:00"
```

```
        else:
            try:
                clean_time = datetime.strptime(clean_time, '%I:%M %p').strftime('%H:%M')
            except ValueError as e:
                print(f"Error parsing time: {e}")
                messages.error(request, "Invalid time format.")
                return
        # Handle date conversion
        try:
            date_iso = datetime.strptime(date_iso, '%b %d, %Y').strftime('%Y-%m-%d')
        except ValueError as e:
            print(f"Error parsing date: {e}")
            messages.error(request, "Invalid date format.")
            return
        t = str(clean_time)
        d = str(date_iso)
        print("time:", t, "date:", d)
        user_name = request.session['user_name'].strip()
        co_travelers = []
        publisher_name = None
        #print("pubrides")
        try:
            pub_rides = PublisherTable.objects.filter(source=source, destination=destination,
date=d, time=t,username=head)
            ## pass_rides = PassengerTable.objects.filter(username1=user_name)
            ## print("pub",pub_rides,"pass",pass_rides)
```

```
        ## if pub_rides.exists() and len(pass_rides)==0:
          ##   return redirect('thank_you')
        ## if pass_rides.exists() and pass_rides.filter(finished=True).exists():
          ##   return redirect('thank_you')
         for pub_ride in pub_rides:
            if pub_ride.username == user_name:  # User is the publisher
                        co_travelers.extend([req for req in [pub_ride.req1, pub_ride.req2, pub_ride.req3] if req and req != "NONE" and req != user_name])
            else:  # User is a passenger
               publisher_name = pub_ride.username
               co_travelers.extend([publisher_name])
                        co_travelers.extend([req for req in [pub_ride.req1, pub_ride.req2, pub_ride.req3] if req and req != "NONE" and req != user_name])
      except PublisherTable.DoesNotExist:
         pub_rides = None


      print(f"Co-travelers: {co_travelers}")
          pub_rides = PublisherTable.objects.filter(source=source, destination=destination, date=d, time=t,username=head)
      for pub_ride in pub_rides:
         x=pub_ride.discription
         if pub_ride.discription==None:
            pub_ride.discription=""
            pub_ride.save()
         print("Discription:",pub_ride.discription,"user:",user_name)
         if user_name in pub_ride.discription:
            return redirect('thank_you')
```

```
            else:
                pub_ride.discription=pub_ride.discription+" "+user_name
                pub_ride.save()
                break
        if not co_travelers:
            messages.error(request, "No co-travelers found for this ride.")
            return redirect('thank_you')
        travelers = ClientTable.objects.filter(username__in=co_travelers)
        if publisher_name and publisher_name != user_name:
            travelers |= ClientTable.objects.filter(username=publisher_name)
        print(f"Travelers: {travelers}")
        ride_details = f"{source}|{destination}|{date_iso}|{clean_time}"
            return render(request, 'rate_ride.html', {'travelers': travelers, 'ride_details': ride_details})
    else:
        return redirect('rate_ride.html')
@login_required
def submit_rating(request):
    if request.method == 'POST':
        ride_details = request.POST.get('ride_details')
        if not ride_details:
            messages.error(request, "No ride details provided.")
            return redirect('your_rides')
        source, destination, date, time = ride_details.split('|')
        user_name = request.session['user_name'].strip()
            pub_ride = PublisherTable.objects.filter(source=source, destination=destination, date=date, time=time, username=user_name).first()
```

```
    pass_ride = PassengerTable.objects.filter(username1=user_name).first()
    if pub_ride:
        pub_ride.finished = True
        pub_ride.save()
    elif pass_ride:
        pass_ride.finished = True
        pass_ride.save()
    for key, value in request.POST.items():
        if key.startswith('rating_'):
            username = key.split('rating_')[1]
            rating = int(value)
            try:
                client = ClientTable.objects.get(username=username)
                client.sumofrating += rating
                client.ppl += 1
                client.rating = client.sumofrating / client.ppl
                client.save()
            except ClientTable.DoesNotExist:
                messages.error(request, f"Client {username} does not exist.")
                continue
    messages.success(request, 'Ratings submitted successfully.')
    return redirect('thank_you')
else:
    return redirect('your_rides')
```

models.py

```
from django.db import models
from django.contrib.auth.models import User
from datetime import datetime
# Create your models here.
class Room(models.Model):
    name = models.CharField(max_length=1000)
class Message(models.Model):
    value = models.CharField(max_length=1000000)
    date = models.DateTimeField(default=datetime.now, blank=True)
    user = models.CharField(max_length=1000000)
    room = models.CharField(max_length=1000000)
```

views.py

