INTRODUCTION

1.INTRODUCTION

Smart Parcel Delivery Network is an innovative web-based platform designed to connect parcel senders with travelers willing to deliver parcels along their planned routes. In the traditional logistics industry, courier services often involve high costs, long delivery times, and limited flexibility, making them inefficient for individuals who need a more affordable and faster alternative. This project addresses these challenges by leveraging peer-to-peer parcel delivery, allowing travelers who are already on the move to carry and deliver parcels for senders. The system not only offers a cost-effective solution for senders but also provides travelers with an opportunity to earn money for utilizing their available space while traveling.

The platform ensures a seamless user experience for both senders and travelers. Senders can easily create an account, post parcel details including pickup and delivery locations, weight, dimensions, and a payment offer. On the other hand, travelers can browse through the list of available parcels and filter them based on their route, parcel size, and weight preferences. Once a traveler selects a parcel for delivery, they initiate the pickup process and keep both the sender and recipient updated through the platform.

To ensure trust and security, the system integrates a secure payment mechanism. The sender's payment is held in escrow until the parcel is successfully delivered and confirmed by the receiver. This approach guarantees that payments are processed only upon successful completion of the delivery, reducing the risk of fraud or lost parcels.

Additionally, real-time tracking and status updates enhance transparency and reliability. Travelers can update the parcel status at various stages, such as “Picked Up,” “In Transit,” and “Delivered,” keeping senders and recipients informed throughout the journey. The platform also includes notification alerts, ensuring that both parties receive timely updates regarding the parcel's progress. By utilizing existing travel routes, this project minimizes reliance on traditional courier services, making parcel delivery more accessible, affordable, and efficient. It offers a sustainable logistics solution by reducing empty travel capacity and promoting resource optimization. The platform not only benefits individual users but also contributes to a more flexible and decentralized logistics network, empowering communities to participate in a mutually beneficial parcel delivery system.

In summary, this project bridges the gap between senders and travelers, transforming parcel delivery into a smarter, more cost-effective, and convenient process for everyone involved.

1.1 ORGANIZATION PROFILE

In today's rapidly evolving business landscape, CubikSoft Technologies integrates innovation with technology to help organizations maximize business value, enhance productivity, and improve customer satisfaction. The company understands the unique challenges businesses face and provides end-to-end technology solutions, including IT infrastructure management, software development, cloud solutions, and digital transformation.

CubikSoft Technologies follows a cost-effective, scalable, and innovative approach, tailoring solutions to meet diverse customer requirements. By adopting industry best practices in technology, processes, and service delivery, the company ensures maximum client satisfaction and long-term business growth.

CORPORATE OVERVIEW

CubikSoft Technologies is a forward-thinking technology company founded with the vision of empowering businesses through digital innovation. The company comprises a team of experienced professionals specializing in software development, IT consulting, and enterprise solutions, ensuring that clients receive cutting-edge, customized solutions that enhance efficiency and profitability.

With extensive industry experience, CubikSoft Technologies excels in designing, deploying, and managing scalable and secure IT infrastructures. Its customer-centric approach and commitment to technical excellence have enabled the company to establish long-term partnerships across multiple industries.

SOLUTIONS

CubikSoft Technologies provides end-to-end solutions designed to help businesses grow, innovate, and stay ahead in a competitive market. With a team of skilled professionals, the company ensures that projects are completed as requested and, most importantly**,** on time.

CubikSoft Technologies offers a wide range of services, including

- Digital Marketing
- Engineering Services
- Testing Services
- Web Development

1.2 SYSTEM SPECIFICATION

1.2.1 HARDWARE CONFIGURATION

PROCESSOR : : Intel Core i5

RAM : 16 GB DDR4

STORAGE : 1 TB SSD

User Device Hardware

- **Smartphones/Tablets**

  PROCESSOR : Quad-core (1.8 GHz or higher)

  RAM : 2 GB

  STORAGE : 32 GB

- **PC/Laptop**

  PROCESSOR : Intel Core i5 or equivalent (4-core, 2.5 GHz or higher)

  RAM : 8 GB DDR4

  STORAGE : 500 GB HDD or 256 GB SSD

1.2.2 SOFTWARE CONFIGURATION

OPERATING SYSTEM : Windows 11

WEB BROWSER : Google Chrome (latest version)

DATABASE : MySQL 8.0 for data storage

BACKEND FRAMEWORK : Spring Boot (3.4.2), Java (JDK 17).

FRONTEND : React.js (Version 19), HTML/CSS/JavaScript

PAYMENT INTEGRATION : PayPal

AUTHENTICATION : JWT for secure login .GEOLOCATION & MAP : Google Maps API for route tracking

2.SYSTEM STUDY

2.1 EXISTING SYSTEM

The current parcel delivery system mainly relies on traditional courier services, which handle the shipping and delivery of parcels. Users must visit courier offices, complete paperwork, and pay for the service in person. The services typically offer limited delivery options, a lack of flexible scheduling, and sometimes long delivery times. Tracking is often not available in real-time, and communication between senders and recipients is limited to updates provided by the courier.

2.1.1 DRAWBACKS

- Inefficient Processes: Manual paperwork and physical office visits make the system time-consuming.
- High Costs: Courier services charge high fees, making parcel delivery expensive.
- Limited Flexibility: Travelers can't participate in deliveries, missing out on potential earnings.
- Lack of Real-Time Tracking: Traditional services offer limited tracking, reducing transparency.
- Inefficient Delivery: The system doesn’t leverage travelers, missing cost-effective delivery opportunities.

2.2 PROPOSED SYSTEM

This system aims to revolutionize parcel delivery by connecting senders with travelers who are already on their journeys. Instead of relying on traditional courier services, which can be costly and inefficient, this platform allows senders to post parcels online, providing details like size, weight, and locations. Travelers can browse these listings, select parcels that align with their route, and deliver them while earning money along the way.

The platform ensures seamless interaction between senders and travelers by offering real-time tracking and notifications. This allows both parties to stay updated on parcel status, enhancing transparency and communication. By utilizing travelers as an additional delivery resource, this solution reduces delivery costs and adds flexibility, making parcel delivery more efficient and accessible.

2.2.1 FEATURES

- User Registration & Authentication – Secure account creation and login for senders and travelers.
- Parcel Posting & Listing – Senders can upload parcel details, and travelers can browse available deliveries.
- Search & Filter Options – Travelers can search for parcels based on location, size, weight, and price.
- Payment Integration – Secure transactions via PayPal with an escrow system.
- Notifications & Alerts – Real-time updates for senders and travelers on parcel status.
- Delivery Confirmation – Travelers receive payment only after the sender confirms successful delivery.
- User Ratings & Reviews – Senders and travelers can rate and review each other to ensure trust and reliability.

