

UDIO



CS09-608(P) B.Tech Mini Project 2016

Done By

Akshaynath R

Amrutha MU

Sachin Govind

Sreelal KM

Guided By

Swaraj KP

Asst Professor

Dept of Computer Science And Engineering
Government Engineering College
Thrissur-680009

ABSTRACT

7 continents, 196 countries with over 7 billion individuals, our world is a vast community which hangs in the balance of a complex network which connects each and every one of us. The grand success of the internet can be owed to its fundamental traits: speed and vast coverage. But in certain scenarios the existence of a virtual network is not sufficient enough and therefore our dependence on a primordial physical layer is unavoidable. The absence of an effective medium for transport of goods poses a day to day challenge for everyone, effective being the key aspect which requires the mode to be swift, accurate and secure. The existing systems involve a time consuming and tedious process. In todays world time is of the essence and definitely no one has time to spend on tasks like these.

Our goal is to create a peer to peer network for package delivery. We achieve this by routing the package in an optimized way. Instead of involving a 3rd party agency we link the sender and receiver via traveler who happens to be going on the same way. The traveler would have already signed up with our program by providing his or her credentials and he/she receives payment for each drop made. Through our program the client gets to dispatch the item through a larger pool of transporters and our end makes sure that the best one is selected. In this manner we are not limited on human resource as in the case of traditional delivery systems and we will never be overwhelmed by the sheer volume of deliveries to be made.

Each individual request is treated in real time and requests are not stacked one over the other. Features include a GPS tracking system to track the whereabouts of your package. Apart from the advantages that the client enjoys from our service, we believe that we have played our role in making the world a little greener. By routing packages through travelers we have removed the requirement for dedicated deliverymen which reduces fuel consumption and also lightens the load on traffic.

ACKNOWLEDGEMENT

We are very thankful to everyone who supported us. We are equally grateful to Asst. Prof. Swaraj KP for guiding and correcting various documents with attention and care. He has taken pain to go through the project and make necessary correction as and when needed. He gave us moral support and guided us in different matters regarding the topic. He had been very kind and patient while suggesting us the outlines of this project and correcting our doubts. We thank him for his overall support. We do extend our sincere thanks to our project coordinators for their valuable support and guidance, and seniors for their guidance.

We are also thankful to all developers of various softwares that we came into use during the course of our project. We would also like to thank the stackoverflow.com team for answering our doubts in the limited amount of time. A big thanks to google search engine for endless matching answers for our questions and also sharelatex.com for their online latex editor. We also thank our friends who has supported and helped us to complete this project successfully.

Contents

List of Tables	vi
List of Figures	vii
List of abbreviations	viii
1 Introduction	1
1.1 Problem Statement	1
1.1.1 Current situation	1
1.1.2 Proposed System	1
2 Feasibility Study	2
2.1 Technical Feasibility	2
2.1.1 Hardware Feasibility	2
2.1.2 Software Feasibility	2
2.2 Financial Feasibility	3
2.2.1 Development Cost	3
2.2.2 Installation Cost	3
2.2.3 Operational Cost	3
2.2.4 Maintenance Cost	3
2.3 Operational Feasibility	3
3 Requirement Analysis	4
3.1 Process Model	4
3.1.1 Agile Model	4
3.1.2 Model Description	4
3.1.2.1 Agile Principles:	4
3.1.2.2 Why Agile?	5
3.1.2.3 Future Enhancements:	6
3.2 Method of requirement elicitation	6
3.2.1 Questionnaire	6
3.2.2 Interview	7
3.3 User requirements	7
3.4 Project requirements	8
3.5 Requirements validation	8
3.6 Software Requirements Specification	8
3.6.1 Document Purpose	8
3.6.2 Product Scope	8

3.6.3	Intended Audience	9
3.6.4	Document Conventions	9
3.6.5	Overall Description	9
3.6.5.1	Product Perspective	9
3.6.5.2	Product Functionality	9
3.6.5.3	Users and Characteristics	10
3.6.5.4	Operating Environment	11
3.6.5.5	User Documentation	11
3.6.6	Specific Requirements	11
3.6.6.1	External Interface Requirements	11
3.6.6.2	Functional Requirements	11
3.6.7	Non Functional Requirements	12
3.6.7.1	Performance Requirement	12
3.6.7.2	Software Quality Attributes	12
4	Design	13
4.1	Introduction	13
4.1.1	Purpose	13
4.1.2	Scope	13
4.1.3	System Overview	13
4.2	Detailed Design	15
4.2.1	Use Case Diagram	15
4.2.2	Class Diagram	16
4.2.3	Entity Relationship Diagram	17
4.2.4	User Interface Design	18
5	Coding	21
5.1	Register	21
5.2	Login	22
5.3	Sender	23
5.4	Rider	24
6	Testing	25
6.1	Types of Testing done	25
6.1.1	Whitebox Testing	25
6.1.2	Blackbox Testing	26
6.2	Advantages and Limitations	27
6.2.1	Advantages	27
6.2.2	Limitations	27
6.3	Future Extensions if possible	28
7	Quality Assurance	29
7.1	Introduction	29
7.1.1	Purpose	29
7.1.2	Scope	29
7.2	Quality Assurance Strategy	30
7.3	Audits and Reviews	31

7.3.1	Work Product Reviews	31
7.4	Further Extension	31
8	Conclusion	33
	Appendices	35
A	User Document	36
A.1	Using the Web Application	36

List of Tables

6.1	Whitebox testing	26
6.2	Blackbox testing	27
7.1	Software Lifecycle/items	30
7.2	Quality Assurance Strategy	31
7.3	Work Product Reviews	31

List of Figures

3.1	Agile Model	5
3.2	Percentage of travelers with cargo space empty	7
3.3	User Feedback	7
4.1	Use Case Diagram	15
4.2	Class Diagram	16
4.3	ER Diagram	17
4.4	Home Page	18
4.5	Sign In Page	18
4.6	Sign Up Page	19
4.7	Dashboard: Rider	19
4.8	Dashboard: Sender	20
A.1	Home Page	36
A.2	Sign In	37
A.3	Sign Up	38
A.4	Create Ride	39
A.5	Create Package	40

List of Abbreviations

GUI - Graphical User Interface
HTTP - Hyper Text Transfer Protocol
HTML - Hyper Text Markup Language
DB - Database
CSS - Cascading Style Sheets
SDLC - Software Development Life Cycle
OS - Operating System
GPS - Global Positioning System
Admin - Administrator
IEEE - Institute of Electrical and Electronics Engineers
API - Application Program Interface
IDE - Integrated Development Environment
UML - Unified Modeling Language
ER - Entity Relationship
SQAP - Software Quality Assurance Plan

Chapter 1

Introduction

1.1 Problem Statement

The aim of the project is to create a web application for a peer to peer network for package delivery. The website allows the sender to route the package through a rider traveling on the same route. Other functionalities include package status check and rider tracking.

1.1.1 Current situation

The motive behind our project arises from the absence of an efficient medium of transport goods. The existing system involves a time consuming and tedious process. We improve this system by eliminating the third party delivery agency and directly link the sender and receiver via a traveller who happens to be travelling in the same direction.

1.1.2 Proposed System

The rider is made to upload his credentials to ensure security and safety of the package. This registers the rider on our site. He/she is now able to upload his/her travel plans on our site. Using this information provided by the rider, a sender would be able to see available riders on the route. He/she then selects a rider whose contact info is displayed on our site. After contacting the rider, the package is dispatched and dropped off at the destination.

Chapter 2

Feasibility Study

2.1 Technical Feasibility

We have analysed the technical feasibility of the project and it is feasible based on the following factors.

2.1.1 Hardware Feasibility

The minimum hardware requirements for developing web application and android app inorder to implement peer to peer package delivery system are given below:

- An Android Device running Android 4.1 Jellybean (or above)
- A computer that can run web browser
- Internet connectivity

2.1.2 Software Feasibility

The minimum software requirements for developing and running web application and android app inorder to implement peer to peer package delivery system are given below:

- Rethink DB Database
- HTML
- CSS
- JavaScript
- Android Development Tools
- Nginx Web Server
- OS- Linux/Windows 8 (or above)

2.2 Financial Feasibility

2.2.1 Development Cost

The cost involved in development purpose includes the web server cost, domain name cost, internet charges and the cost of devices needed for developing the application and demonstrating its working.