```
from django.shortcuts import render, redirect, get_object_or_404
from chat.models import Room, Message
from django.http import HttpResponse, JsonResponse
# Create your views here.
def chathome(request):
    return render(request, 'chathome.html')
def room(request, room):
    username = request.GET.get('username')
    room_details = get_object_or_404(Room, name=room)
    return render(request, 'room.html', {
        'username': username,
        'room': room,
        'room_details': room_details
```

```
    })
def checkview(request):
    room = request.POST['room_name']
    username = request.POST['username']
    if Room.objects.filter(name=room).exists():
        return redirect('/'+room+'/?username='+username)
    else:
        new_room = Room.objects.create(name=room)
        new_room.save()
        return redirect('/'+room+'/?username='+username)
def send(request):
    message = request.POST['message']
    username = request.POST['username']
    room_id = request.POST['room_id']
    new_message = Message.objects.create(value=message, user=username, room=room_id)
    new_message.save()
    return HttpResponse('Message sent successfully')
def getMessages(request, room):
    room_details = Room.objects.get(name=room)
    messages = Message.objects.filter(room=room_details.id)
    return JsonResponse({"messages":list(messages.values())})
```

urls.py

```
from django.contrib import admin
from django.urls import path, include
from authentication.views import *
from django.conf import settings
```

```
from django.contrib.staticfiles.urls import staticfiles_urlpatterns
from django.conf.urls.static import static
from chat.views import *
urlpatterns = [
    path('', home, name="recipes"),  # Home page
    path("admin/", admin.site.urls),  # Admin interface
    path('login/', login_page, name='login_page'),  # Login page
    #path('logout/', logout, name='logout'),
    path('register/', register_page, name='register'),  # Registration page
