Finalization Phase

RentOBike

Introduction

This page is the final page of the project, named as, RentOBike. It is a web application project. This project is created for the people who want to rent the bicycle for the tourism purpose or for the use from house to school, university or the office or roaming purposes. Moreover, the owner of the bicycle who wants that community would use their bicycles for these purposes. The path is from making the first idea and complete it in the final part is the hard task and great for learning also. In this stage, we look back the project's path, changes made from feedback, technical issues fixed, all the important lessons learned along the way.

Project Recapitulation

Our website, RentOBike, is the online platform that is established by seeing the need of the eco-friendly system. The website is helped the bike owners and people who wants to rent the bicycles. And both of the parties can connect easily. For the website, we created the tools that make sure what users want in actual and fulfil their needs. RentOBike has become a new kind of solution in the bike rental market.

Objectives Achieved

- User-Friendly Interface: It is developed using Flask and Jinja2. It ensures that a smooth and intuitive user experience.
- Database Integration: we employed the Flask-SQLAlchemy for efficient data handling and storage, managing user and bicycle information securely.
- Responsive Design: We ensure that the application is accessible across the various devices so it can enhance the user engagement.
- Technical Debt Management: We addressed the initial technical debts, including the integration of a payment feature and search functionality. So it can enhance the platform utility and user satisfaction.

Development Insights and Reflections

The development phase is characterized by iterative progress, getting feedback from the conception phase, and refine the project to meet the user needs. The following enhancements are included:

- Enhanced Security Features
- User Experience Improvements
- Technical Debt Addressing

Lessons Learned

The lessons that been learned while doing the project are:

- Importance of Feedback
- Agility in Development
- Technical Debt Management

Conclusions and Future Directions

Through, this website RentOBike, we try to show that how the digital solutions can solve the real-world problems. For future, we may try to make it better and more powerful or try to add more features. We can plan to use more data analysis to give a better user experience. We would see if we can offer the service in more places.

In the end, we can say that the project met its goal and teach us a lot of things. It shows us that being a creative one how we can solve the real-world problems by creating the software.

Acknowledgments

We would like to thank the Prof. Dr. Holger Klus, who provided the feedback from the conception and the development phase that help us to create and complete the project on time.