2.2.2 Installation Cost

No particular installation cost is needed other than the cost of hardware devices.

2.2.3 Operational Cost

Execution of the application does not actually require any operational cost. The only operational cost required is the cost of power supplies to hardware devices as well as the internet connectivity charges if any.

2.2.4 Maintenance Cost

Includes cost required for the maintenance of the server.

2.3 Operational Feasibility

The operational feasibility of the application is dependent on the following factors:

- A stable net connection to run the server
- Number of active travellers in the same route
- GPS coverage en route

Chapter 3

Requirement Analysis

3.1 Process Model

3.1.1 Agile Model

Agile model is selected for the project. Agile SDLC model is a combination of iterative and incremental process models with focus on process adaptability and customer satisfaction by rapid delivery of working software product. Since we are building a customer based software it is very essential that customer feedback is obtained at the earliest of development stages. Agile model helps to gain customer feedback at the earliest and also to bring frequent updates to the project very easily. The working software can be made available in minimum amount of time which will help in future testing and analysis. Therefore, Agile SDLC model is the most suitable SDLC model which can be used for our project.

3.1.2 Model Description

Agile model believes that every project needs to be handled differently and the existing methods need to be tailored to best suit the project requirements. In agile the tasks are divided to time boxes (small time frames) to deliver specific features for a release.

Iterative approach is taken and working software build is delivered after each iteration. Each build is incremental in terms of features; the final build holds all the features required by the customer.

3.1.2.1 Agile Principles:

The Agile Manifesto is based on twelve principles:

- Customer satisfaction by early and continuous delivery of valuable software
- Welcome changing requirements, even in late development
- Working software is delivered frequently (weeks rather than months)

- Close, daily cooperation between business people and developers
- Projects are built around motivated individuals, who should be trusted
- Face-to-face conversation is the best form of communication (co-location)
- Working software is the principal measure of progress
- Sustainable development, able to maintain a constant pace
- Continuous attention to technical excellence and good design
- Simplicitythe art of maximizing the amount of work not doneis essential
- Best architectures, requirements, and designs emerge from self-organizing teams
- Regularly, the team reflects on how to become more effective, and adjusts accordingly

Here is a graphical illustration of the Agile Model:



Figure 3.1: Agile Model

3.1.2.2 Why Agile?

- Is a very realistic approach to software development.
- Promotes teamwork and cross training.
- Functionality can be developed rapidly and demonstrated.
- Resource requirements are minimum.
- Suitable for fixed or changing requirements.

- Delivers early partial working solutions.
- Good model for environments that change steadily.
- Minimal rules, documentation easily employed.
- Enables concurrent development and delivery within an overall planned context.
- Little or no planning required.
- Easy to manage.
- Gives flexibility to developers.

3.1.2.3 Future Enhancements:

- Cross platform implementation of the Application.
- Improving the User Interfaces according to user feed-backs.
- Including artificial intelligence and machine learning inorder to enhance matching algorithm.
- Implementing monetization methods.

3.2 Method of requirement elicitation

The techniques we have used for gathering requirements from users were:

- Questionnaire
- Interview

3.2.1 Questionnaire

We delivered a set of questions to users for analyzing their feedback, a sample Questionnaire that was produced before the users is given below:

- Do you send and receive packages frequently?
- Do you have easy access to internet?
- Have you ever wished that shipment delivery be made on the same day?
- Have you ever felt that it would have been better if there was any means through which the heavy amount of time spent on official procedures/ protocols could be eliminated when packages are sent?
- Have you ever wished for a logistics facility which can deliver your package to the receiver safe and sound?

- Are you a frequent traveler?
- Do you often travel alone with luggage space to spare?
- Have you ever wondered that the money you pay for your car fuel when you are travelling alone is too much?

3.2.2 Interview

The same set of questions which were used in the questionnaire was used for the interview too.

3.3 User requirements

On the basis of requirement survey conducted among the users, we reached to a conclusion that more than 90% of the users frequently travel with their cargo space mostly empty. We also found that about 70% agreed to the fact that our program proves to be profitable for them.

Here are some of our findings represented graphically:

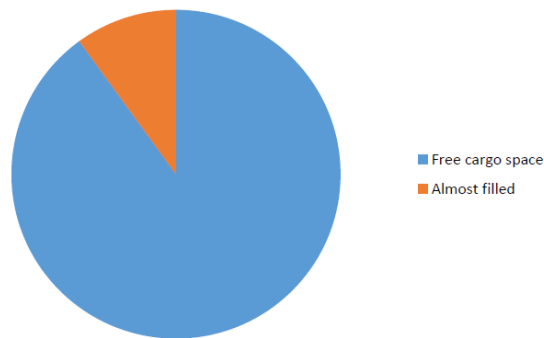


Figure 3.2: Percentage of travelers with cargo space empty

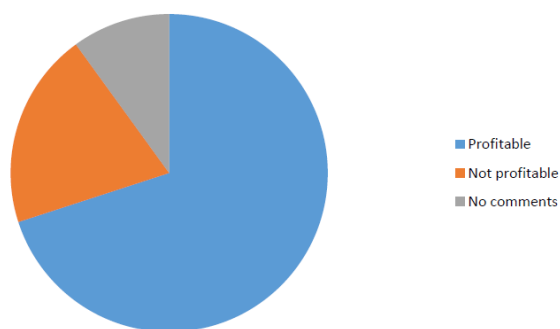


Figure 3.3: User Feedback

3.4 Project requirements

On the basis of the requirements demanded by the user the following were found:

- Ability to deliver a package courier on the same day of dispatch
- Ability to match the sender with a person travelling in the same direction as the destination of the package the sender has to ship
- Ability to make use of the cargo space in a persons vehicle and compensate for the vehicles fuel cost while he/she is travelling
- Ability to keep the sender constantly updated about the progress of his/her shipment through an intelligent tracking system

3.5 Requirements validation

The goal of requirement validation is to seek out and correct problems before resources are committed to implementing the requirements. It is concerned with examining the requirements to certify that they meet the stakeholders intentions and to ensure that they define the right system. The key activities in requirement validation are conducting requirement reviews, demonstrating prototypes, validating the conceptual models etc. We are adopting requirement review as mechanism for requirement validation.

3.6 Software Requirements Specification

The software requirements specified in this chapter are applicable for the development of a web application for package delivery. The document gives the detailed description of the both functional, non-functional and interface requirements of the project.

3.6.1 Document Purpose

The purpose is to create a peer to peer network for package delivery. The software can be used by the sender and rider and managed by an administrator. This mainly consists of product details, route and time details, payment details, online rating facility etc. New sender and rider can register into this by giving their details. The users can login into their home page using username and password. This software interfaces with the users and provides a better package delivery service.

3.6.2 Product Scope

The absence of an effective medium for transport of goods poses a day to day challenge for everyone, effective being the key aspect which requires the mode

to be swift, accurate and secure. The existent system involves time consuming and tedious process. In today's world time is of the essence and definitely no one has time to spend on tasks like these. Our system creates a peer to peer network for package delivery. Instead of involving a third party delivery agency, we link the sender and receiver via a traveller who happens to be going on the same direction. The traveller would have already signed up with our programs by providing his/her credentials and he/she receives payment for each drop made. The client gets to dispatch the item through a larger pool of transporters and our end makes sure that the best one is selected. A GPS tracking system is also included to track the whereabouts of the package.

3.6.3 Intended Audience

This document is intended for those who want to use a fast and efficient system to send their packages and the riders who are ready to use their empty cargo space for delivering packages, securely with a credit.

3.6.4 Document Conventions

This document follows IEEE standard format and the conventions followed for fonts, formatting and naming are the standards followed in Computer Science and Engineering Department of Government Engineering College Thrissur.

3.6.5 Overall Description

This section mainly deals with the general factors of this package delivery system, connection between the system components and its requirements.