    path('mainpage/', main_page, name='mainpage'),  # Main page
    path("publisher/", publisher_page, name="publisher"),
    path("publisherdatabase", publisher_database, name="publisherdatabase"),
    path("searchforpublisher", search_for_publisher, name="searchforpublisher"),
    path("request/", request_page, name="request"),
    path("no_rides/", no_rides, name="no_rides"),
    path("chat/", chat_page, name="chat"),
    path("sendingrequest", sendingrequest_page, name="sendingrequest"),
    path("rejectrequest", rejectrequest_page, name="rejectrequest"),
    path("acceptrequest", acceptrequest_page, name="acceptrequest"),
    path("your_rides/", your_rides, name="your_rides"),
    path("profile/", profile_page, name="profile"),
    path("edit/", edit_page, name="edit"),
    path("edited/", edited_page, name="edited"),
    path('delete_published_ride/', delete_published_ride, name='delete_published_ride'),
    path('delete_passenger_ride/', delete_passenger_ride, name='delete_passenger_ride'),
    path('finish_ride/', finish_ride, name='finish_ride'),
```

```
    path('submit_rating/', submit_rating, name='submit_rating'),
    path('thank_you/', thank_you, name='thank_you'),
    path('chat_home/', chathome, name='chat_home'),
    path('submit_rating/', submit_rating, name='submit_rating'),
    path('chat_home/checkview', checkview, name='checkview'),
    path('send', send, name='send'),
    path('getMessages/<str:room>/', getMessages, name='getMessages'),
    path('<str:room>/', room, name='room'),
]
if settings.DEBUG:
    urlpatterns += static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
urlpatterns += staticfiles_urlpatterns()
```

6. TESTING

6.1 TESTING

6.1.1 Unit Testing