Appendix

1. app.py

```
from flask import Flask, render_template, request, redirect, url_for, session, flash
from flask_sqlalchemy import SQLAlchemy
from werkzeug.security import generate_password_hash, check_password_hash
from werkzeug.utils import secure_filename
from datetime import datetime, timezone
import os
from itsdangerous import URLSafeTimedSerializer, SignatureExpired, BadSignature


#Sets up the Flask application, SQLAlchemy database URI, modifications tracking, secret keys, and upload folder for storing bicycle photos.
app = Flask(__name__)
UPLOAD_FOLDER = 'static/uploads'
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///rentobike.db'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
app.config['SECRET_KEY'] = 'your_secret_key'
app.config['SECURITY_PASSWORD_SALT'] = 'your_security_password_salt'
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER
db = SQLAlchemy(app)

#Defines the data models for User, Bicycle, Booking, and Review using SQLAlchemy. These models include fields relevant to their respective entities and relationships between them, such as users owning bicycles and making bookings.
class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(80), unique=True, nullable=False)
```

```
    email = db.Column(db.String(120), unique=True, nullable=False)
    password = db.Column(db.String(80), nullable=False)
    user_type = db.Column(db.String(20), nullable=False)
    address = db.Column(db.String(255), nullable=True)
    pincode = db.Column(db.String(20), nullable=True)
    mobile_number = db.Column(db.String(20), nullable=True)
    city = db.Column(db.String(100), nullable=True)
    country = db.Column(db.String(100), nullable=True)

class Bicycle(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(100), nullable=False)
    description = db.Column(db.String(255), nullable=True)
    owner_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)
    image_url = db.Column(db.String(255), nullable=True)
    bike_type = db.Column(db.String(50), nullable=True)
    # Add more fields as necessary, such as image URLs

    owner = db.relationship('User', backref=db.backref('bicycles', lazy=True))


class Booking(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    start_date = db.Column(db.DateTime, nullable=False)
    end_date = db.Column(db.DateTime, nullable=False)
    bicycle_id = db.Column(db.Integer, db.ForeignKey('bicycle.id'), nullable=False)
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)
    status = db.Column(db.String(50), nullable=False, default='pending')  # Example statuses: pending,
confirmed, cancelled
    bicycle = db.relationship('Bicycle', backref=db.backref('bookings', lazy=True))
    user = db.relationship('User', backref=db.backref('bookings', lazy=True))

class Review(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    rating = db.Column(db.Integer, nullable=False)
    comment = db.Column(db.Text, nullable=True)
    bicycle_id = db.Column(db.Integer, db.ForeignKey('bicycle.id'), nullable=False)
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)
    created_at = db.Column(db.DateTime, default=datetime.now(timezone.utc))

    bicycle = db.relationship('Bicycle', backref=db.backref('reviews', lazy=True))
    user = db.relationship('User', backref=db.backref('reviews', lazy=True))

with app.app_context():
    db.create_all()

#Generates a secure token for email confirmation using URLSafeTimedSerializer.
def generate_confirmation_token(email):
    serializer = URLSafeTimedSerializer(app.config['SECRET_KEY'])
    return serializer.dumps(email, salt=app.config['SECURITY_PASSWORD_SALT'])


#Validates a given token with an expiration time, used for email confirmation links.
def confirm_token(token, expiration=3600):
    serializer = URLSafeTimedSerializer(app.config['SECRET_KEY'])
    try:
        email = serializer.loads(
            token,
            salt=app.config['SECURITY_PASSWORD_SALT'],
            max_age=expiration
        )
    except (SignatureExpired, BadSignature):
```

```
        return False
    return email


#The homepage view, displaying the latest bicycles available for rent and user session information.
@app.route('/')
def home():
    bicycles = Bicycle.query.order_by(Bicycle.id.desc()).limit(4).all()
    username = session.get('username')
    return render_template('home.html', username=username, bicycles=bicycles)



#Handles user registration, including form submission, password hashing, and user creation in the
database.
@app.route('/signup', methods=['GET', 'POST'])
def signup():
    if request.method == 'POST':
        username = request.form.get('username')
        email = request.form.get('email')
        password = request.form.get('password')
        hashed_password = generate_password_hash(password)
        user_type = request.form.get('user_type')
        address = request.form.get('address')
        pincode = request.form.get('pincode')
        mobile_number = request.form.get('mobile_number')
        city = request.form.get('city')
        country = request.form.get('country')
        terms_accepted = 'terms_accepted' in request.form  # Check if terms_accepted checkbox is
checked

        if not terms_accepted:
            flash("You must agree to the terms and conditions to sign up.", "error")
            return redirect(url_for('signup'))

        user = User.query.filter((User.username == username) | (User.email == email)).first()
        if user:
            flash("User already exists!", "error")  # Using flash to show error message
            return redirect(url_for('signup'))

        # Create a new User instance including the new fields
        new_user = User(
            username=username,
            email=email,
            password=hashed_password,
            user_type=user_type,
            address=address,
            pincode=pincode,
            mobile_number=mobile_number,
            city=city,
            country=country
        )


        db.session.add(new_user)
        db.session.commit()

        # Automatically log the user in
        session['user_id'] = new_user.id
        session['username'] = new_user.username
        session['user_type'] = new_user.user_type
```

```
        return redirect(url_for('home'))
    return render_template('signup.html')



#Manages user login, including authentication and session management.
@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        username = request.form.get('username')
        password = request.form.get('password')
        user = User.query.filter_by(username=username).first()

        if user and check_password_hash(user.password, password):
            session['user_id'] = user.id
            session['username'] = user.username
            session['user_type'] = user.user_type

            return redirect(url_for('home'))
        else:
            return "Invalid username or password", 401

    return render_template('login.html')



#Logs out a user by clearing their session data.
@app.route('/logout')
def logout():
    # Remove user from session to log them out
    session.pop('user_id', None)
    session.pop('username', None)
    return redirect(url_for('home'))



#Allows logged-in users to list a new bicycle for rent, including uploading images and saving bicycle