3.6.5.1 Product Perspective

It describes the overall structure of the system. The implementation of this system requires a central database which is to be maintained and regularly updated by the administrator. Both server to node and node to server communications are possible. The sender can register their package delivery details which are accepted and allocated to a particular rider based on the rider's travelling route and time details by administrator. The rider provides the service timely and efficiently. After completion of the delivery process the rider can report it to admin and the sender can convey their feedbacks online through a rating facility.

3.6.5.2 Product Functionality

The software consists of the following Modules

- Sender Module
- Rider Module
- Algorithm module

- Admin Module

Software helps to match the sender with a rider. Here we refer the person who wants to send a package to a place as sender and the rider is the person who is travelling to that destination in his vehicle at that time.

3.6.5.3 Users and Characteristics

This section outlines the use cases for each of the active users. The software mainly consists of two users:

1. Sender
2. Rider

The Sender module consists of:

- Sender login: Sender can register with the website and create an account. He can login with the username and password.
- Profile: Sender can update his profile page. It will also show all the past package deliveries
- Search : Sender can search for the Rider to sent his package
- Package details: Sender can list the details about package to be sent, like the dimension, weight, type of item in the package etc.
- Tracking: Sender can track the package through maps.
- Payment: All the payment for package delivery is listed. Sender needs to set up his bank to pay the payment.
- Sender logout: Sender can logout of the site whenever required

The Rider module consists of:

- Rider Login: Any person can register as a Rider. He needs to submit details about his vehicle and Driver's licence ID
- Profile : Rider can create the profile with the information about his, vehicle ,place etc
- Package delivery : Shows details about the riders past package deliveries and new delivery requests
- Payment: Shows payment details. Rider needs to set up his bank account to receive the payment
- Rider Logout : Rider can logout of site whenever required

3.6.5.4 Operating Environment

The software is developed to be operated in any web browser in any OS. There must be an active internet connection with sufficient bandwidth. For the android client application, an android running device with GPS and network functionality is required.

3.6.5.5 User Documentation

Users will be provided with user manuals which will include an overview of the product, complete specification of the software, technical details, backup procedure and contact information for any software related queries.

3.6.6 Specific Requirements

3.6.6.1 External Interface Requirements

User Interface

Users of the software mainly falls under three broad categories. Administrator, *Sender and *Rider. The interface for each class of users is different. The access and manipulation of overall data should be constrained purposefully to each class. All pages of the system are of a consistent theme and clear structure. The occurrence of errors should be minimized through the use of checkboxes, radio buttons and scroll down in registration form in order to reduce the amount of text input from user. The website which is the core interfacing medium for all these users could be built up using Bootstrap framework and jquery for frontend. Google maps API is used for all map tracking in the site. The backend runs on python-flask framework. RethinkDB database is used to store and retrieve information efficiently and fast.

Hardware Interface

Hardware interface is built in Operating system with essential hardware components such as microprocessors, basic input and output devices. The android application requires an android running Smartphone with network and GPS facility. The backend is hosted in a server with 24X7 internet connectivity and sufficient bandwidth.

Software Interface

Software development requires an active internet connection. The development requires python-flask framework, RethinkDB database, Android IDE, Nginx server.

3.6.6.2 Functional Requirements

Functionality: The system shall provide separate logins for each class of users

Usability: UDIO helps to match a sender with a suitable rider. It also helps to track the package and helps to automate the payment. The riders are rated based on their past deliveries. The package delivery cost is automatically calculated based on the distance and availability of riders. UDIO helps to match a sender with a suitable rider. It also helps to track the package and helps to automate the payment. The riders are rated based on their past deliveries. The package delivery cost is automatically calculated based on the distance and availability of riders.

3.6.7 Non Functional Requirements

3.6.7.1 Performance Requirement

The information provided by the software must be precise and accurate. The database shall be able to accommodate records of all the details of users. The software shall support use of multiple users at a time. The software must use Google Maps to provide shortest route map. The software must provide security.

3.6.7.2 Software Quality Attributes

The quality of the system is maintained in such a way so that it can be user friendly for all users. The software has the quality attributes such as adaptability, availability, reliability, security and maintainability.

Adaptability: The software should have the ability to adapt to evolving information inclusion requirements. This may be the case with user information or with product details.

Availability: The facilities provided by the software can be made available to all the users.

Reliability: The system is reliable as far as the network (constitute the central server and software users) and other hardware components work correctly.

Security: Each user has his own username and password for accessing the account. Only the authorised person can make changes in the database.

Maintainability: The application must be maintainable. The proper working of the software have to be constantly monitored and in case of software malfunctioning and failures, it should be corrected at the earliest.

Chapter 4

Design

4.1 Introduction

4.1.1 Purpose

This Software Design Document describes the Architectural and System design of this web application. The purpose of this document is to provide an overall description of the general design to be followed in implementing object. With the help of UML modelling we have structured the software design.

Intended Audience: This document is intended for those who want to use a fast and efficient system to send their packages and the riders who are ready to use their empty cargo space for delivering packages, securely with a credit.

4.1.2 Scope

The absence of an effective medium for transport of goods poses a day to day challenge for everyone, effective being the key aspect which requires the mode to be swift, accurate and secure. The existent system involves time consuming and tedious process. In today's world time is of the essence and definitely no one has time to spend on tasks like these. Our system creates a peer to peer network for package delivery. Instead of involving a third party delivery agency, we link the sender and receiver via a traveller who happens to be going on the same direction. The traveller would have already signed up with our programs by providing his/her credentials and he/she receives payment for each drop made. The client gets to dispatch the item through a larger pool of transporters and our end makes sure that the best one is selected. A GPS tracking system is also included to track the whereabouts of the package.

4.1.3 System Overview

This document is a description of the stages, states, test cases, classes, communication, and transitions, involved in the project. They have been explained in detail with the help of following diagrams:

- A use case diagram at its simplest is a representation of a user's interaction with the system that shows the relationship between the user and

the different use cases in which the user is involved. A use case diagram can identify the different types of users of a system and the different use cases and will often be accompanied by other types of diagrams as well. Use case diagrams are used to gather the requirements of a system including internal and external influences. These requirements are mostly design requirements. So when a system is analysed to gather its functionalities use cases are prepared and actors are identified.

- Class diagram if just a brief outline of all the classes involved, they are syntactically incomplete, and are to give just a brief outline of the structuring of the project. The class diagram is the main building block of object oriented modelling. It is used both for general conceptual modelling of the application, and for detailed modelling translating the models into programming code. Class diagrams can also be used for data modelling. The classes in a class diagram represent both the main objects, interactions in the application and the classes to be programmed.
- ER diagram is diagrammatic representation associated with the ER model. An ER model is an abstract way of describing a database. In the case of a relational database, which stores data in tables, some of the data in these tables point to data in other tables. ER diagrams actually, a graphical representation of entities and their relationships to each other, typically used in computing in regard to the organization of data within database or information systems. An entity is a piece of data-an object or concept about which data is stored. A relationship is how the data is shared between entities.

All the designs contained in this document were drawn using the tool ArgoUML and Microsoft Word.

4.2 Detailed Design

4.2.1 Use Case Diagram

A Use Case diagram at its simplest is a representation of a users interaction with the system that shows the relationship between the users and the different use cases in which the user is involved.