Unit testing involves verifying individual components of the RideShare project to ensure they function as expected. For user registration, the test checks if new users can register successfully with valid details. User login tests ensure that registered users can log in using correct credentials. The publish a ride test confirms that users can publish rides with accurate details. Search functionality is validated to ensure it returns the correct rides based on user input. The test for sending a ride request checks if users can request to join a published ride. Rating and feedback functionality is tested to ensure users can rate and provide feedback on completed rides. The chat with publisher test verifies that users can send and receive messages in the chat feature. Finally, the logout test ensures that users can log out and are redirected to the login page.

6.1.2 Integration Testing

Integration testing assesses how different components of the RideShare project work together. The user authentication and ride management integration test ensures that user authentication properly integrates with the ride management system, allowing users to publish and manage rides post-authentication. The Mapbox API integration test checks that the API correctly shows location suggestions, enhancing the user experience. The chat functionality integration test ensures that real-time messaging works as intended across the platform. Additionally, database integration is tested to confirm that data is accurately saved and retrieved from the PostgreSQL database.

6.1.3 Functional Testing

Functional testing focuses on validating the core features of the RideShare application. The publish ride functionality test ensures that published rides appear correctly in the list of available rides, and users can view and interact with these rides. The ride request functionality test verifies that requests to join rides are successfully sent by users and received by ride publishers. User rating functionality is tested to confirm that ratings are properly submitted, stored, and displayed on user profiles and ride listings. Additionally, functional testing includes verifying that all interactive elements, such as forms and buttons, perform as expected and meet user requirements.