SYSTEM DESIGN AND DEVELOPMENT

3.SYSTEM DESIGN AND DEVELOPMENT

3.1 FILE DESIGN

The system has been designed with following file designs

Backend (Spring Boot - Java)

- CourierApplication.java
- Controller.java
- User.java
- TravellerDetail.java
- ParcelDetail.java
- Parcelrepo.java
- Travellerrepo.java
- Userrepo.java
- Service.java
- ServiceImplementation.java
- GlobalExceptionHandler.java

CourierApplication.java

This file is the main entry point of the Spring Boot application. It initializes and starts the application, setting up the necessary components and configurations. Think of it as the application's "on" switch.

Controller.java

This file acts as the request handler. It receives incoming web requests (like user signup, login, adding a parcel), processes them using the Service layer, and sends back responses to the user. It's the interface between the outside world and your application's logic.

User.java

This file defines the structure of a "User" in your system. It contains information about each user, such as their name, email, mobile number, and password. It's a blueprint for how user data is stored.

TravellerDetail.java

This file stores additional information about users who are also travelers, like their Aadhar number and license details. It extends the basic "User" information with traveler-specific data.

ParcelDetail.java

This file describes the details of a parcel, including its dimensions, weight, price, sender and receiver addresses, and status. It's the blueprint for how parcel data is stored.

Parcelrepo.java

This file provides the tools for interacting with the database for ParcelDetail objects. It allows you to perform actions like saving a parcel, retrieving parcels, and searching for parcels. It's the data access layer for parcels.

Travellerrepo.java

This file provides the tools for interacting with the database for TravellerDetail objects. It allows you to save, retrieve, and search traveler information. It's the data access layer for travelers.

Userrepo.java

This file provides the tools for interacting with the database for User objects. It allows you to save, retrieve, and search user information. It's the data access layer for users.

Service.java

This file defines the services your application provides, like user signup, login, adding a parcel, and registering a traveler. It's the contract that specifies what actions your application can perform.

ServiceImplementation.java

This file contains the actual code that implements the services defined in Service.java. It contains the logic for each service operation, like validating user data, interacting with the database, and handling errors. It's where the "work" gets done.

GlobalExceptionHandler.java

This file handles errors that occur in your application. It provides a central place to manage exceptions and return user-friendly error messages. It's the application's error management system.

Frontend File Structure (React js)

- Navbar.js
- ParcelSelection.js
- TravelerRegistration.js
- ParcelForm.js
- ParcelTracking.js
- Notification.js

- Dashboard.js
- Login.js
- Signup.js
- ParcelListing.js
- App.css
- Navbar.css
- ParcelSelection.css
- Dashboard.css
- App.js

Components

- Navbar.js: Provides site-wide navigation, allowing users to easily access different sections of the platform (e.g., dashboard, parcel listing, etc.).
- ParcelSelection.js: Enables travelers to browse and choose parcels they wish to deliver. Displays parcel details and selection options.
- TravelerRegistration.js: A form for travelers to register their information, including name, email, license details, and other required credentials.
- ParcelForm.js: A form for senders to enter the details of the parcel they want to send, such as dimensions, weight, destination, and other relevant information.
- ParcelTracking.js: Displays the current status and location of a parcel, allowing users to track its journey in real time.
- Notification.js: Displays updates and alerts related to parcel status changes, delivery confirmations, and other important notifications.

Pages

- Dashboard.js: Provides a personalized overview of the user's account, including active parcels, ongoing tasks, recent activity, and other relevant information.
- Login.js: Allows users to securely log in to the platform using their email address and password.
- Signup.js: A registration form for new users to create accounts, specifying whether they are registering as a sender or a traveler.
- ParcelListing.js: Displays a list of all available parcels that have been posted by senders and are ready for travelers to pick up.

Styles

- App.css: The primary stylesheet containing global styles applied to the entire application.
- Navbar.css: Styles specific to the Navbar component, controlling its appearance and layout.
- ParcelSelection.css: Styles for the ParcelSelection component, defining how the parcel selection interface looks.
- Dashboard.css: Styles for the Dashboard page, managing its layout, colors, and other visual elements.

Other

- App.js: The main application file. It sets up routing, connecting different pages and components, and acts as the entry point for the frontend application.

3.2 INPUT DESIGN

Input design ensures accurate and efficient data entry in the SwiftBridge platform. It is structured to minimize errors, enhance user experience, and maintain data integrity. The system accepts inputs from both Parcel Senders and Travelers through various forms and interactive components.

User Registration Form

This form allows users to create an account, ensuring identity verification and secure login.

- Name (Text)
- Email ID (Email Input)
- Mobile Number (Numeric Input)
- Password (Password Input)
- Address (Text)
- Aadhar Card / Driver's License (File Upload for Travelers)

Parcel Posting Form

Senders use this form to provide parcel details for travelers to view and select.

- Parcel Image (File Upload)
- Parcel Dimensions (Text/Numeric Input)
- Parcel Weight (Numeric Input)
- Price or Payment Offer (Numeric Input)
- Pickup Location (Address Selector)
- Delivery Location (Address Selector)

Parcel Search and Selection Form

Travelers can browse available parcels and select one for delivery.

- Pickup Location Filter (Dropdown or Address Selector)
- Drop Location Filter (Dropdown or Address Selector)
- Parcel Size (Dropdown Selection)
- Parcel Weight (Dropdown Selection)
- Select Parcel (Button Click)

Tracking and Status Update Form

This form enables travelers and senders to update and track the status of a parcel.

- Status Selection (Dropdown: Picked Up, In Transit, Delivered)
- Delivery Confirmation (Button Click: Confirm Delivery)

3.3 OUTPUT DESIGN

Output design refers to the way information is displayed to users. In SwiftBridge**,** the output screens provide critical information such as parcel details, user details, transaction statuses, and tracking updates. Below are the key output screens and their functions:

Dashboard Screen (User Dashboard)

What the Screen Displays

- User information (Name, Email, Mobile, Address)
- List of posted parcels with details:
    - Parcel Image
    - Dimensions, Weight, Price
    - Pickup & Delivery Locations
    - Current Status (Posted, Selected, In Transit, Delivered)
- Buttons for Uploading a Parcel or Selecting a Parcel (if the user is a traveler)

Functions

- Display user profile details
- List all parcels posted by the user
- Show parcel statuses (e.g., Posted, Selected, Delivered)
- Allow navigation to Upload Product or Traveler Registration
- If a parcel is selected, display traveler details

Upload Product Screen

What the Screen Displays

- Input fields for Parcel Details (Dimensions, Weight, Price, Pickup & Delivery Address)
- Image upload section
- Submit button to upload the parcel

Functions

- Allow users to input parcel details
- Validate required fields
- Handle image uploads
- Submit parcel data to the backend

Parcel Selection Screen (For Travelers)

What the Screen Displays

- List of available parcels that match the traveler's location
    - Parcel Image, Dimensions, Weight, Price
    - Pickup & Delivery Address
- Select button for choosing a parcel