details to the database.
@app.route('/add-bicycle', methods=['GET', 'POST'])
def add_bicycle():
    if 'user_id' not in session:
        return redirect(url_for('login'))

    if request.method == 'POST':
        name = request.form.get('name')
        description = request.form.get('description')
        bike_type = request.form.get('bike_type')
        owner_id = session.get('user_id')

        # Ensure the upload directory exists
        os.makedirs(app.config['UPLOAD_FOLDER'], exist_ok=True)

        photo_urls = []
        if 'bicyclePhotos' in request.files:
            photo_files = request.files.getlist('bicyclePhotos')
            for photo in photo_files:
                if photo and photo.filename:
                    unique_filename =
f"{datetime.now().strftime('%Y%m%d%H%M%S')}_{secure_filename(photo.filename)}"
                    photo_path = os.path.join(app.config['UPLOAD_FOLDER'], unique_filename)
                    photo.save(photo_path)
                    # Store the relative path from the static directory
```

```
                photo_urls.append(os.path.join('uploads', unique_filename))

        # Here you would need to adjust how you handle multiple image URLs based on your database
schema
        image_url = ';'.join(photo_urls)  # Example: joining URLs by semicolon as a simple solution

        new_bicycle = Bicycle(name=name, description=description, bike_type=bike_type,
owner_id=owner_id, image_url=image_url)
        db.session.add(new_bicycle)
        db.session.commit()

        return redirect(url_for('home'))

    return render_template('add_bicycle.html')


#Displays the details of a specific bicycle, including its reviews and images.
@app.route('/bicycle_details/<int:bicycle_id>')
def bicycle_details(bicycle_id):
    bicycle = Bicycle.query.get_or_404(bicycle_id)
    # Splitting the image URLs into a list if image_url is not None
    if bicycle.image_url:
        bicycle.image_urls = bicycle.image_url.split(';')
    else:
        bicycle.image_urls = []
    return render_template('bicycle_details.html', bicycle=bicycle)


#Shows the profile of the currently logged-in user, including their listed bicycles and bookings.
@app.route('/profile')
def profile():
    if 'user_id' not in session:
        return redirect(url_for('login'))

    user_id = session.get('user_id')
    user = User.query.get(user_id)
    if not user:
        return "User not found", 404

    return render_template('user_profile.html', user=user)


@app.route('/terms')
def terms():
    return render_template('terms.html')


@app.route('/faq')
def faq():
    return render_template('faq.html')


@app.route('/contact')
def contact():
    return render_template('contact.html')


@app.route('/privacy')
def privacy():
    return render_template('privacy.html')
```

```python
#Handles the booking of a bicycle by a user, including date selection and booking confirmation.
@app.route('/book-bicycle/<int:bicycle_id>', methods=['GET', 'POST'])
def book_bicycle(bicycle_id):
    if 'user_id' not in session:
        return redirect(url_for('login'))

    bicycle = Bicycle.query.get_or_404(bicycle_id)

    if request.method == 'POST':
        start_date_str = request.form['start_date']
        end_date_str = request.form['end_date']

        # Convert string dates to datetime objects
        start_date = datetime.strptime(start_date_str, '%Y-%m-%d')
        end_date = datetime.strptime(end_date_str, '%Y-%m-%d')

        new_booking = Booking(
            user_id=session['user_id'],
            bicycle_id=bicycle_id,
            start_date=start_date,
            end_date=end_date
        )
        db.session.add(new_booking)
        db.session.commit()

        return redirect(url_for('booking_confirmation', booking_id=new_booking.id))

    return render_template('book_bicycle.html', bicycle=bicycle)


#Allows users to submit reviews for bicycles they have rented, including ratings and comments.
@app.route('/submit_review/<int:bicycle_id>', methods=['POST'])
def submit_review(bicycle_id):
    if 'user_id' not in session:
        return redirect(url_for('login'))

    rating = request.form.get('rating')
    comment = request.form.get('comment')
    user_id = session['user_id']

    review = Review(rating=rating, comment=comment, bicycle_id=bicycle_id, user_id=user_id)
    db.session.add(review)
    db.session.commit()

    return redirect(url_for('bicycle_details', bicycle_id=bicycle_id))


#Lists all bicycles available for rent, with pagination.
@app.route('/bicycles')
def bicycles():
    page = request.args.get('page', 1, type=int)  # Default to first page
    per_page = 10  # Number of items per page

    # Fetch the paginated bicycles
    pagination = Bicycle.query.paginate(page=page, per_page=per_page, error_out=False)
    bicycles = pagination.items

    return render_template('bicycles.html', bicycles=bicycles, pagination=pagination)


#Displays a custom 404 error page when a route is not found.
@app.errorhandler(404)
```

```python
def page_not_found(e):
    return render_template('404.html'), 404


@app.route('/bicycle/<int:bicycle_id>')
def bicycle_detail(bicycle_id):
    bicycle = Bicycle.query.get_or_404(bicycle_id)
    return render_template('bicycle_details.html', bicycle=bicycle)

@app.route('/booking-confirmation/<int:booking_id>')
def booking_confirmation(booking_id):
    booking = Booking.query.get_or_404(booking_id)
    bicycle = booking.bicycle  # Access the bicycle related to the booking
    owner = User.query.get(bicycle.owner_id)  # Access the owner of the bicycle

    if not owner.address or not owner.city or not owner.mobile_number:
        print("Missing owner information for user ID:", owner.id)
    return render_template('booking_confirmation.html', booking=booking, bicycle=bicycle,
owner=owner)



#Password reset functionality through forgot_password() and reset_password() routes, facilitating
users to reset their passwords if forgotten.
@app.route('/forgot-password', methods=['GET', 'POST'])
def forgot_password():
    if request.method == 'POST':
        email = request.form.get('email')
        mobile_number = request.form.get('mobile_number')

        return redirect(url_for('reset_password'))

    return render_template('forgot_password.html')

@app.route('/reset-password', methods=['GET', 'POST'])
def reset_password():
    if request.method == 'POST':
        password = request.form.get('password')
        confirm_password = request.form.get('confirm_password')

        # Check if passwords match
        if password != confirm_password:
            flash('Passwords do not match. Please try again.', 'error')
            return redirect(url_for('reset_password'))

        user_id = session.get('user_id')
        if user_id:

            user = User.query.get(user_id)

            user.password = generate_password_hash(password)

            db.session.commit()

            flash('Password reset successfully. Please log in with your new password.', 'success')
            return redirect(url_for('login'))
        else:
            flash('Invalid user. Please try again.', 'error')
            return redirect(url_for('login'))

    return render_template('reset_password.html')
```

```
if __name__ == '__main__':
    app.run(debug=True)
```