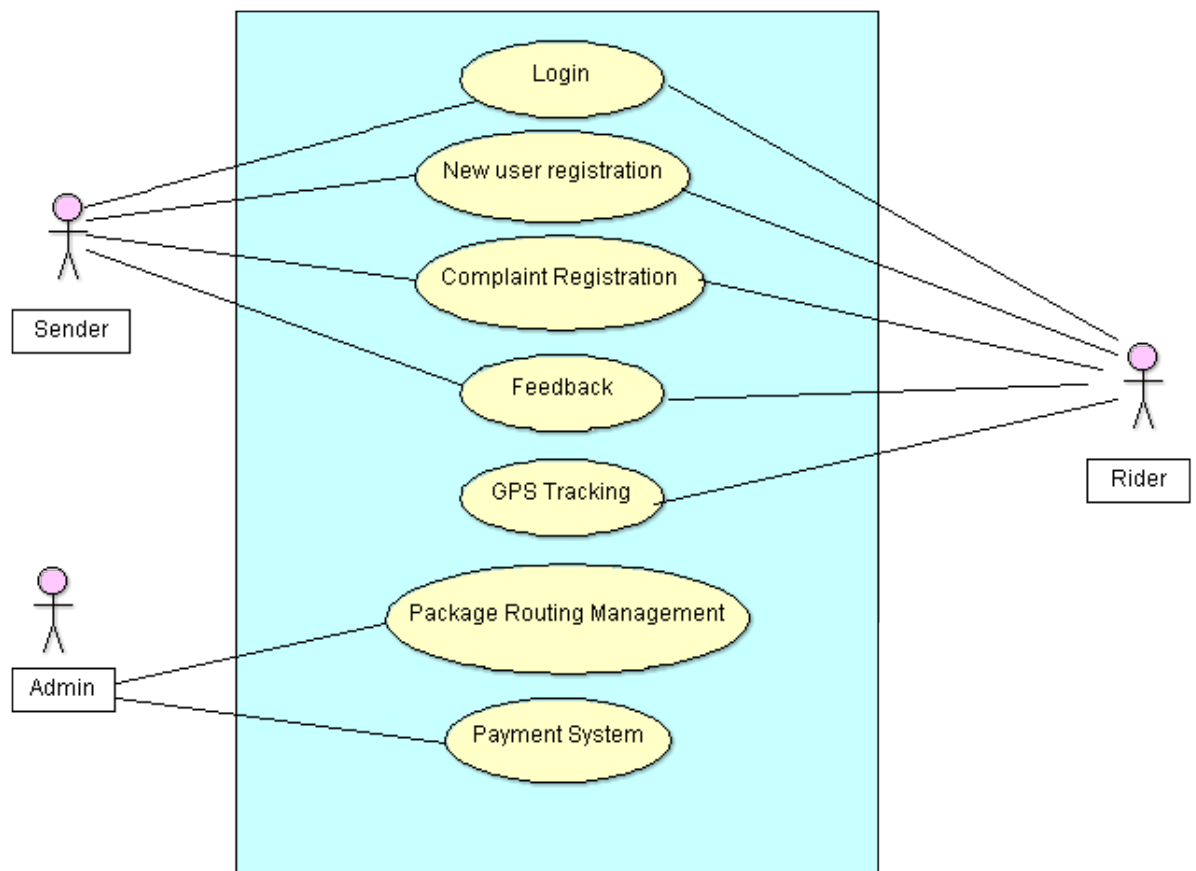


Figure 4.1: Use Case Diagram

4.2.2 Class Diagram

Class diagrams are visual representations of the static structure and composition of a particular system using the conventions set by the unified modelling language (UML). It describes the structure of the system by showing the systems classes, their attributes, operations (or methods) and the relationships among objects.

The following components will be a part of the proposed system:

- A database to store the information of the clients.
- A webpage which acts as an interface for the clients which provides them with necessary details.
- An application for the traveller to notify status and details.

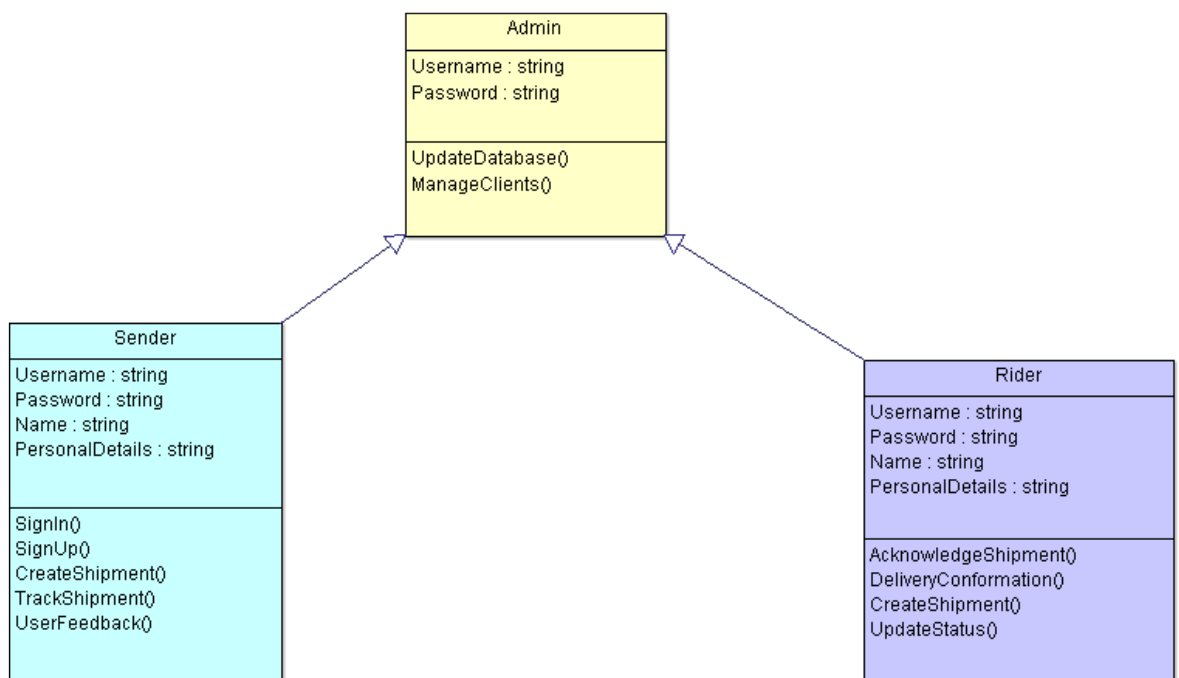


Figure 4.2: Class Diagram

4.2.3 Entity Relationship Diagram

In software engineering, an entity relationship model (ER model) is a data model for describing the data or information aspects of a business domain or its process requirements, in an abstract way that lends itself to ultimately being implemented in a database such as a relational database. The main components of ER models are entities (things) and the relationships that can exist among them. An ER diagram is a logical representation of an organisation data, and consists of three primary components:

- **Entities:** Major categories of data and are represented by rectangles
- **Attributes:** Characteristics of entities and are listed within entity rectangles
- **Relationships:** Business relationships between entities and are represented by lines