6.1.4 Security Testing

Security testing examines the measures in place to protect the RideShare application from potential threats. Authentication security testing ensures that the login and registration mechanisms restrict access to authorized users only, preventing unauthorized access. Data protection testing verifies that sensitive user data, including passwords is encrypted and securely stored to protect against unauthorized access.

6.2 TEST CASES

Test Case ID	Test Case Description	Steps to Execute	Expected Output	Test Result
TC1	User Registration	1. Navigate to registration page. 2. Enter valid user details. 3. Submit form.	User is registered successfully, and user record is created in the database.	Success
TC2	User Login	1. Navigate to login page. 2. Enter valid credentials. 3. Submit form.	User is logged in and redirected to the home page.	Success
TC3	Publish a Ride	1. Log in. 2. Go to "Publish Ride" section. 3. Enter ride details. 4. Submit.	Ride is published and visible in the list of available rides when searched by a user.	Success
TC4	Search for Rides	1. Log in. 2. Enter search criteria ("From" and "To"). 3. Submit search.	List of rides matching the search criteria is displayed.	Success

Test Case ID	Test Case Description	Steps to Execute	Expected Output	Test Result
TC5	Send a Ride Request	1. Log in. 2. Select a published ride. 3. Send a request to join the ride. 4. Verify email notification.	Request is sent, and publisher receives an email notification about the new request.	Success
TC6	Accept a Ride Request	1. Log in as a publisher. 2. Go to "Requests" section. 3. Accept a ride request.	Request is accepted, and the passenger is notified.	Success
TC7	Reject a Ride Request	1. Log in as a publisher. 2. Go to "Requests" section. 3. Reject a ride request.	Request is rejected, and the passenger is notified.	Success

Test Case ID	Test Case Description	Steps to Execute	Expected Output	Test Result