Functions

- Display parcels available for travelers
- Allow travelers to select a parcel for delivery
- Update parcel status to 'Selected'

Login Screen

What the Screen Displays

- Input fields for Email and Password
- Login button
- Sign-up link for new users

Functions

- Authenticate users
- Redirect to the dashboard on successful login
- Show error messages for incorrect credentials

Traveler Registration Screen

What the Screen Displays

- Input fields for Aadhar Card / Driver's License
- Submit button to complete traveler registration

Functions

- Allow users to register as a traveler
- Validate and store traveler details
- Redirect to Parcel Selection after successful registration

Tracking Screen

What the Screen Displays

- Parcel details
- Current parcel status: Picked Up, In Transit, Delivered
- Location tracking on a Google Map
- Delivery confirmation button

Functions

- Allow senders to track their parcels in real-time
- Show travelers the route and current parcel status
- Confirm parcel delivery

3.4 DATABASE DESIGN

The database is designed to efficiently store user, parcel, and traveler details. It ensures secure storage, retrieval, and updates of data while maintaining relationships between different entities. The database consists of multiple tables that collectively manage user registration, parcel posting, traveler authentication, and parcel tracking. The tables in the database are as follows:

- **User**
- **ParcelDetail**
- **Traveler**

The User table has userid as the primary key, ensuring that each user has a unique identifier. This allows efficient management of both parcel senders and travelers within the system. The ParcelDetail table has productid as its primary key, ensuring that each parcel entry is unique and linked to a specific sender via the senderid foreign key. Similarly, the Traveler table has id as its primary key, linking traveler details to the userid foreign key, ensuring unique traveler records.

The general goal of the database design is to store interrelated data with minimal redundancy while ensuring quick and efficient access. After defining input and output requirements, the database is structured to meet user needs by providing easy, flexible, and cost-effective data management. The proposed database design follows normalization principles, ensuring efficiency and consistency. The relationships between the tables support seamless integration of user data, parcel tracking, and traveler details within the SwiftBridge platform.

3.5 SYSTEM DEVELOPMENT

3.5.1 DESCRIPTION OF MODULES

User Management Module

The User Management Module is responsible for handling all user-related operations, including registration, authentication, and profile management. Users can sign up as either senders or travelers, providing necessary details such as name, email, mobile number, and address. Travelers must also upload identity verification documents like an Aadhar Card or Driver's License to ensure security and trustworthiness.

Once registered, users can log in using their email and password**,** with authentication managed via JWT tokens to ensure security. Role-based access control ensures that senders and travelers can only perform actions relevant to their role. Users can also update their profile details, including contact information and identification proof, as needed.

Parcel Management Module

This module allows senders to post parcels that need delivery. When posting a parcel, the sender provides:

- Parcel Image (uploaded for reference)
- Parcel Dimensions (length, width, height)
- Parcel Weight (in kilograms)
- Pickup & Delivery Locations (precise addresses)
- Payment Offer (amount the sender is willing to pay)

After posting, senders have the flexibility to edit or delete parcel listings before a traveler selects them for delivery. Travelers can browse the available parcels, viewing the details before making a selection. This module ensures that all parcel-related data is properly stored and accessible for matching.

Parcel Matching & Selection Module

Once a sender posts a parcel, the Parcel Matching & Selection Module helps travelers find suitable delivery opportunities. Travelers can search and filter parcels using:

- Pickup & Delivery Locations (to find parcels along their route)
- Parcel Size & Weight (to match their carrying capacity)

After finding a suitable parcel, the traveler can reserve it for delivery. Once a parcel is selected, the sender receives a notification**,** confirming that a traveler has accepted the delivery request. This module optimizes the parcel-matching process, ensuring efficient and secure transactions between senders and travelers**.**

Parcel Tracking & Status Module

This module allows travelers to update the status of a parcel at different stages of the delivery process. The statuses include:

- Picked Up – The traveler has collected the parcel from the sender.
- In Transit – The parcel is currently being transported.
- Delivered – The parcel has reached its final destination.

Senders can track their parcel's progress in real-time, ensuring transparency and security. Once the parcel is delivered, the sender confirms the completion of the delivery, finalizing the transaction. This module also integrates with the Geolocation API**,** allowing real-time tracking of parcels along the delivery route.

Payment & Transaction Module

The Payment & Transaction Module ensures secure and efficient handling of payments between senders and travelers. When posting a parcel, the sender specifies a payment offer**,** which the traveler receives after completing the delivery.

Payments are processed through Stripe or PayPal, integrating an escrow system where the amount is held until the sender confirms successful delivery. This prevents fraud and ensures both parties fulfill their commitments.

The module includes:

- Payment Setup **–** Senders define the delivery price.
- Secure Payment Processing – Transactions are handled through a trusted gateway.
- Payment Release – Travelers receive the payment after delivery confirmation.

This module ensures transparency and security in all financial transactions, providing a smooth payment experience for users

TESTING AND IMPLEMENTATION

4. TESTING AND IMPLEMENTATION

TESTING

Testing is a crucial phase in the software development lifecycle to ensure that the system functions as expected, meets user requirements, and operates without defects. The Smart Parcel Delivery Network platform undergoes rigorous testing to verify performance, security, and functionality before deployment.

Functional Testing

Functional testing ensures that all features of Smart Parcel Delivery Network work correctly. Each module, including User Management, Parcel Management, Tracking, and Payments**,** is tested individually and as an integrated system. Test cases cover user registration, login, parcel posting, selection, tracking, and payment processing to validate expected behavior.

Unit Testing

Unit testing is conducted at the code level to verify the correctness of individual functions and components. Each backend API endpoint in the Spring Boot application and each frontend component in React.js is tested using frameworks like JUnit for Java and Jest for JavaScript.

Integration Testing

Integration testing ensures smooth interaction between different modules and external services. The connection between the frontend, backend, and MySQL database is validated, along with third-party integrations like Google Maps API for geolocation and Stripe/PayPal for payment processing.

Performance Testing

Performance testing evaluates how well Smart Parcel Delivery Network handles various workloads. The system is tested under different user loads to identify bottlenecks and optimize performance. Load testing measures system behavior under peak usage, while stress testing assesses its stability under extreme conditions.

Security Testing

Security testing is performed to safeguard user data and transactions. Penetration testing identifies vulnerabilities like SQL injection and cross-site scripting. Authentication mechanisms using JWT (JSON Web Token) are tested to prevent unauthorized access, and encryption methods are validated to ensure data security.

User Acceptance Testing (UAT)

User acceptance testing is conducted with real users to gather feedback and validate the system's usability. Testers perform real-world scenarios like parcel booking, selection, and delivery tracking to confirm that Smart Parcel Delivery Network meets user expectations before the final deployment.

Through comprehensive testing, Smart Parcel Delivery Network ensures a seamless, secure, and high-performance experience for both senders and travelers on the platform.