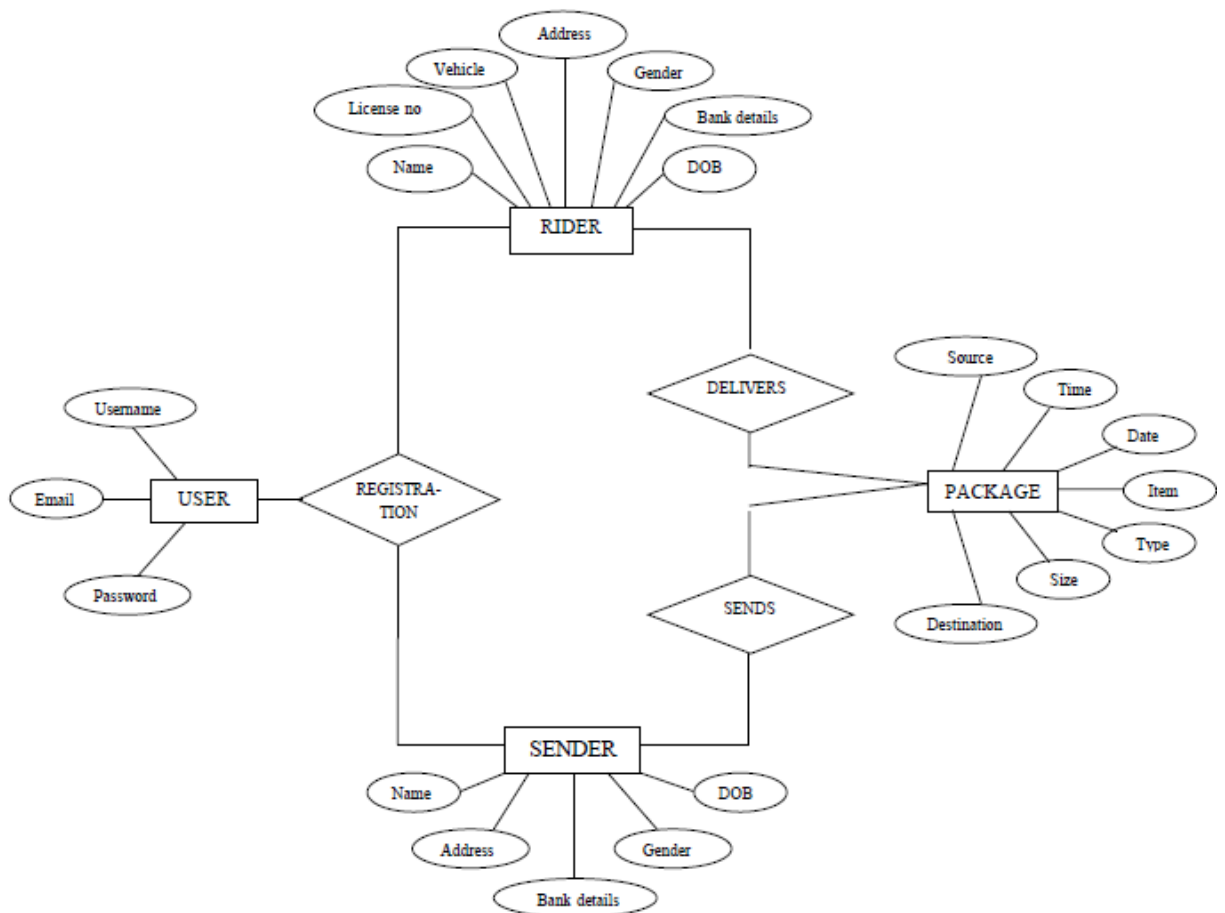


Figure 4.3: ER Diagram

4.2.4 User Interface Design

The user interface for the web application is being designed as follows:

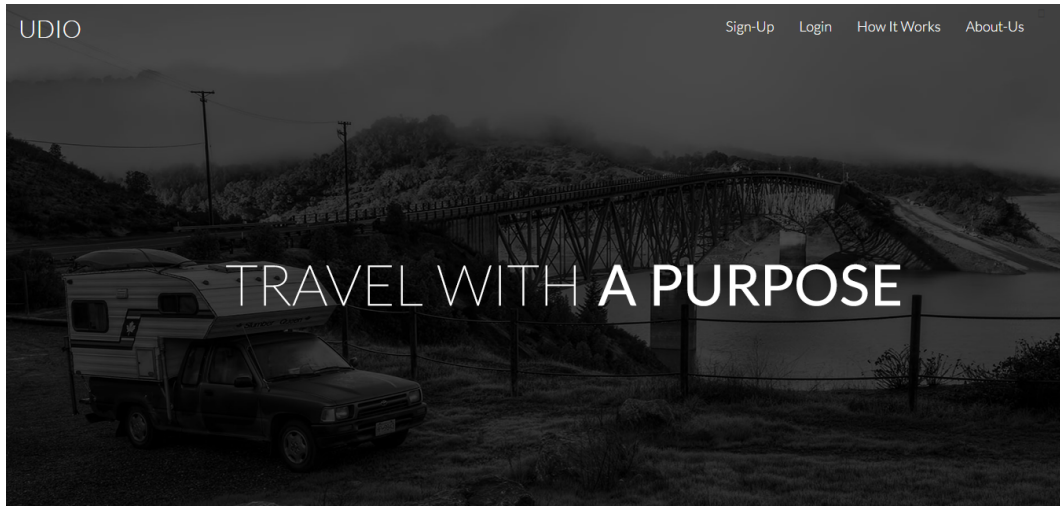


Figure 4.4: Home Page

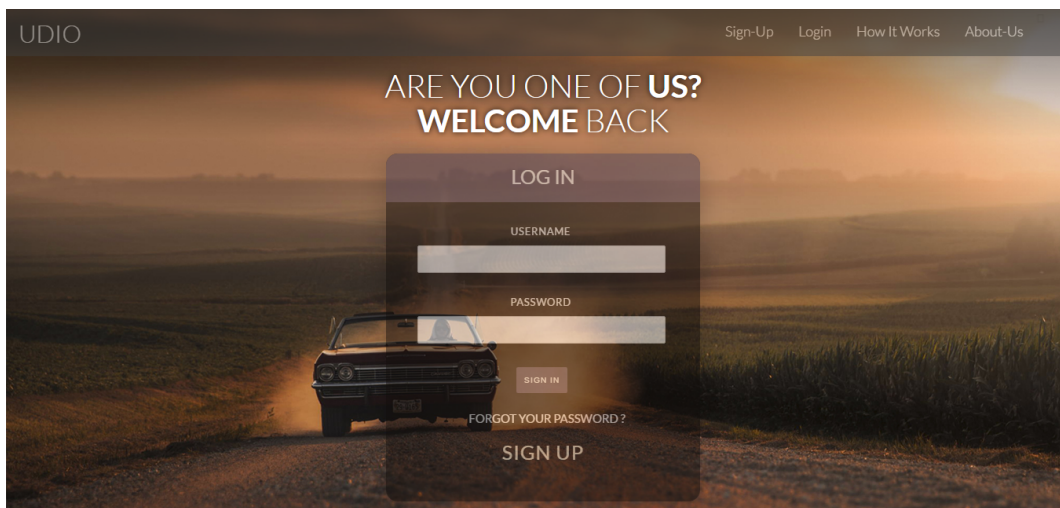
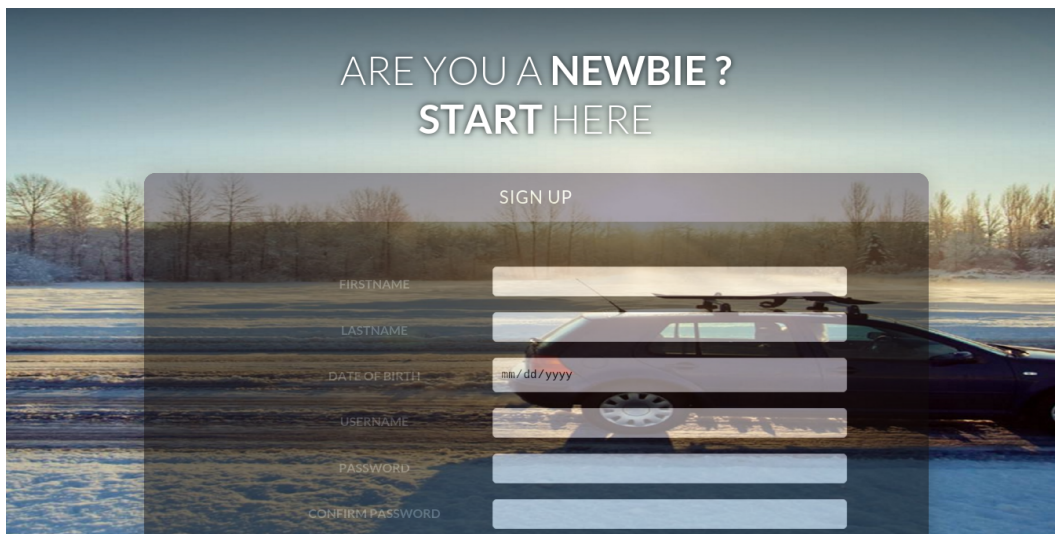