2. home.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>RentOBike</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">
</head>
<body>

<header>
    <div class="header-content">
        <h1>
            <a href="{{ url_for('home') }}">RentOBike</a>
            <img src="{{ url_for('static', filename='logow.png') }}" class="logo">
        </h1>
        <nav id="navMenu">
            {% if session.get('username') %}
                <span>Welcome, <a href="{{ url_for('profile') }}">{{ session.get('username') }}</a>!</span>
                {% if session.get('user_type') == 'owner' %}
                    <a href="{{ url_for('add_bicycle') }}">Add Bicycle</a>
                {% endif %}
                <a href="{{ url_for('logout') }}">Sign Out</a>
            {% else %}
                <a href="{{ url_for('login') }}">Login</a>
                <a href="{{ url_for('signup') }}">Signup</a>
            {% endif %}
        </nav>
    </div>
</header>

<div class="sidebar">
    <a href="{{ url_for('home') }}">Home</a>
    <a href="{{ url_for('bicycles') }}">Available Bicycles for Rent</a>
</div>

<main>
    <section class="bicycles">
        <h2>Feel the Ride</h2>
        <div class="bicycle-gallery">
            {% for bicycle in bicycles %}
                <div class="bicycle">
                    <h3>{{ bicycle.name }}</h3>
                        <img src="{{ url_for('static', filename=bicycle.image_url) }}" alt="{{ bicycle.name }}">
                    <p>{{ bicycle.description }}</p>
                </div>
            {% else %}
                <p>No bicycles available at the moment.</p>
            {% endfor %}
        </div>
    </section>
</main>