IMPLEMENTATION

Implementation is the process of deploying the Smart Parcel Delivery Network platform into a live environment, ensuring it functions as intended for users. The implementation phase involves setting up the infrastructure, configuring services, and making the system available for public use.

Frontend Deployment

- Platform: Vercel
- Reason**:** Automatic deployments, free SSL, and fast content delivery

Backend Deployment

- Platform: Render
- Reason: Supports Java-based applications, provides free-tier hosting

Database Deployment

- Platform: Clever Cloud
- Reason: Offers free MySQL hosting with persistent storage

JWT Authentication & Security

- Platform**:** Managed within Spring Boot (JWT-based authentication)
- Reason: Secure authentication mechanism without external costs

Maps & Tracking Integration

- Platform: Google Maps API
- Reason: Real-time parcel tracking and location-based services

CONCLUSION

5. CONCLUSION

Smart Parcel Delivery Network successfully provides a seamless and efficient platform connecting parcel senders with travelers willing to deliver packages. By integrating React.js for the frontend, Spring Boot for the backend, and MySQL as the database, the system ensures a robust and scalable architecture. The deployment strategy enhances reliability, while Google Maps API enables real-time parcel tracking.

The implementation of JWT authentication, secure payment processing with Stripe, and role-based access control ensures that the platform is both secure and user-friendly. With features like parcel posting, selection, tracking, and payment processing, this project streamlines the entire parcel delivery workflow.

Through rigorous testing and deployment, the platform has been optimized for performance and reliability. Its ability to match senders with travelers efficiently not only enhances convenience but also promotes cost-effective and eco-friendly delivery solutions.

In conclusion, this project stands as a well-structured, scalable, and secure platform that bridges the gap between parcel senders and travelers, providing a smart, reliable, and user-centric solution for modern delivery needs.

BIBLIOGRAPHY

BIBLIOGRAPHY

Reference Books

- Clean Code: A Handbook of Agile Software Craftsmanship – Robert C. Martin
  Helps developers follow best coding practices, improving readability, maintainability, and efficiency.
- Full-Stack Development with Spring Boot and React – Juha Hinkula
  A practical guide to integrating React with Spring Boot, handling API calls, and building modern web applications.
- MySQL High Performance – Baron Schwartz
  Provides insights into MySQL database optimization, query performance tuning, and data integrity techniques.
- React Up & Running – Stoyan Stefanov
  Explains the core concepts of React JS, including component-based development, hooks, and state management.
- Spring Boot in Action – Craig Walls
  Covers in-depth concepts of Spring Boot, including REST API development, database integration, and security mechanisms like JWT authentication.

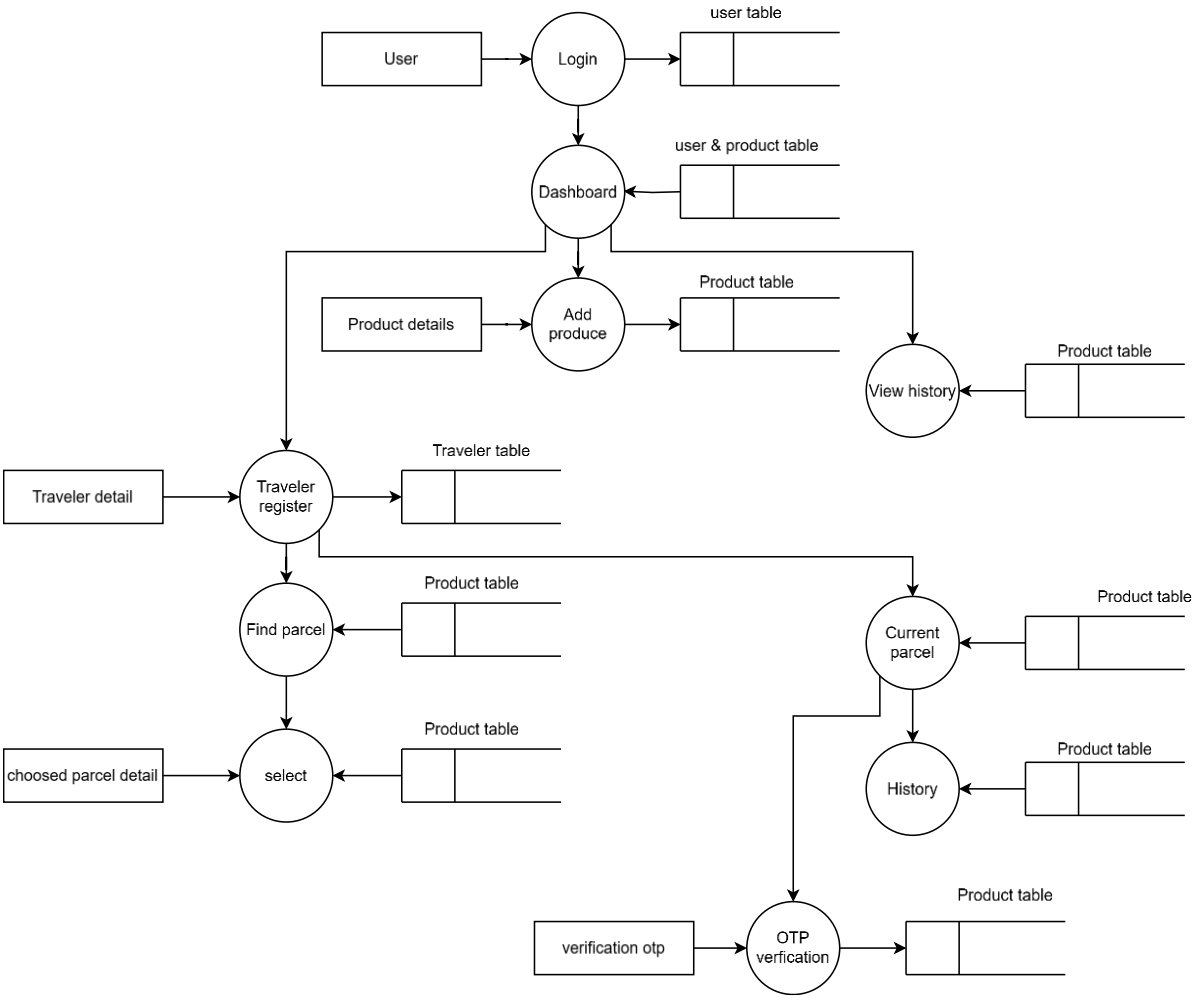
Reference Websites

- Axios GitHub Repository – https://github.com/axios/axios
- GeeksforGeeks – https://www.geeksforgeeks.org/
- MDN Web Docs – https://developer.mozilla.org/
- MySQL Documentation – https://dev.mysql.com/doc/
- React JS Official Documentation –https://react.dev/
- Spring Boot Documentation – https://spring.io/projects/spring-boot
- W3Schools – https://www.w3schools.com/