Figure 4.5: Sign In Page



ARE YOU A **NEWBIE** ?
START HERE

SIGN UP

FIRSTNAME

LASTNAME

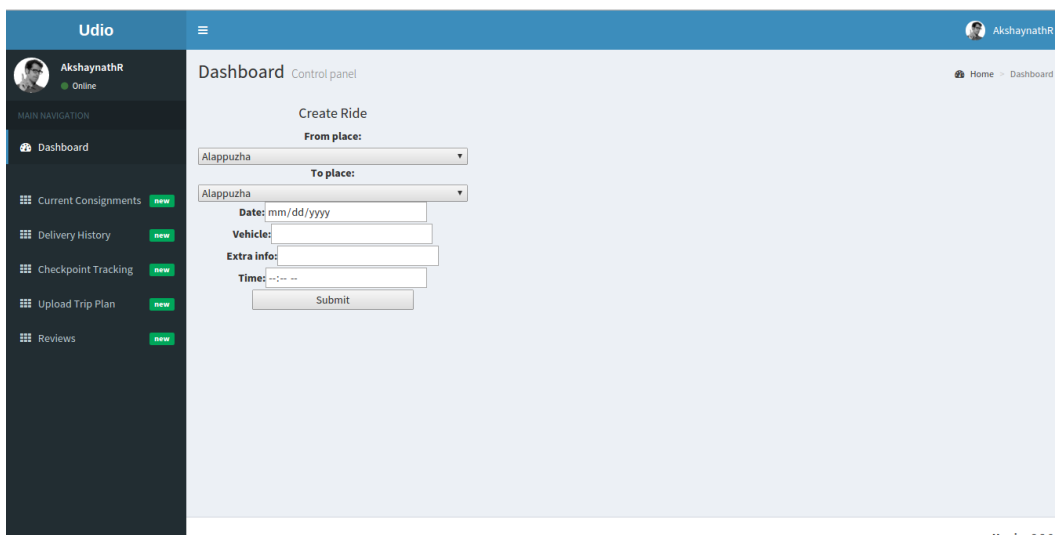
DATE OF BIRTH

USERNAME

PASSWORD

CONFIRM PASSWORD

Figure 4.6: Sign Up Page



Udio

AkshaynathR Online

MAIN NAVIGATION

- Dashboard
- Current Consignments **new**
- Delivery History **new**
- Checkpoint Tracking **new**
- Upload Trip Plan **new**
- Reviews **new**

Dashboard Control panel

Create Ride

From place:

To place:

Date:

Vehicle:

Extra info:

Time:

Submit

Home > Dashboard

Version 2.3.0

Figure 4.7: Dashboard: Rider

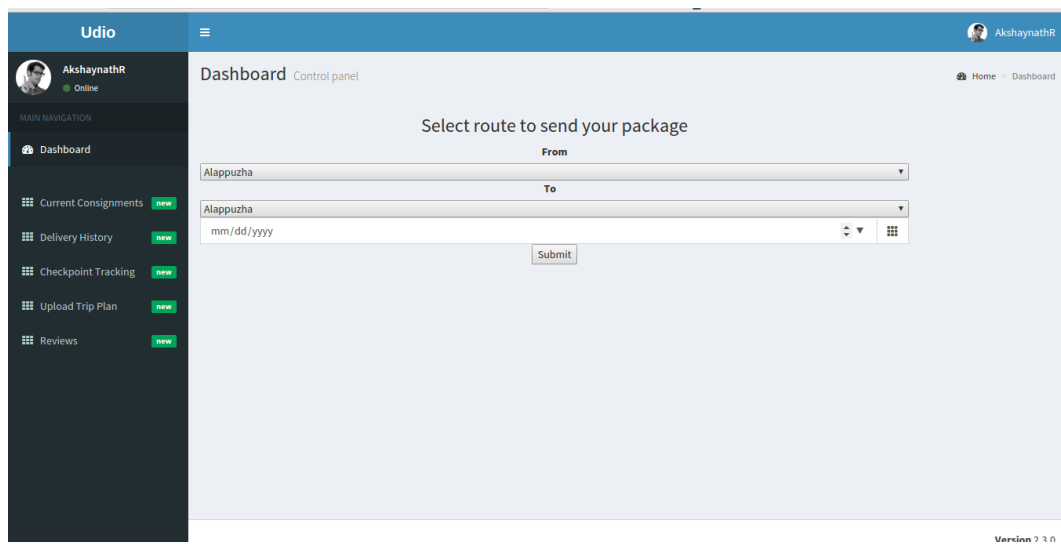


Figure 4.8: Dashboard: Sender

Chapter 5

Coding

5.1 Register

The function register() , registers the user by accepting all necessary information and sets a username and password.

```
def register()
{
    if(http 'GET' request)
        return{webpage for registration}

    if(http 'POST' request)
    {
        //read all values required for registration
        like firstname,lastname etc.

        //check username duplicates
        if(duplicate found)
        {
            return{"error: username already exists"}
        }
        else
        {
            //insert to database
        }

        r.db('VDIO').table('user').insert([
        {
            'firstname':firstname,
            'lastname':lastname,
            'address':"address",
            'username':username,
            'password':password,
```

```
        'country':country
        ..
        ..
    ]]).run(connection)

    close database connection

    return (signup page)
}
}
```

5.2 Login

The function login(), logs-in the user by verifying the username and password provided.

```
def login()
{
    if(http request is 'GET')
    {

        if 'user' in session //check session for user
            return redirect(webpage for dashboard)
        else
            return(login page)
    }

    if(http request is 'POST')
    {
        username=read username input
        password= read password input

        //if username or password is empty
        return("error:username or password empty")

        //check if user already exists

        //create session
        session['user']=user

        //if user already exists:
        check password
        if True:
            authenticate()
        else: return("authentication error")
    }
}
```

```
    }  
}
```

5.3 Sender

The function `findrider()`, lists out all riders commuting between the source and destination at a given time.

```
//function to find rider travelling to a place  
def find_rider()  
{  
  
    if(http 'GET' request)  
        {return (webpage for rider)}  
  
    if(http'POST'request)  
    {  
        // Reads the input to_place  
        to_place = request.form['to_place']  
  
        // reads the input from_place  
        from_place = request.form['from_place']  
  
        create database connection()  
  
        //filter data from database  
        data=r.db('UDIO').table('ride').filter(  
            {'from'.from_place}).run(conn)  
  
        return data  
    }  
}  
  
def home()  
{  
    //function to sender homepage  
  
    if(http request is 'GET')  
        return(home/udio.html)  
}
```


5.4 Rider

The function `tripplan()`, is used to record the details of the trip, like place date, vehicle, time etc, uploaded by the rider.

```
def tripplan()
{
    if(http request is 'GET')
    {
        //check if user is logged in
        //if user is not in section
        {redirect to login page}

    }

    if(http request is 'POST')
    {
        read values like to_place,date,vehicle,time..

        create_connection()

        //insert the data in database & rider table
        ride=r.db('UDIO').table('rider').insert([
        {
            'to_place':to_place,
            'licence':licenseno,
            'from_place':from_place,
            ..
            ..
        }]).run(conn)

        return {ride created page}
    }
}
```

Chapter 6

Testing

6.1 Types of Testing done

6.1.1 Whitebox Testing

Whitebox testing (also known as clear box testing, glass box testing, transparent box testing, and structural testing) tests internal structures or workings of a program, as opposed to the functionality exposed to the end user. In whitebox testing an internal perspective of the system, as well as programming skills, are used to design test cases. The tester chooses inputs to exercise paths through the code and determine the appropriate outputs. This is analogous to testing nodes in a circuit, e.g. InCircuit Testing (ICT).

While whitebox testing can be applied at the unit. Integration and system levels of the software testing process, it is usually done at the unit level. It can test paths within a unit, paths between units during integration, and between subsystems during a system level test. Though this method of test design can uncover many errors or problems, it might not detect unimplemented parts of the specification or missing requirements.

Techniques used in whitebox testing include:

1. API testing (application programming interface) testing of the application using public and private APIs.
2. Code coverage creating tests to satisfy some criteria of code coverage (e.g. the test designer can create tests to cause all statements in the program to be executed at least once).
3. Fault injection methods intentionally introducing faults to gauge the efficiency of testing strategies.
4. Mutation testing methods.
5. Static testing methods.

Code coverage tools can evaluate the completeness of a test suite that was created with any method, including black box testing. This allows the software team to examine parts of a system that are rarely tested and ensures that the most important function points have been tested. Code coverage as a software metric can be reported as a percentage for:

- Function coverage, which reports on functions executed
- Statement coverage, which reports on the number of lines executed to complete the test 100 percent statement coverage ensures that all code paths, or branches (in terms of control flow) are executed at least once. This is helpful in ensuring correct functionality, but not sufficient since the same code may process different inputs correctly or incorrectly.