<footer>
    <section class="about-us">
```

```
        <h3>About Us</h3>
        <nav>
            <a href="{{ url_for('contact') }}">Contact</a>
            <a href="{{ url_for('terms') }}">Terms and Conditions</a>
            <a href="{{ url_for('privacy') }}">Privacy Policy</a>
            <a href="{{ url_for('faq') }}">FAQ</a>
        </nav>
    </section>
</footer>


</body>
</html>
```

3. requirements.txt

```
blinker==1.7.0
click==8.1.7
colorama==0.4.6
Flask==3.0.2
Flask-SQLAlchemy==3.1.1
greenlet==3.0.3
itsdangerous==2.1.2
Jinja2==3.1.3
MarkupSafe==2.1.5
SQLAlchemy==2.0.29
typing_extensions==4.10.0
Werkzeug==3.0.1
```

4. docker-compose.yml

```
version: '3.8'
services:
  web:
    build: .
    ports:
      - "5000:5000"
    environment:
      FLASK_ENV: development
    volumes:
      - .:/app
```

5. Dockerfile

```
# Use an official Python runtime as a parent image
FROM python:3.8-slim

# Set the working directory in the container
WORKDIR /app

# Copy the current directory contents into the container at /app
COPY . /app

# Install any needed packages specified in requirements.txt
RUN pip install --no-cache-dir -r requirements.txt

# Make port 5000 available to the world outside this container
EXPOSE 5000

# Define environment variable
ENV FLASK_APP=app.py
ENV FLASK_RUN_HOST=0.0.0.0
```

```
# Run app.py when the container launches
CMD ["flask", "run"]
```

6. README.md

```
# Web Application Project

This project is a web application developed as part of the software engineering course. It showcases the design and implementation skills in creating a web-based solution using Flask, a popular Python web framework. The application aims to provide a simple yet functional interface for users to interact with.

## Features

- Web interface built with Flask.
- Integration with a database using Flask-SQLAlchemy for data persistence.
- User-friendly interface designed with Jinja2 templates.

## Getting Started

These instructions will guide you through setting up and running the project locally using Docker. Ensure you have Docker installed on your system before proceeding.

### Prerequisites

- Docker
- Git (for cloning the repository)

### Installation

1. **Clone the repository**

   Open a terminal and run:

   ```bash
   git clone https://github.com/ichsandeep/PSEproject.git

### Navigate to the project directory

- Change into the project directory with:

   ```bash
    cd PSEproject

### Running the Application with Docker

- Build the Docker image

- In the project root directory, where the Dockerfile is located, build the Docker image using:

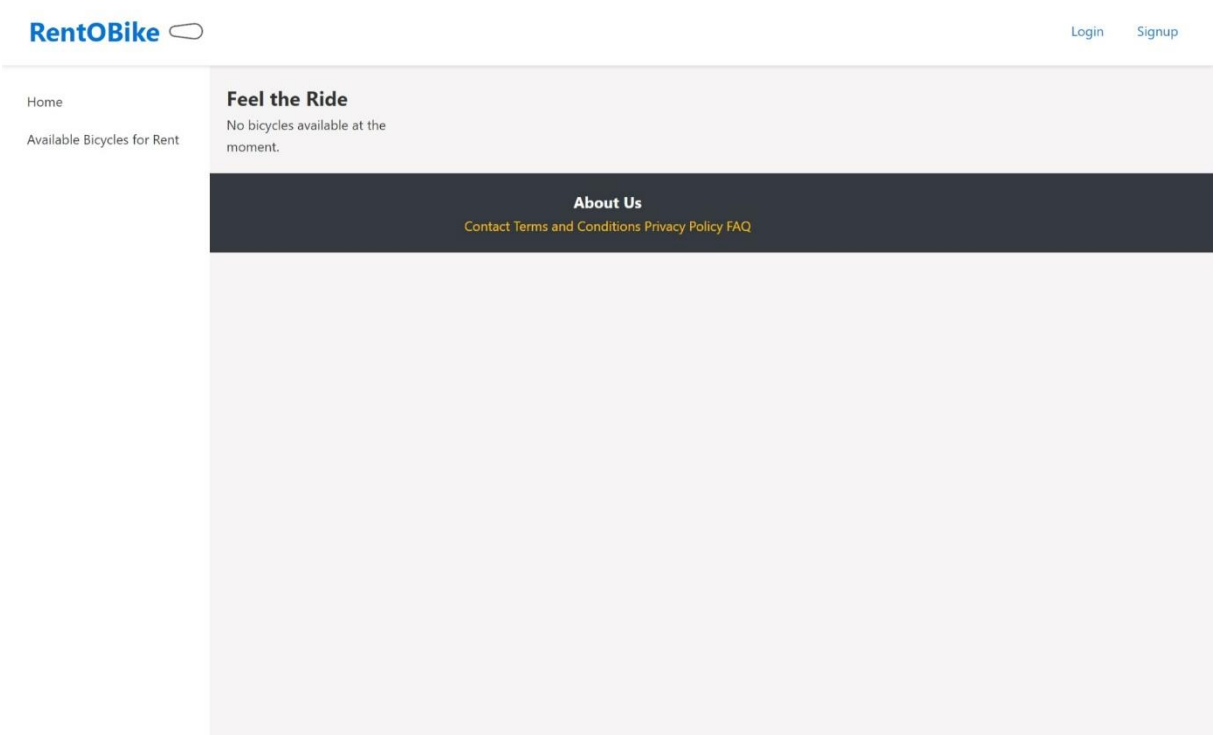
   ```bash
    docker-compose build

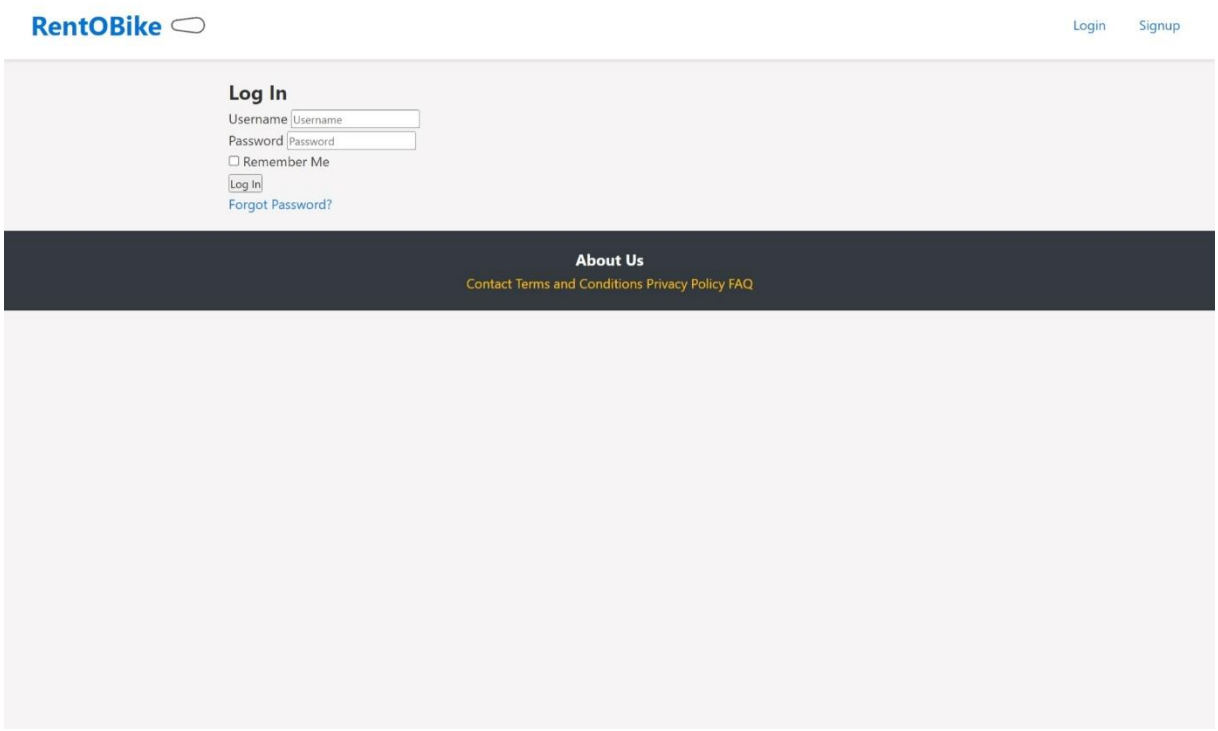
### Run the application

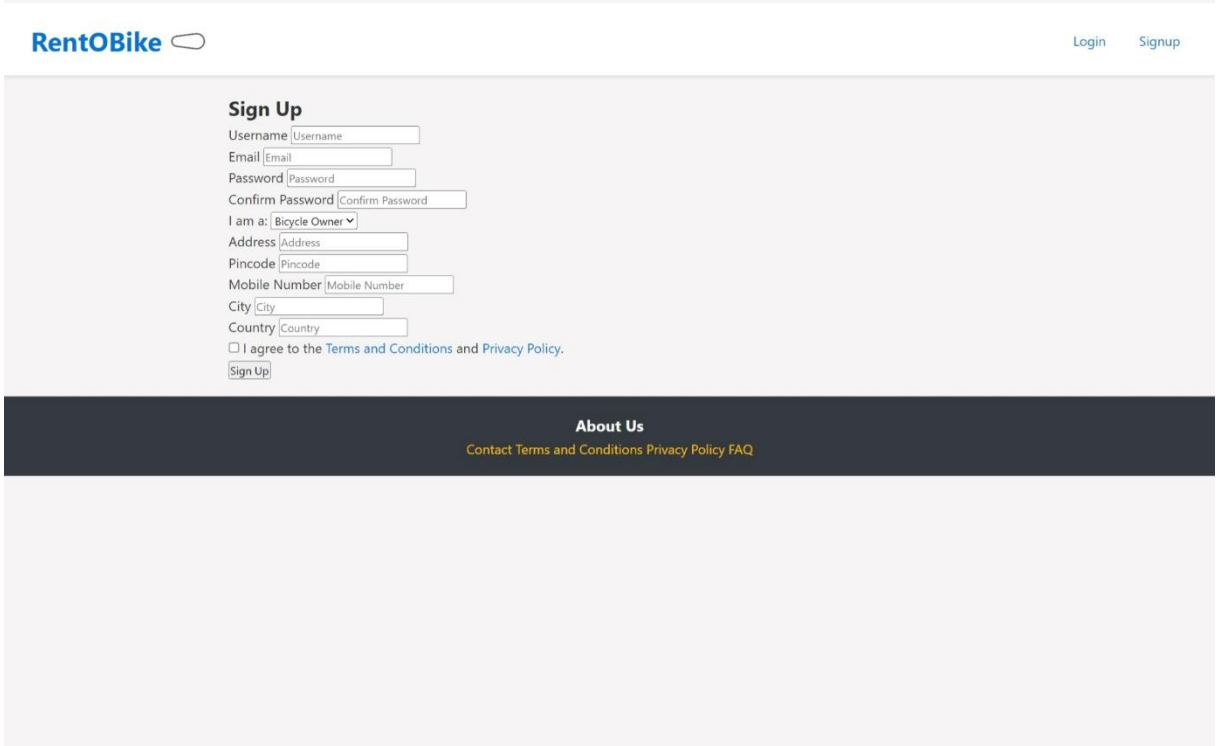
- Start the application with Docker Compose:

   ```bash
    docker-compose up

### Access the application
```

- The web application should now be running and accessible through your web browser at:

  ```arduino
  http://localhost:5000

- Adjust the port in the URL if you've configured it differently in your docker-compose.yml or Flask application.

### Contributing

- We welcome contributions to improve the project. Feel free to fork the repository, make changes, and submit pull requests. If you have any suggestions or encounter issues, please open an issue in the GitHub repository.

### License

- This project is licensed under the MIT License - see the LICENSE file in the repository for more details.

### Acknowledgments

- Thanks to the Flask team for creating such a powerful and easy-to-use web framework.
- Special thanks to all contributors and users of this project.

"This README is now ready to be used. Just ensure that all instructions accurately reflect your project's setup and requirements. If your project uses a different port or has additional setup steps, remember to update those parts accordingly."

## 7. Screenshots of the Webpage (Homepage, Login, Signup)


  ```





GitHub Repository Link

https://github.com/ichsandeep/PSEproject

https://github.com/ichsandeep/PSEproject/tree/main/Sandeep_Sandeep_102210527_DLMCSPSE01