APPENDICES

APPENDICES

A. DATAFLOW DIAGRAM



B. TABLE STRUCTURE

User Table

Column Name	Data Type	Constraints
userid	BIGINT	PRIMARY KEY
name	VARCHAR(25)	NOT NULL
emailid	VARCHAR(50)	UNIQUE, NOT NULL
mobileno	BIGINT	NOT NULL
password	VARCHAR(10)	NOT NULL
address	VARCHAR(50)	NOT NULL
joindate	DATE	NOT NULL

Traveler Table

Column Name	Data Type	Constraints
id	INT	PRIMARY KEY, AUTO_INCREMENT
userid	BIGINT	FOREIGN KEY (REFERENCES User(userid))
adharNumber	BIGINT(12)	UNIQUE, NOT NULL
license	VARCHAR(15)	UNIQUE, NOT NULL

Parcel Table

Column Name	Data Type	Constraints
productid	VARCHAR(15)	PRIMARY KEY
senderid	BIGINT	FOREIGN KEY (REFERENCES User(userid))
travellerid	BIGINT	FOREIGN KEY (REFERENCES Traveler(id)), NULLABLE
image	VARCHAR(50)	NOT NULL
dimantion	VARCHAR(10)	NOT NULL
weight	DOUBLE	NOT NULL
price	INT	NOT NULL
from_address	VARCHAR(50)	NOT NULL
to_address	VARCHAR(50)	NOT NULL
status	VARCHAR(10)	DEFAULT 'posted'
otp	INT(4)	NOT NULL

C. SAMPLE CODING

```
package com.courier.demo.serviceImplemenation;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.nio.file.StandardCopyOption;
import java.sql.Blob;
import java.util.Collection;
import java.util.Collections;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Optional;
import java.util.Random;
import java.util.UUID;
import java.util.stream.Collectors;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.dao.DataIntegrityViolationException;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Service;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.multipart.MultipartFile;
import org.springframework.web.server.ResponseStatusException;
import com.courier.demo.exceptions.EmailIdAlreadyExistingException;
import com.courier.demo.exceptions.MobileNumberAlreadyExistingException;
import com.courier.demo.jparepository.Parcelrepo;
import com.courier.demo.jparepository.Travellerrepo;
import com.courier.demo.jparepository.Userrepo;
import com.courier.demo.model.ParcelDetail;
//import com.courier.demo.model.Traveller;
```

```
import com.courier.demo.model.TravellerDetail;
import com.courier.demo.model.User;
import com.courier.demo.service.Servece;
import jakarta.transaction.Transactional;

@Service
public class ServiceImplementation implements Servece{
	@Autowired
	Userrepo repo;



	@Autowired
	Parcelrepo parcelrepo;

	@Autowired
	Travellerrepo traveller;

	private static final Random random = new Random();

   public Long generateUniqueUserId() {
      Long userId;
      do {
         userId = 100000 + random.nextLong(900000); // Generate 6-digit ID
      } while (repo.existsByUserid(userId)); // Check for uniqueness
      return userId;
   }
	//uswer signup
	@Override
	public String Signup(User user) {
		Long unique=generateUniqueUserId();
		user.setUserid(unique);
		if(repo.existsByEmailid(user.getEmailid())) {
			throw new EmailIdAlreadyExistingException("This email id is already
in use");
```

```
                }
                if(repo.existsByMobileno(user.getMobileno())) {
                        throw new MobileNumberAlreadyExistingException("This mobile number is already exiating");
                }
                try {
                        repo.save(user);
                        return "user register successfully";
                }catch(DataIntegrityViolationException ex){
                        throw new RuntimeException("an unexpected errror occur");
                }
        }

        @Override
                public ResponseEntity<Map<String, Object>> Login(String emailid,String password) {
                Optional<User> useroptional=repo.findByEmailid(emailid);
                if(useroptional.isPresent()) {
                        User user = useroptional.get();
              Map<String, Object> response = new HashMap<>();
              response.put("message", "Successfully logged in");
              response.put("user", user); // Return user details
              return ResponseEntity.ok(response);
              }
                 return null;
                 }

        @Override
    public String addProduct(ParcelDetail parcel) {
       try{
           User existingUser = repo.findById(parcel.getUser().getUserid())
                 .orElseThrow(() -> new RuntimeException("User not found"));
           parcel.setUser(existingUser);
          // Save to database
```

```
            parcelrepo.save(parcel);
            return "Parcel successfully added with ID: " + parcel.getProductid();
        } catch (Exception e) {
            e.printStackTrace();
            return "Error saving parcel: " + e.getMessage();
        }
    }
        private static final String UPLOAD_DIR =
        "C:\\Users\\mshar\\Desktop\\Project\\images";
        @Override
        public String addProduct(MultipartFile file, String dimantion, Double weight, Integer
price, String fromAddress,
                                 String toAddress, Long userId) {
                    try {
                     User existingUser = repo.findById(userId)
                .orElseThrow(() -> new RuntimeException("User not found"));
                     String fileName = "P-" + UUID.randomUUID().toString().substring(0, 8) + "-
" + file.getOriginalFilename();
        Path filePath = Paths.get(UPLOAD_DIR, fileName);
        Files.copy(file.getInputStream(),
        filePath, StandardCopyOption.REPLACE_EXISTING);

        ParcelDetail parcel = new ParcelDetail();
        parcel.setUser(existingUser);
        parcel.setImage(fileName);  // Store only the file name/path
        parcel.setDimantion(dimantion);
        parcel.setWeight(weight);
        parcel.setPrice(price);
        parcel.setFrom_address(fromAddress);
        parcel.setTo_address(toAddress);

        parcelrepo.save(parcel);
        return "Parcel successfully added with ID: " + parcel.getProductid();
                    }catch(IOException e) {
```

```
                return("error saeing mssagee"+e.getMessage());
            }catch(Exception a) {
                return("Error saving paercel"+a.getMessage());
            }

        }

        public ResponseEntity<List<ParcelDetail>> getParcelsBySenderId(Long senderid) {
            List<ParcelDetail> parcels = parcelrepo.findByUser_Userid(senderid);
            if (parcels.isEmpty()) {
                return new ResponseEntity<>(HttpStatus.NOT_FOUND);
            } else {
                return new ResponseEntity<>(parcels, HttpStatus.OK);
            }
          }

        @Override
        public ResponseEntity<String>  registerTraveller(TravellerDetail travelledet){
                traveller.save(travelledet);
      return new ResponseEntity<>("successfully updated", HttpStatus.OK);
        }

        @Override
        public boolean istravell(Long userid) {
                System.out.println(traveller.existsByUser_userid(userid));
                return traveller.existsByUser_userid(userid);
        }

        @Override
        public List<ParcelDetail> travellerparcel(Long userid){
                if(traveller.existsByUser_userid(userid)) {
                    return
parcelrepo.findByTravelleridIsNullAndUser_UseridNot(userid);
                }
```

```
            return Collections.emptyList();
        }

        @Override
        public String selectedParcel(Long userid,String productid){
            if(traveller.existsByUser_userid(userid)) {
                System.out.println(traveller.existsByUser_userid(userid));
                ParcelDetail parceldetail=parcelrepo.findById(productid).get();
                parceldetail.setTravellerid(userid);
                parceldetail.setStatus("selected");
                parcelrepo.save(parceldetail);
                return "Parcel selected successfully";
            }
            return "something worong please try again";
        }

        @Override
        public User travellerdet(Long userid) {
            User user=repo.findByUserid(userid);
            return user;
        }

       @Override
       public Long findTravellerid (String productid) {
            Long travellerid=parcelrepo.findTravellerIdByProductid(productid);
            return travellerid;
       }

}


import { BrowserRouter as Router, Routes, Route } from "react-router-dom";
import Signup from "./Component/Signup";
import Login from "./Component/Login";
```

```
import Dashboard from "./Component/Dashboard";
import UploadProduct from "./Component/UploadProduct";
import TravelerRegistration from "./Component/TravelerRegistration";
import ParcelSelection from "./Component/ParcelSelection";

function App() {
  return (
    <Router>
      <Routes>
        <Route path="/signup" element={<Signup />} />
        <Route path="/login" element={<Login />} />
        <Route path="/dashboard" element={<Dashboard />} />
        <Route path="/upload-product" element={<UploadProduct />} />
        <Route path="/register-traveler" element={<TravelerRegistration />} />
        <Route path="/select-parcel" element={<ParcelSelection />} />
      </Routes>
    </Router>
  );
}

export default App;

import { useState, useEffect } from "react";
import axios from "axios";
import { useParams, useNavigate } from "react-router-dom";

const Dashboard = () => {
  const navigate = useNavigate();
  const { senderid } = useParams();
  const [user, setUser] = useState(null);
  const [parcels, setParcels] = useState([]);
  const [isTraveler, setIsTraveler] = useState(false);
  const [travelerDetails, setTravelerDetails] = useState(null);
```

```
  useEffect(() => {
    const storedUser = JSON.parse(localStorage.getItem("user"));
    if (storedUser) {
      setUser(storedUser);

      // Check if the user is a traveler
      axios
        .get(`http://localhost:8080/SwiftBridge/istraveler?userid=${storedUser.userid}`)
        .then((response) => setIsTraveler(response.data))
        .catch((error) =>
          console.error("Error checking traveler status:", error)
        );

      // Fetch parcels
      axios
        .get(`http://localhost:8080/SwiftBridge/view/${storedUser.userid}`)
        .then((response) => {
          setParcels(response.data);

          // Check if any parcel has status "selected"
          response.data.forEach((parcel) => {
            if (parcel.status === "selected") {
              // Fetch traveler ID using productid
              axios

.get(`http://localhost:8080/SwiftBridge/findTraveller?productid=${parcel.productid}`)
                .then((travelerResponse) => {
                  const travelerId = travelerResponse.data;
                  // Fetch traveler details
                  axios
                    .get(`http://localhost:8080/SwiftBridge/travellerDetail?userid=${travelerId}`)
                    .then((travelerDetailResponse) => {
                      setTravelerDetails(travelerDetailResponse.data); // Set traveler details
                    })
```

```
            .catch((error) => console.error("Error fetching traveler details:", error));
          })
          .catch((error) => console.error("Error fetching traveler ID:", error));
        }
      });
    })
    .catch((error) => console.error("Error fetching parcels:", error));
  }
}, [senderid]);

return (
  <div className="min-h-screen bg-gray-100 p-6">
    <div className="max-w-4xl mx-auto bg-white p-6 rounded-lg shadow-md">

      <h2 className="text-2xl font-bold text-gray-700">User Dashboard</h2>
      {user && (
        <div className="mt-4 p-4 bg-indigo-100 rounded-lg">
          <p>
            <strong>Name:</strong> {user.name}
          </p>
          <p>
            <strong>Email:</strong> {user.emailid}
          </p>
          <p>
            <strong>Mobile:</strong> {user.mobileno}
          </p>
          <p>
            <strong>Address:</strong> {user.address}
          </p>
          <button
            onClick={() => navigate("/upload-product")}
            className="mt-4 px-4 py-2 bg-indigo-600 text-white rounded-lg hover:bg-indigo-
700"
          >
```