Sr No	Modules	Function	Expected Output	Whether obtained the expected output or not
1	Register	Registration()	Checks duplicate users while registration. Registers a new user.	Yes
2	Login	Login()	Login system for users.	Yes
3	Sender	Sender()	Finds the riders available in a particular route. Sends a ride request. Track location of each rides.	Yes
4	Rider	Rider()	Creates a new ride. Remove the ride from listing.	Yes

Table 6.1: Whitebox testing

6.1.2 Blackbox Testing

The technique of testing without having any knowledge of the interior workings of the application is Blackbox testing. The tester is oblivious to the system architecture and does not have access to the source code. Typically, when performing a black box test, a tester will interact with the systems user interface

by providing inputs and examining outputs without knowing how and where the inputs are worked upon.

After the completion of implementation, we turned to Blackbox testing and thus hosted a demo version of interpreter for extensive testing.

Sr No	Modules	Expected Output	Whether obtained the expected output or not
1	Register	Checks duplicate users while registration. Registers a new user.	Yes
2	Login	Login system for users.	Yes
3	Sender	Finds the riders available in a particular route. Sends a ride request. Track location of each rides.	Yes
4	Rider	Creates a new ride. Remove the ride from listing.	Yes

Table 6.2: Blackbox testing

6.2 Advantages and Limitations

6.2.1 Advantages

- Open Source Software .
- Web compatible application .
- Universal system compatibility
- Low cost of implementation
- Universal availability

6.2.2 Limitations

- One entry in the list
- No offline support

6.3 Future Extensions if possible

The future enhancements of the proposed system that we have estimated are given below:

- Including artificial intelligence to improve the matching algorithm
- Cross platform implementation of the Application
- Extending payment options
- Improving the User Interfaces according to user feed-backs

Chapter 7

Quality Assurance

7.1 Introduction

Software quality assurance (SQA) consists of a means of monitoring the software engineering processes and methods used to ensure quality. SQA encompasses the entire software development process, which includes processes such as requirements definition, software design, coding, source code control, code reviews, change management, configuration management, testing, release management, and product integration. SQA is organised into goals, commitments, abilities, activities, measurements, and verifications.

7.1.1 Purpose

The aim of this project is to create a peer to peer network for package delivery. It is achieved by routing the package in an optimized way. Instead of involving a 3rd party agency, the sender and receiver are linked via a traveler who happens to be going on the same way. The traveler would have already signed up without program by providing his/her credentials and he/she receives payment for each drop made.

7.1.2 Scope

Every member in our team is responsible for the actions planned in this document such as documenting the results throughout the development of the project, reviewing the project progress, and testing the project quality, controlled by this plan. The following are the portions of the software lifecycle that are covered by the SQAP: **User requirements, Software requirements, Design, Implementation and Testing.**

The list of software items to be covered in each of the above mentioned lifecycle phases are given below:

Software Lifecycle Phase	Software Item
User requirements	User requirement document
Design	Software Design Document
Implementation	Document including template of functions
Testing	Document showing testing done in project with summary results

Table 7.1: Software Lifecycle/items

The Software Quality Assurance Plan covers the software items. In addition, the SQAP is also covered by this quality assurance plan.

7.2 Quality Assurance Strategy

To assure quality of software deliverable in each software development phase, we will use the test factor/test phase matrix. The matrix has two elements. Those are the test factor and the test phase. The risks coming from software development and the process for reducing the risks should be addressed by using this strategy. The test factor is that the risk or issue which is needed to be tackled, and the test phase is that the phase of software development which conducts the test. The matrix should be customised and developed for each project. Thus, we will adapt the strategy to our project through four steps.

In the first step, we will select the test factors and rank them. The selected test factors such as reliability, maintainability, portability or etc, will be placed in the matrix according to their ranks.

The second step is for identifying the phases of the development process. This phase should be recorded in the matrix. The last step is that deciding the test phase of addressing the risks. In this step, we will decide that which risks will be placed at each development phase.

The matrix forms a part of the quality assurance strategy and as mentioned above this matrix would be used in each of the project lifecycle phases to identify the risks associated in each of the phases with respect to the testing factors. The risks would also be accompanied with their mitigation strategies and in case the risk materialised into a problem, the respective mitigation would be applied. It is for these reasons, that a mention is made about the matrix here in a separate section of the document and not mixed with other sections of the document to avoid repetition.

Testphase/ Test	Requirements	Design	Coding	Testing factors
Correctness	Risk: The SRS may not be correct as per the goals of the SQAP Strategy: Formal Techview of SRS	Risk: Software design document may not be correct as per the SRS Strategy: Formal Techview of SRS	Risk: For lengthy code functions defined in software design document yearn more for code correctness	Risk : User may not give the correct input
Performance	Risk: User may have to compromise	Risk: Software design document may not have the performance as per the requirement	Risk: Lack of syntactic structures in selected language selected	Risk: May have several input cases which affect the performance
Continuity of processing	Risk: Unsatisfiable requirements	Risk: Improper way of designing requirements	Risk: Possibility of error occurrences	Risk: Possibility of error occurrences

Table 7.2: Quality Assurance Strategy

7.3 Audits and Reviews

7.3.1 Work Product Reviews

Work Product	When reviewed by Quality Assurance	How reviewed by Quality Assurance
Software requirement specification	After a new release	All the requirements raised by the user are achieved.
Software document design	After modification	Desktop application developed. Development done in Python.
Coding	New release	All algorithm and designs coded and implemented perfectly.
Testing	New release	The product is subjected to both black-box testing and white box testing. All user requirements are met.

Table 7.3: Work Product Reviews

7.4 Further Extension

We have completed our project in all aspects. All functional and non-functional requirements of all seven modules are met. The source code was made and compiled in Linux. The product is easily usable and user documentation for all the

modules is provided. The system can be extended by adding more features. The project has gone successfully through all the phases of software development and has been tested per the requirements.

Further Extensions include :

- Cross platform implementation of the Application.
- Improving the User Interfaces according to user feed backs.
- Extending the web application for more browsers.

Chapter 8

Conclusion

The project works has been completed through the agile development life cycle. On our way working on this interesting project, we learned many things. We got an invaluable experience by working on this project and learned various technologies on a need based manner. We believe that the system we developed is an improvement over the current scenario. We were able to incorporate latest technological advancements to ease the working of the logistics management. We intend to extend the project to improve user functionality and ease of use. We hope that our project had an impact on the intended fields.

Bibliography

- [1] Daryl K. Harms and Kenneth M. McDonald. *The Quick Python Book, Second Edition*. Manning Publications, 2010.
- [2] Thomas A. Powell. *HTML & CSS: The Complete Reference, Fifth Edition*. McGraw-Hill Osborne, 2010.
- [3] <http://www.tutorialspoint.com/python/>.
- [4] <http://stackoverflow.com/>, question and answer site for professional and enthusiast programmers.
- [5] John Pollock. *Javascript, A Beginner's Guide*. McGraw Hill Education (India), 2009.