TC8	Rate a Ride	1. Go to "Your Rides". 2. Select a completed ride. 3. Enter rating. 4. Submit.	Rating and review are submitted and displayed on user profile, in available rides, and in the request page when publisher gets a request from a passenger.	Success
TC9	Chat with Users	1. Log in. 2. Initiate chat with another user (publisher or passenger) using room code. 3. Send and receive messages.	Messages are sent and received in real-time with the other user.	Success
TC10	Display Published Rides	1. Log in. 2. Navigate to "Your Rides" section. 3. Verify that published rides are listed.	All published rides appear correctly in the "Your Rides" section.	Success

Test Case ID	Test Case Description	Steps to Execute	Expected Output	Test Result
TC11	Display Traveling Rides	1. Log in. 2. Go to "Your Rides" section. 3. Verify that traveling rides are listed.	All rides the user is currently traveling in appear correctly.	Success

7. OUTPUT SCREENS

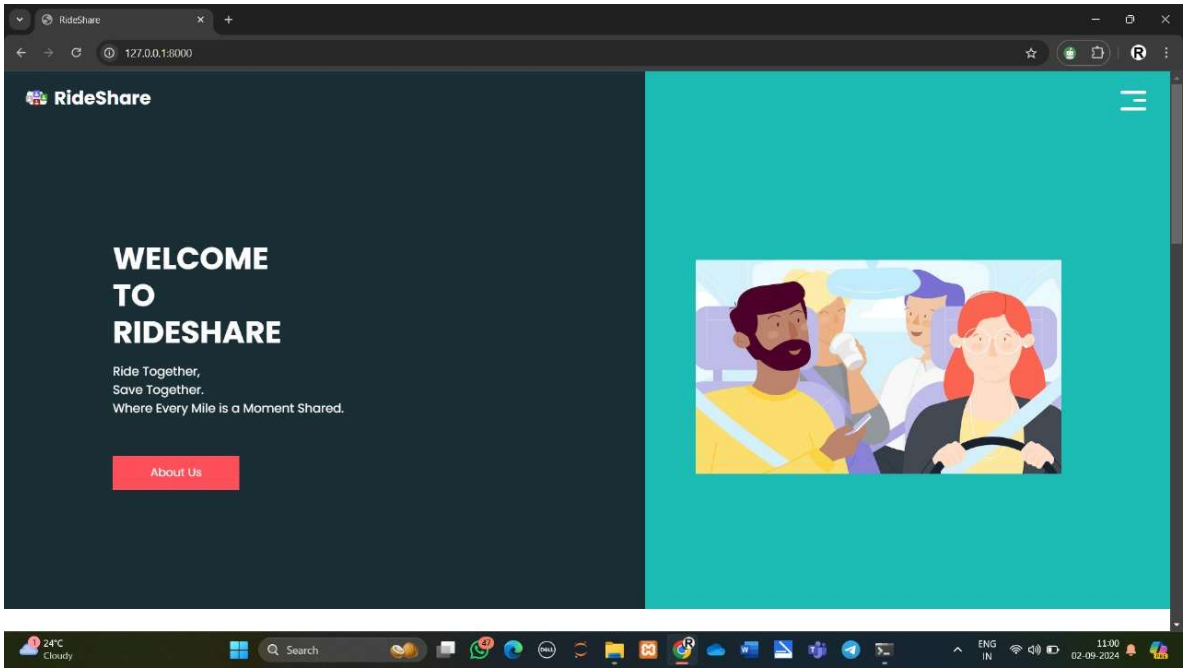

Fig 7.1 Welcome Page

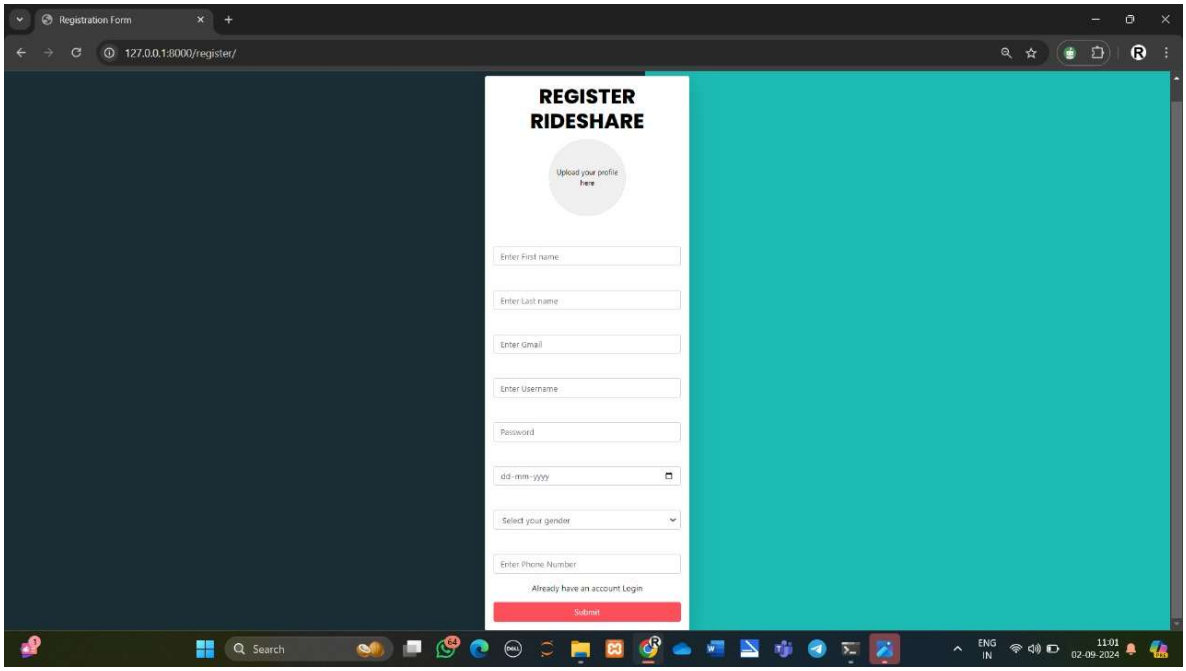

Fig 7.2 Registration Page

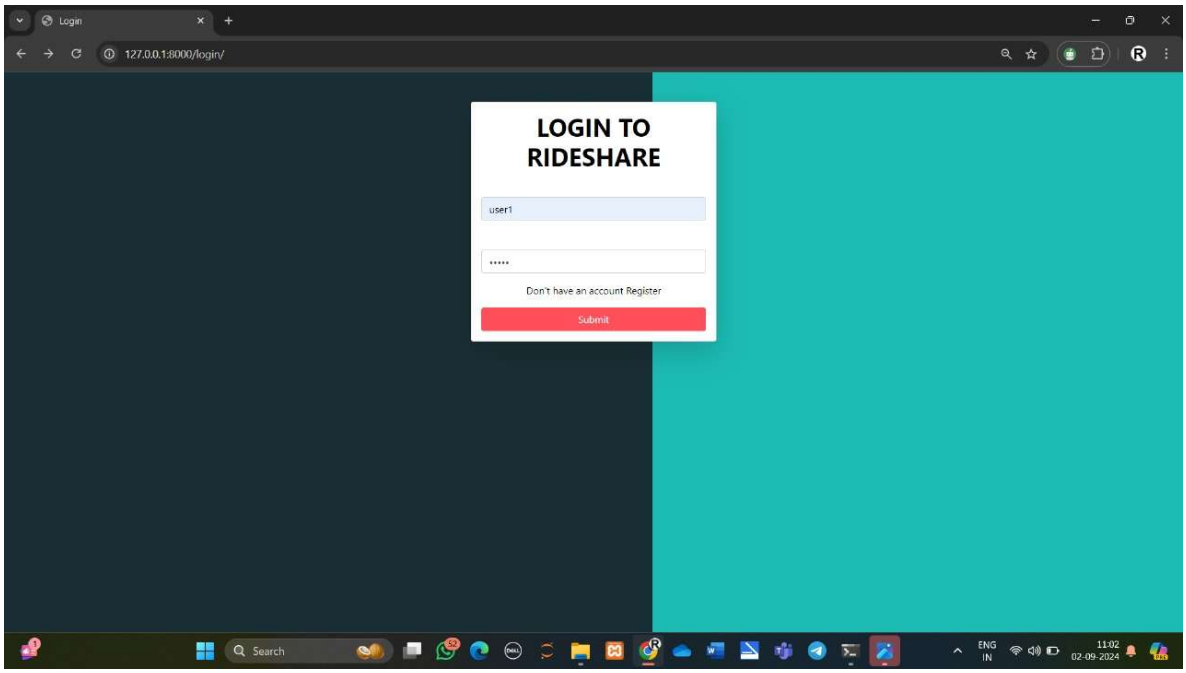

Fig 7.3 Login Page

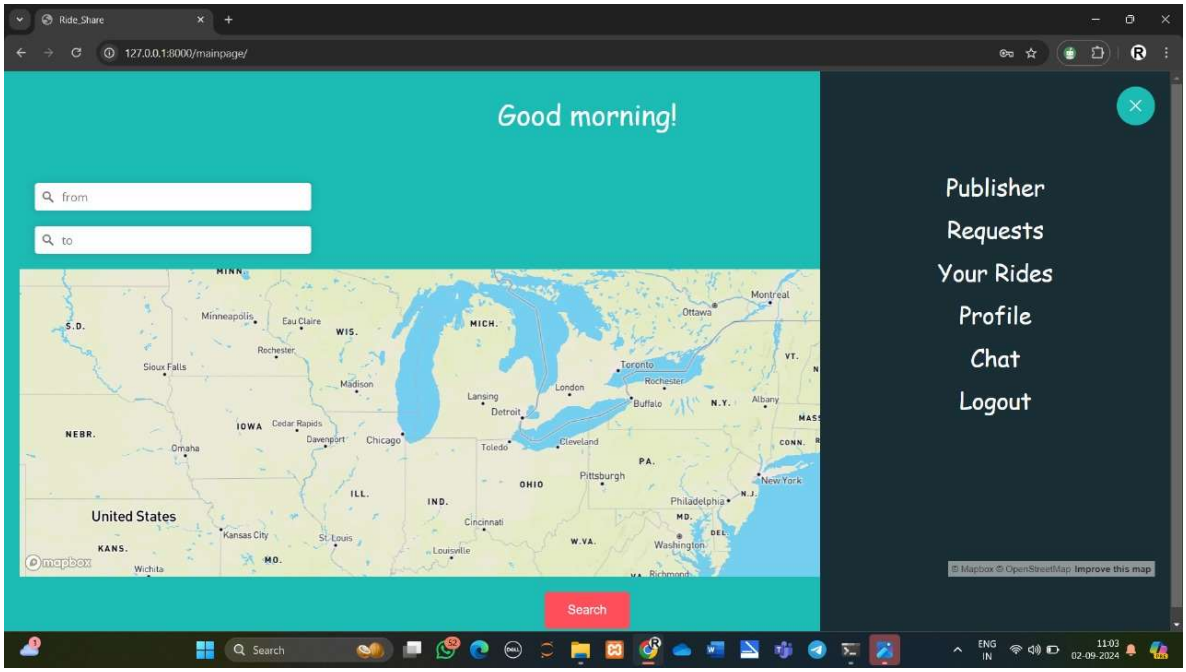

Fig 7.4 Home Page

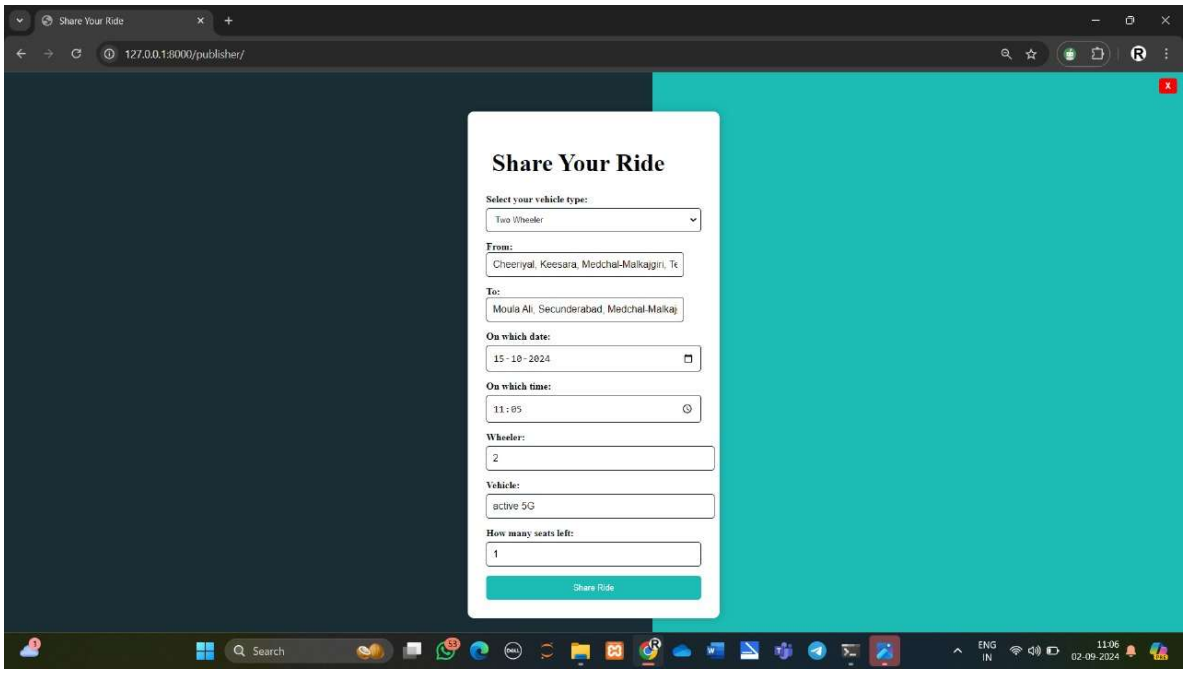

Fig 7.5 Publish Ride Page

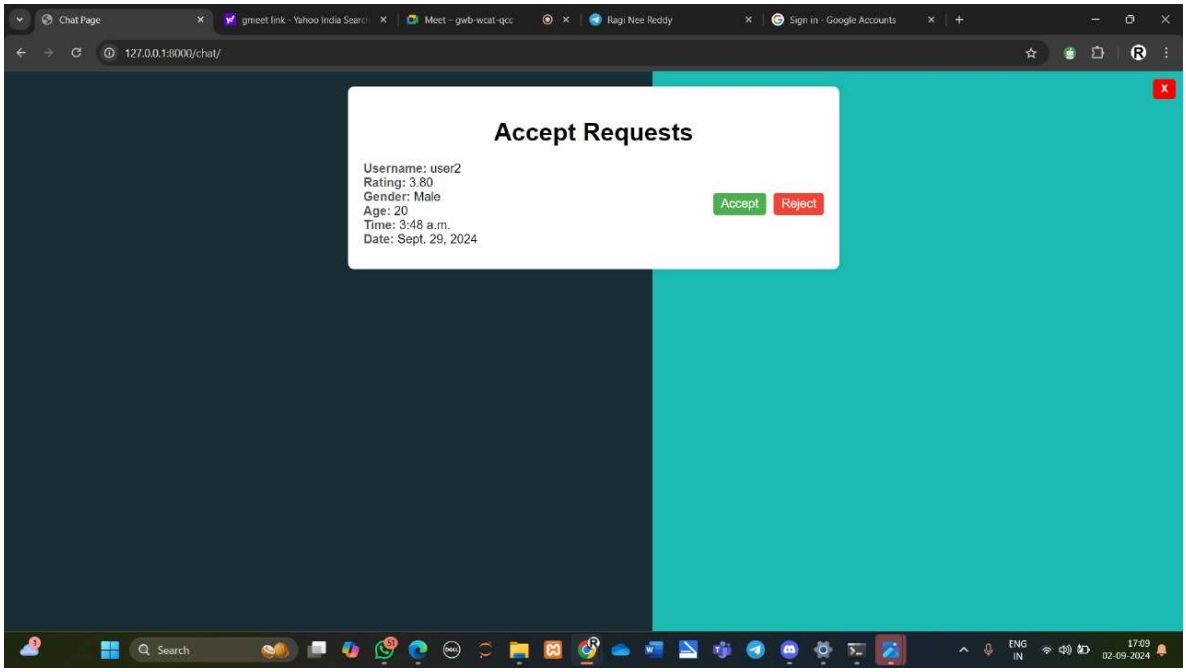

Fig 7.6 Ride Requests Page

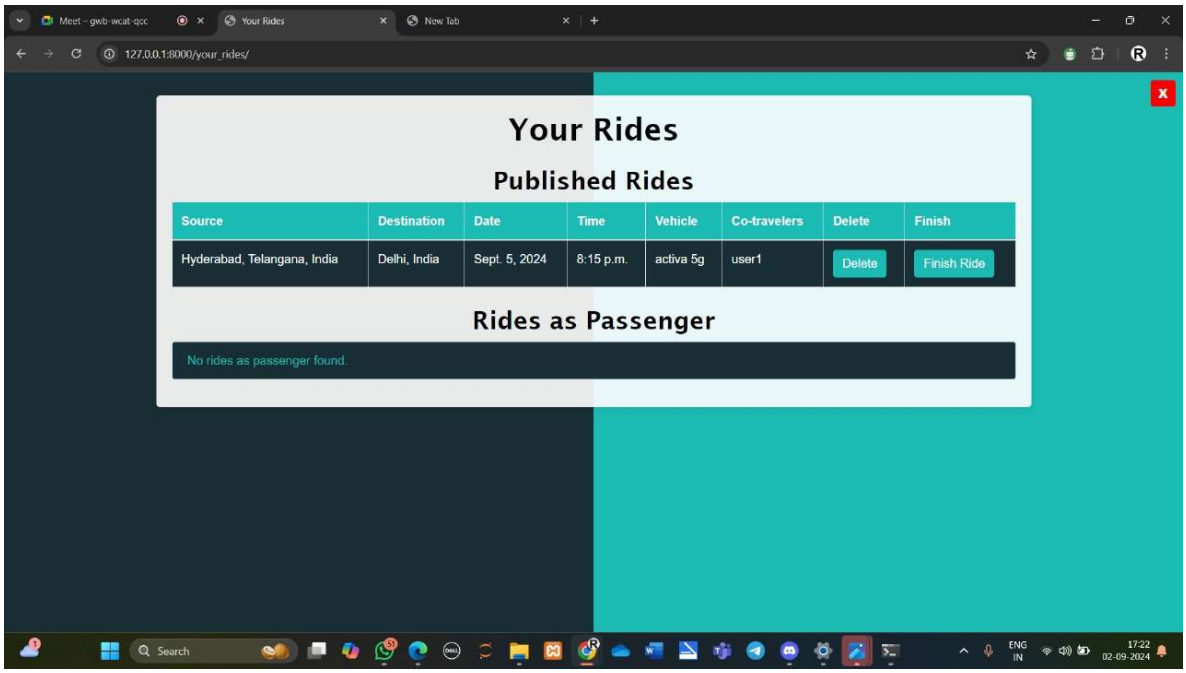

Fig 7.7 Your Rides Page

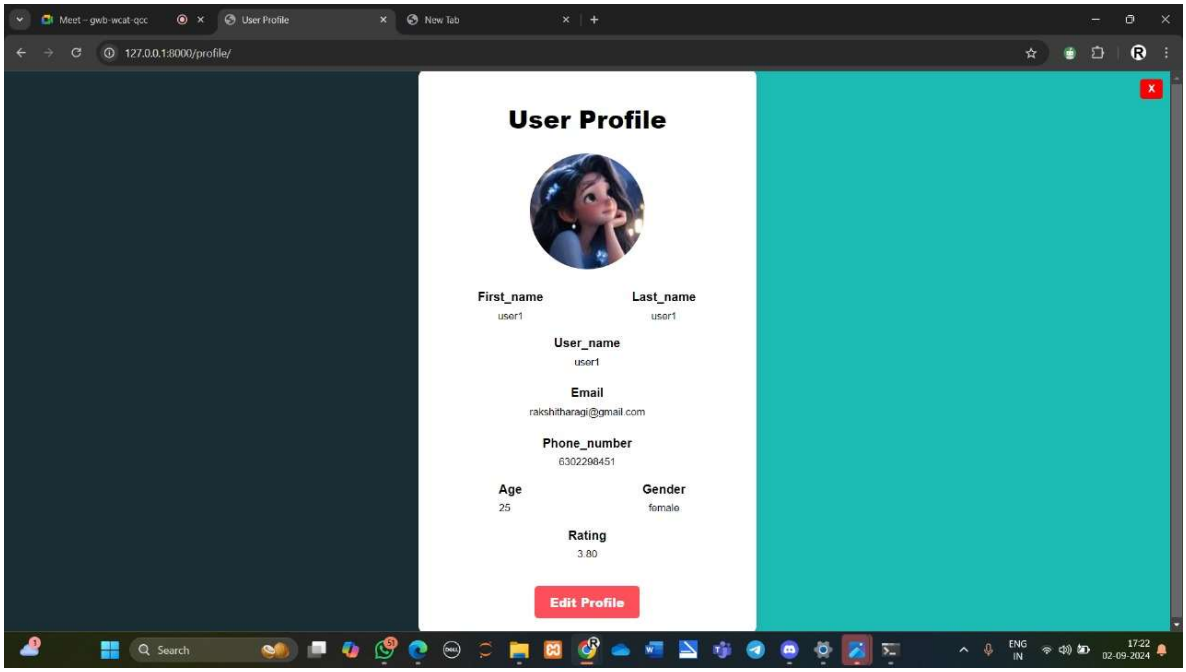

Fig 7.8 Profile Page

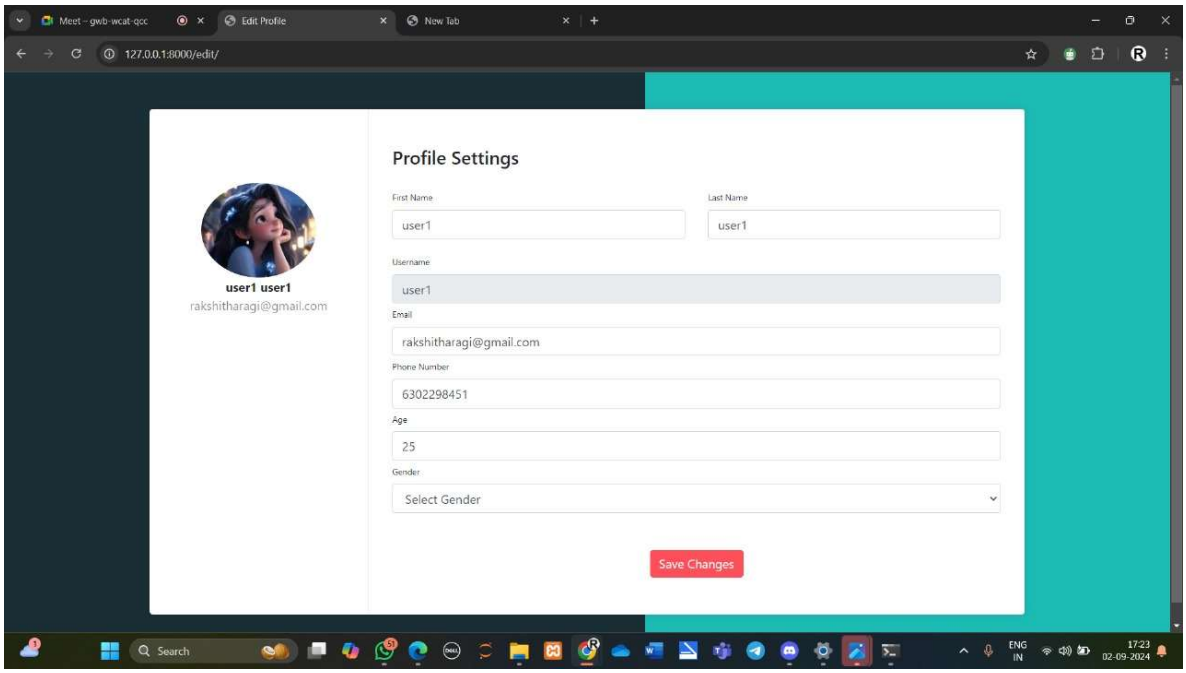

Fig 7.9 Edit Profile Page

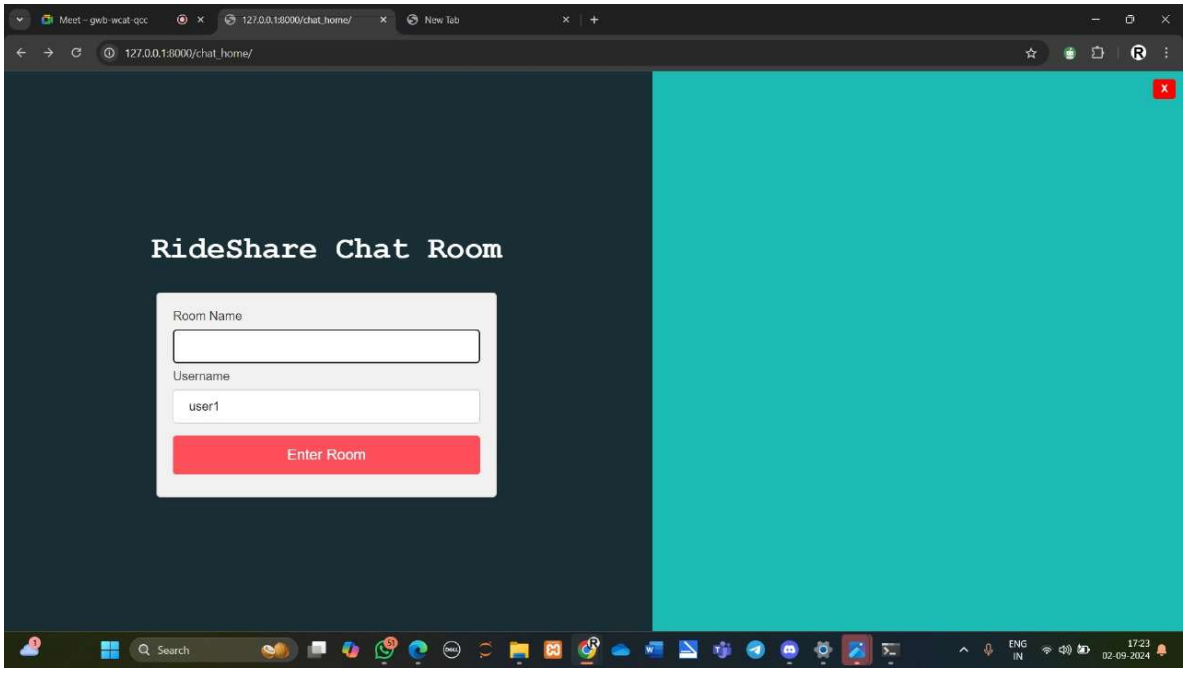

Fig 7.10 Chat Room Page

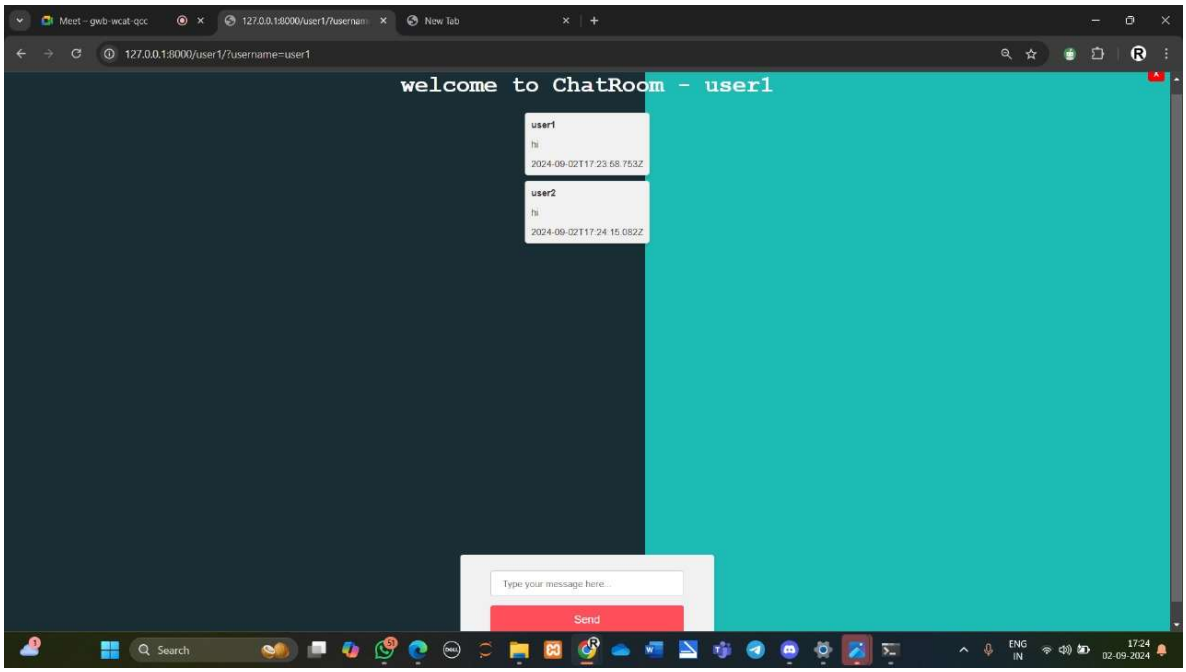

Fig 7.11 Chat Page

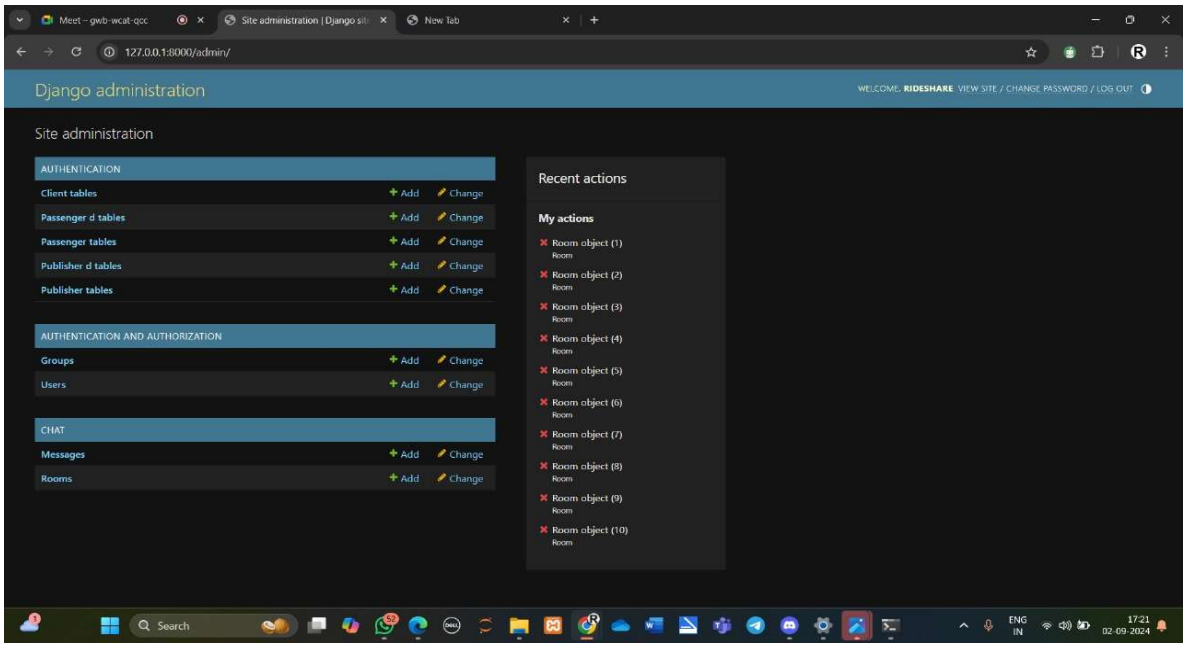

Fig 7.12 Admin Page

8. CONCLUSION

8.1 CONCLUSION