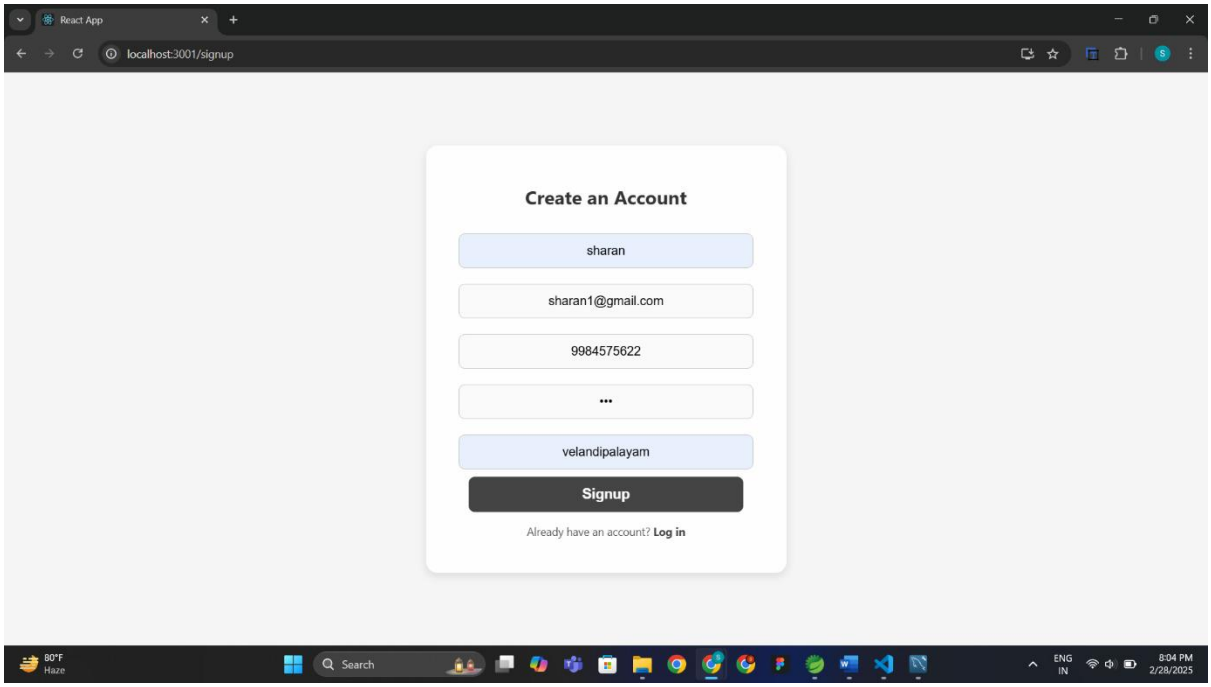
```
          Upload Product
        </button>
        {/* Traveler Button */}
        <button
          onClick={() => {
            if (isTraveler) {
              navigate("/select-parcel"); // Navigate to parcel selection
            } else {
              navigate("/register-traveler"); // Navigate to traveler registration
            }
          }}
          className="mt-4 ml-4 px-4 py-2 bg-green-600 text-white rounded-lg hover:bg-green-700"
        >
          {isTraveler ? "view parcel" : "register"}
        </button>
      </div>
    )}

    <h3 className="text-xl font-bold text-gray-600 mt-6">Your Parcels</h3>
    <div className="mt-4 space-y-4">
      {parcels.length > 0 ? (
        parcels.map((parcel, index) => (
          <div key={index} className="bg-white p-4 rounded-lg shadow-md">
            <img
              src={process.env.PUBLIC_URL + parcel.image}
              alt="Parcel"
              onError={(e) =>
                (e.target.src =
                  process.env.PUBLIC_URL +
                  "assert/1739541433205_courier.drawio.png.png")
              }
              className="w-full h-40 object-cover rounded-md"
            />
```

```
<p>
  <strong>Dimensions:</strong> {parcel.dimantion}
</p>
<p>
  <strong>Weight:</strong> {parcel.weight} kg
</p>
<p>
  <strong>Price:</strong> ${parcel.price}
</p>
<p>
  <strong>From:</strong> {parcel.from_address}
</p>
<p>
  <strong>To:</strong> {parcel.to_address}
</p>
<p
  className={`text-white px-2 py-1 inline-block rounded-md ${
    parcel.status === "posted" ? "bg-blue-500" : "bg-green-500"
  }`}
>
  {parcel.status}
</p>

{/* Display traveler details if parcel is selected */}
{parcel.status === "selected" && travelerDetails && (
  <div className="mt-4 bg-green-100 p-4 rounded-md">
    <h4 className="text-xl font-bold">Traveler Details</h4>
    <p>
      <strong>Name:</strong> {travelerDetails.name}
    </p>
    <p>
      <strong>Mobile:</strong> {travelerDetails.mobileno}
    </p>
  </div>
```