Appendices

Appendix A

User Document

A.1 Using the Web Application

- Open browser and navigate to "www.udio.in"

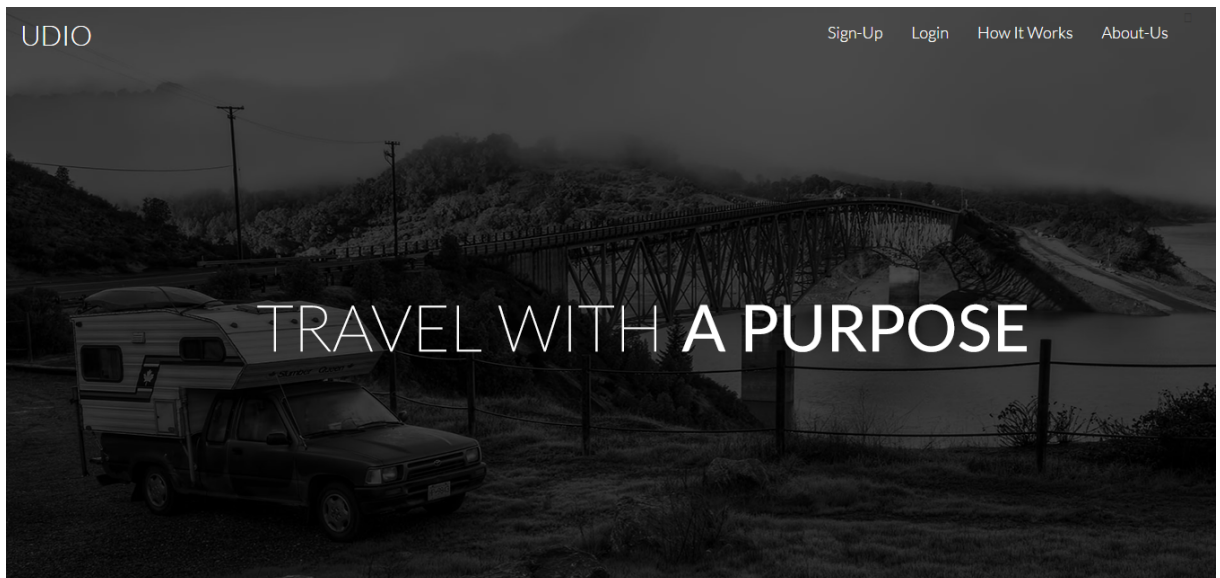


Figure A.1: Home Page

- Click on Sign In/Sign Up

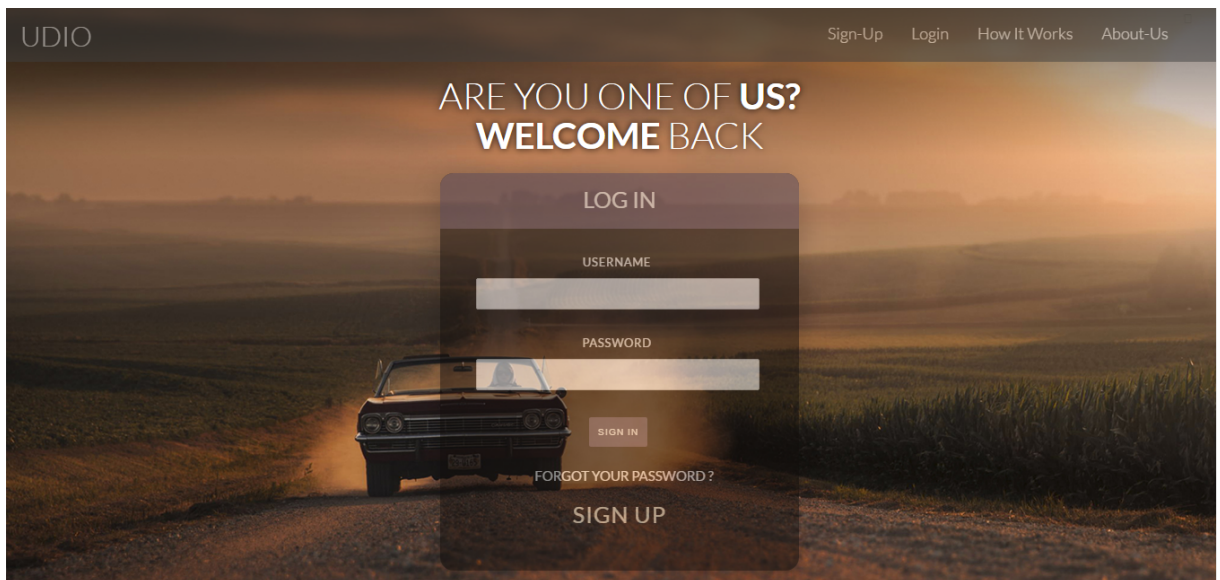
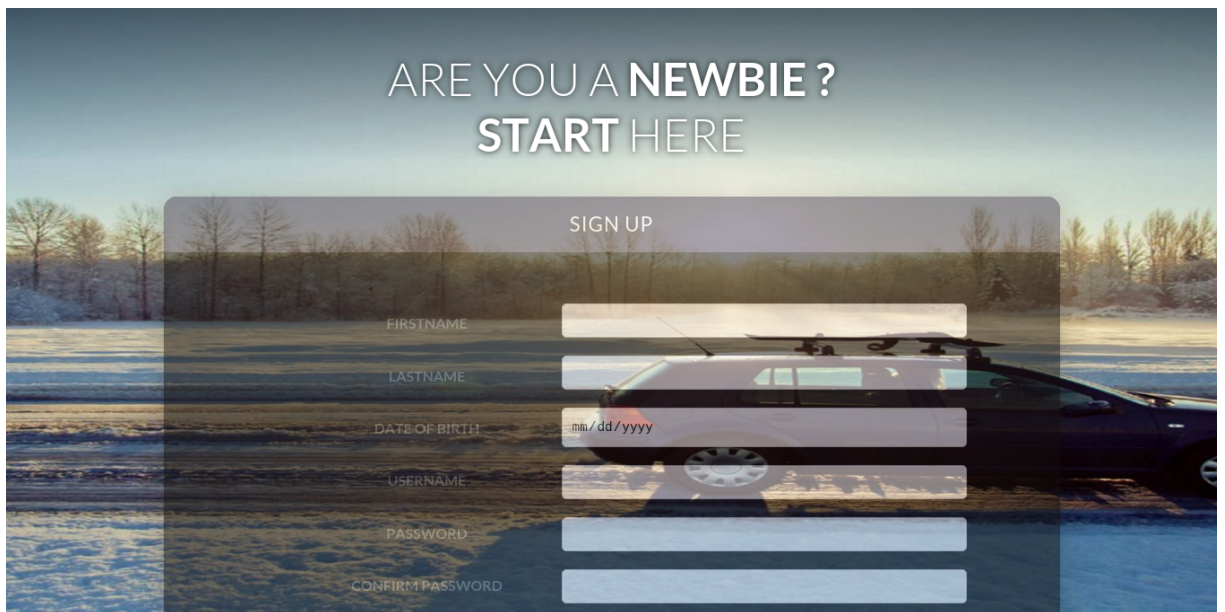


Figure A.2: Sign In

- Click on SIGN UP to register



ARE YOU A **NEWBIE ?**
START HERE

SIGN UP

FIRSTNAME

LASTNAME

DATE OF BIRTH

USERNAME

PASSWORD

CONFIRM PASSWORD

Figure A.3: Sign Up

- Ride creation for riders

The screenshot displays the 'Udio' web application interface. On the left is a dark sidebar with a user profile for 'AkshaynathR' (Online) and a 'MAIN NAVIGATION' menu containing: Dashboard, Current Consignments (new), Delivery History (new), Checkpoint Tracking (new), Upload Trip Plan (new), and Reviews (new). The main content area is titled 'Dashboard' with a subtitle 'Control panel'. It features a 'Create Ride' form with the following fields: 'From place:' (dropdown menu showing 'Alappuzha'), 'To place:' (dropdown menu showing 'Alappuzha'), 'Date:' (text input with placeholder 'mm/dd/yyyy'), 'Vehicle:' (text input), 'Extra info:' (text input), and 'Time:' (text input with placeholder '--:-- --'). A 'Submit' button is located below the 'Time' field. The top right of the dashboard shows a breadcrumb trail 'Home > Dashboard' and the user's name 'AkshaynathR'. The bottom right corner of the page indicates 'Version 2.3.0'.

Figure A.4: Create Ride

- New package creation for senders

The screenshot displays the Udio web application interface. On the left is a dark sidebar with the user's profile (AkshaynathR, Online) and a main navigation menu with options: Dashboard, Current Consignments, Delivery History, Checkpoint Tracking, Upload Trip Plan, and Reviews. The main content area is titled 'Dashboard' and 'Control panel'. It features a form titled 'Select route to send your package' with two dropdown menus for 'From' and 'To', both currently set to 'Alappuzha'. Below these is a date input field labeled 'mm/dd/yyyy' and a 'Submit' button. The bottom right corner of the page indicates 'Version 2.3.0'.

Figure A.5: Create Package