The RideShare project transforms the carpooling experience with its innovative, web-based platform that simplifies and enhances ride-sharing for all users. By providing a seamless interface for publishing, searching, and managing rides, the system streamlines the process of finding and booking rides. The integration of real-time features, such as chat ensures an interactive and efficient user experience. For administrators and publishers, the platform offers robust tools for managing ride details, handling requests, and collecting feedback, thereby improving operational efficiency and accuracy. Overall, the RideShare project showcases how modern web technologies can optimize traditional carpooling methods, delivering a more efficient and user-friendly service. The project's successful implementation sets a strong foundation for future enhancements and innovations in the ride-sharing domain.

8.2 FURTHER ENHANCEMENTS

1.Mobile Application Development: Transition the web-based application to a mobile app to increase accessibility and user convenience.

2.Live Location Tracking: Integrate live location tracking for rides to provide real-time updates and improve safety.

3.Integration with Payment Gateways: Introduce payment gateway integration to facilitate secure transactions and handle ride payments directly through the platform.

4.Dynamic Pricing Model: Introduce a dynamic pricing model that adjusts costs based on distance, demand, and other factors.

5. WhatsApp Integration: Implement a "Contact via WhatsApp" feature to facilitate real-time communication between users, allowing for seamless and familiar messaging.

9. BIBLIOGRAPHY

9.1 REFERENCES

1. https://github.com/kannan4k/django-carpool

2. https://1000projects.org/mini-project-on-online-car-pooling-system.html

3. https://github.com/karancode-singh/ShareMyGaddi-carpooling

4. https://www.free-css.com/free-css-templates

10. APPENDICES

A. Software Used

- **Django:** The web framework used to build the RideShare application.
- **PostgreSQL:** The relational database management system used to store user data, rides, ratings, etc.
- **Mapbox API:** Used for geolocation services, providing location suggestions and mapping functionalities.