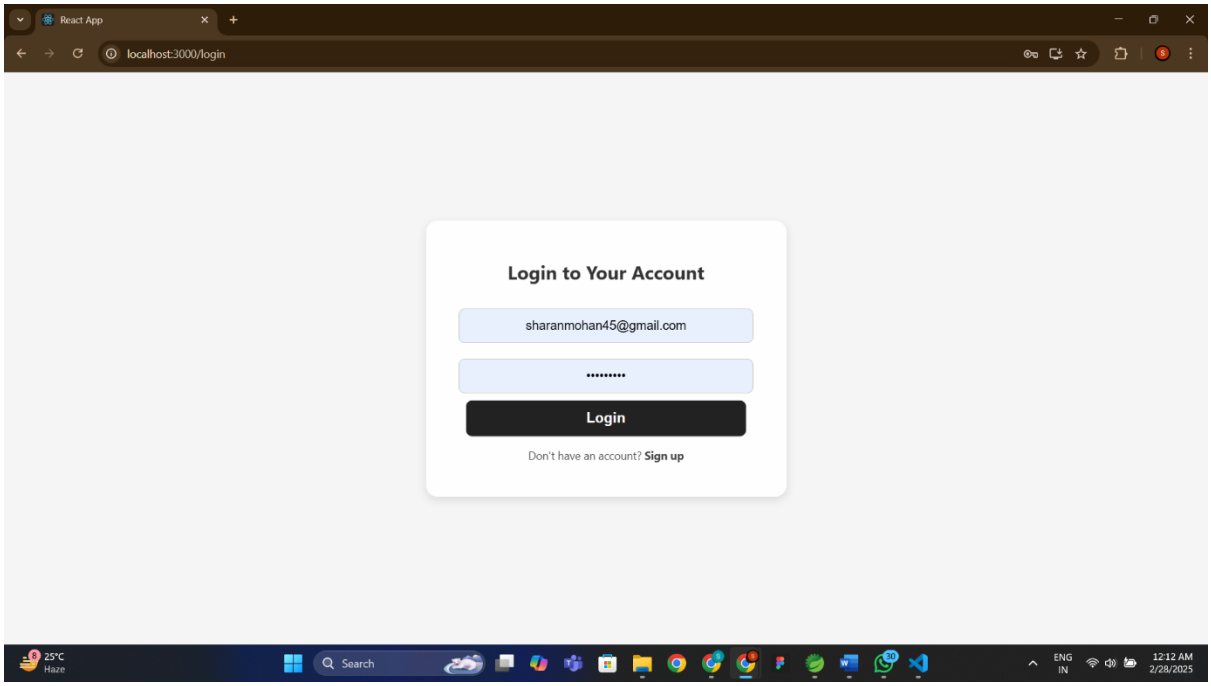
```
            )}
          </div>
        ))
      ) : (
        <p>No parcels found.</p>
      )}
    </div>
  </div>
 );
};

export default Dashboard;
```