- **AJAX:** Implemented for real-time chat communication between users.
- **HTML, CSS, JS:** Used for frontend development, ensuring a responsive and interactive user interface.

B. Testing Methods Used

- **Unit Testing:** Conducted to verify that individual components of the application, such as user registration, login, and ride publication, function as intended.
- **Integration Testing:** Ensured that different modules of the application (e.g., authentication, ride management etc) work together seamlessly.
- **Functional Testing:** Validated that the application's functionalities, such as publishing rides, sending ride requests, and rating rides, operate correctly in the real-world environment.
- **Security Testing:** Ensured the application is secure against common vulnerabilities like unauthorized access, and verified that sensitive data is encrypted.

C. Glossary

- **Publisher**: A user who creates a ride listing.
- **Passenger:** A user who requests to join a ride.

- **CSRF:** Cross-Site Request Forgery, a type of attack that forces users to execute unwanted actions on a web application in which they are authenticated.
- **AJAX:** Asynchronous JavaScript and XML, used to update web pages asynchronously by exchanging small amounts of data with the server behind the scenes.

D. System Requirements

- **Operating System:** Windows 11.
- **Browser:** Chrome, Firefox, Edge, or Safari with JavaScript enabled.
- **Memory:** Minimum 4 GB RAM.
- **Internet Connection:** Stable internet connection.

E. Database Schema

1. PassengerTable

- username1, username2, source, destination, date, time, finished

2. PublisherTable

- username, phoneNumber, source, destination, date, time, wheele, vehicle, seats, discription, req1, req2, req3, requested_by, finished

3. ClientTable

- username, phone_number, gender, rating, sumofrating, ppl, age, profile_pic, email

4. Room

- name

5. Message

- value, date, user, room