D. SAMPLE INPUT

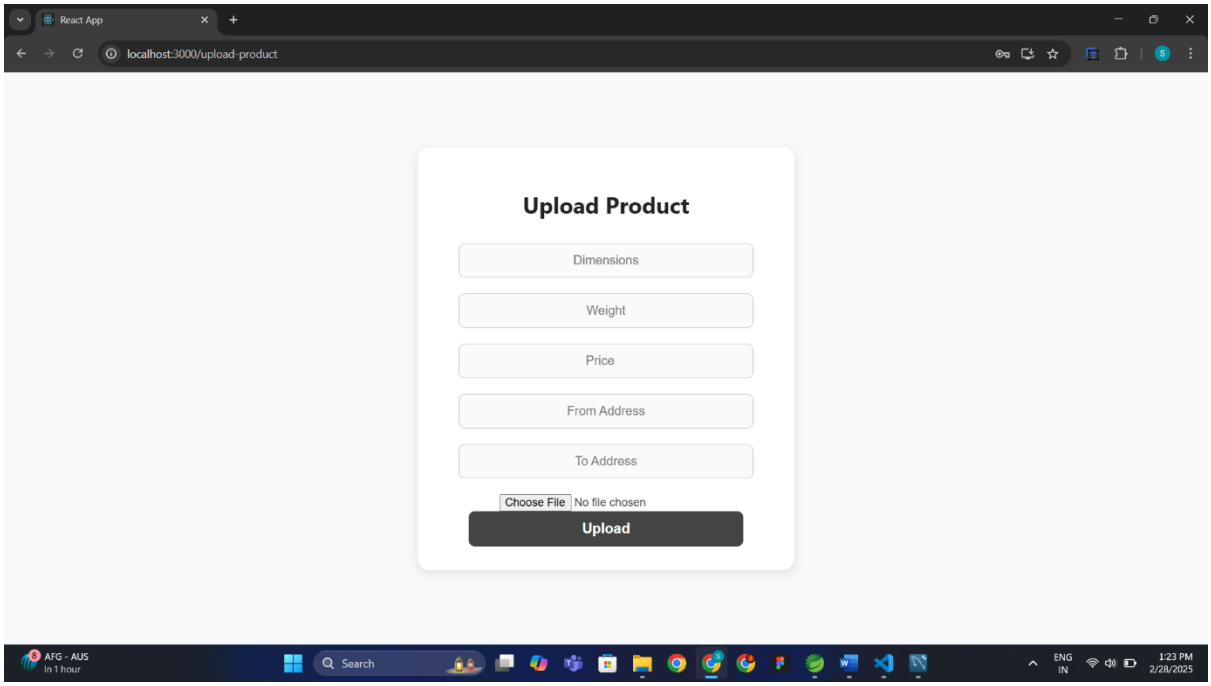
Signup Page



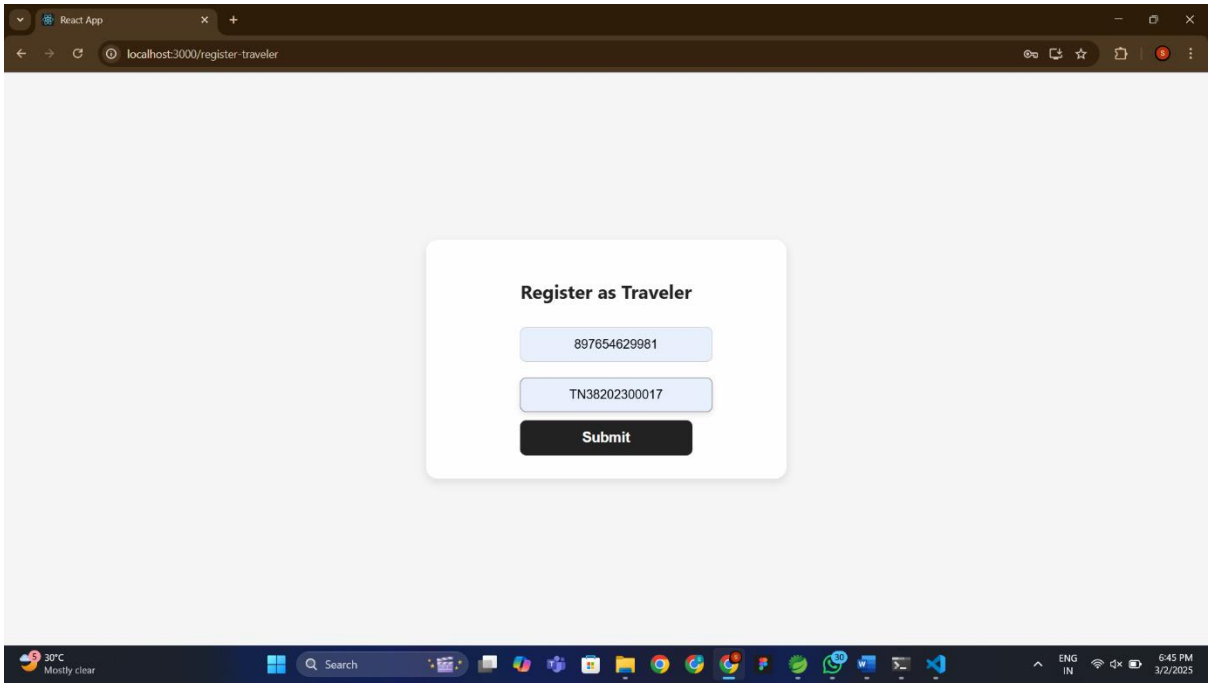
Login Page



Upload parcel

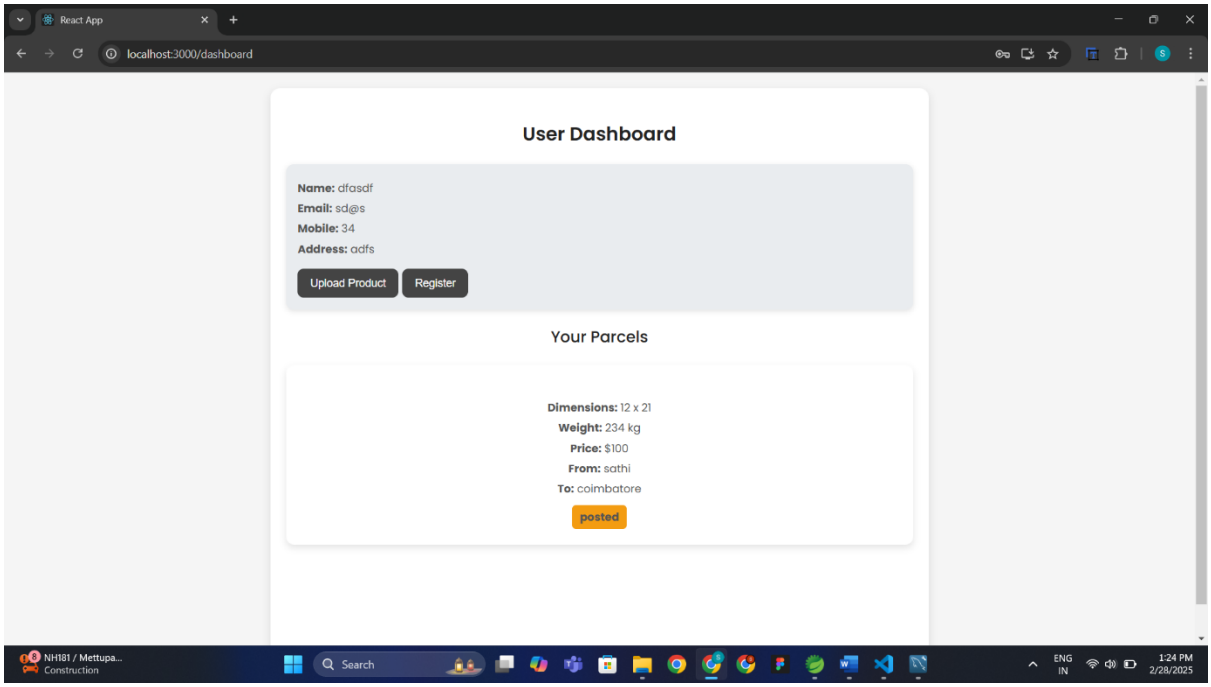


Traveler registration



E. SAMPLE OUTPUT

Dashboard



Parcel selection