

TRANSPORTATION MANAGEMENT SYSTEM

A PROJECT REPORT

Submitted by

SAKSHI RAKESHBHAI PANCHAL

211260116038

In partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

In

Information Technology

SAL Engineering and Technical Institute, Ahmedabad



Gujarat Technological University, Ahmedabad

April, 2025



SAL Engineering and Technical Institute

Opp. Science City, Sola-Bhadaj Road, Ahmedabad, Gujarat 380060

CERTIFICATE

This is to certify that the project report submitted along with the project entitled **Transportation Management System** has been carried out by **Sakshi Rakeshbhai Panchal** under my guidance in partial fulfillment for the degree of Bachelor of Engineering in Information Technology, 8th Semester of Gujarat Technological University, Ahmedabad during the academic year 2024-25.

Prof. Harshit Vora

Internal Guide

Prof. Sachi Bhavsar

Head of Department



TO WHOM IT MAY CONCERN

Date: - 19/04/2025

This is to certify that Ms. Sakshi Rakeshbhai Panchal has worked as an Intern Backend Developer with Tech It Easy from 20-01-2025 to 19-04-2025. During this period, Ms. Sakshi has worked on Saral Transport: A Logistics Management System and successfully completed her job duties and responsibilities to our satisfaction.

We would like to express our sincere appreciation for Ms. Sakshi's hard work and dedication throughout her internship with our organization. Her work has exceptional technical expertise, creative problem-solving abilities, and a commitment to delivering excellence. Ms. Sakshi's skills and experience gained during her internship will undoubtedly contribute to her future professional endeavors.

She exhibited a positive attitude, a strong willingness to learn and take on new challenges, and effective communication skills. These qualities greatly contributed to her success as a valuable member of our team. It has been a pleasure working with her and witnessing her growth and development during her internship with our organization. We wish Ms. Sakshi all the best in her future endeavours, and we are confident that she will achieve great success.

With regards,

For, TECH IT EASY-

For, TECH IT EASY

Authorized Signatory

TECH IT EASY

930, SAHJANAND ESTATE, RAIPUR, AHMEDABAD | 9265435410 | techiteasy11@gmail.com

GTU CERTIFICATE



SAL Engineering and Technical Institute

Opp. Science City, Sola-Bhadaj Road, Ahmedabad, Gujarat 380060

DECLARATION

We hereby declare that the Internship / Project report submitted along with the Internship /Project entitled **Transportation Management System** submitted in partial fulfillment for the degree of Bachelor of Engineering in Information Technology to Gujarat Technological University, Ahmedabad, is a bonafide record of original project work carried out by me / us at **SAL Engineering and Technical Institute** under the supervision of **Mr. Dharmik Trivedi** and that no part of this report has been directly copied from any students' reports or taken from any other source, without providing due reference.

Name of Student

Sakshi Panchal

Sign of Student

ACKNOWLEDGEMENT

I wish to express our sincere gratitude to our External guide **Mr. Dharmik Trivedi** for continuously guiding me at the company and answering all my doubts with patience. I would also like to thank my Internal Guide **Prof. Harshit Vora** for helping us through our internship by giving us the necessary suggestions and advice along with their valuable co-ordination in completing this internship.

We also thank our parents, friends and all the members of the family for their precious support and encouragement which they had provided in completion of our work. In addition to that, we would also like to mention the company personals who gave us the permission to use and experience the valuable resources required for the internship.

Thus, in conclusion to the above said, I once again thank the staff members of **Tech It Easy** for their valuable support in completion of the project.

Thank You.

ABSTRACT

*The **Transportation Management System (TMS)** is a comprehensive software solution designed to streamline and enhance logistics operations. This system is built using **Python**, **MySQL**, and **FastAPI**, providing a robust backend infrastructure for efficient data handling and processing. The TMS aims to improve the management of transportation activities by automating key processes such as order tracking, dispatch scheduling, and proof of delivery (POD) management.*

By implementing this TMS, logistics companies can achieve improved visibility, streamlined operations, and enhanced customer satisfaction through accurate and timely data management. The system's scalability and modular design allow for future enhancements and integration with advanced technologies to further optimize transportation workflows.

LIST OF FIGURES

Fig 2.3.1 Sequence of Operation Followed.....	6
Fig 3.1 Spiral Model.....	24
Fig 5.1 Block Diagram.....	33
Fig 5.2 Flowchart Diagram.....	34
Fig 5.3 ER Diagram.....	35
Fig 5.5 Use Case Diagram.....	36
Fig 5.6 Sequential Diagram.....	37

LIST OF TABLES

Table 6.1.1	Bilty_chalan table.....	38
Table 6.1.2	Item_bilty table.....	38
Table 6.1.3	Menu table.....	39
Table 6.1.4	Separate_bilty_chalan table.....	39

LIST OF ABBREVIATIONS

TMS	Transportation Management System
SRS	Software Requirement Specification
API	Application Programming Interface
IoT	Internet of Things
DBMS	Database Management System
AI	Artificial Intelligence
POD	Proof Of Delivery
CRUD	Create,Read,Update,Delete
SQL	Structured Query Language
HTTP	Hypertext Transfer Protocol
REST	Representational State Transfer
FTP	File Transfer Protocol
ETL	Extract,Transform,Load
BiltyMaster	Data table containing shipment and POD upload details
Media	Previous data table used for created date references
ERP	Enterprise Resource Planning
KPI	Key Performance Indicator
CSV	Comma-Separated Values
XML	Extensible Markup Language
SDK	Software Development Kit
JSON	JavaScript Object Notation

LIST OF CONTENTS

Acknowledgement.....	I
Abstract.....	II
List of Figures.....	III
List of Tables.....	IV
List of Abbreviations.....	V
Table of Contents.....	VI
Chapter 1 Company Review.....	1
1.1 About the Company.....	1
1.1.1 Who We Are.....	1
1.1.2 What We Do.....	1
1.1.3 Why Choose Us.....	2
1.2 Aim and Objective of the Internship.....	3
Chapter 2 Overview of the Department.....	4
2.1 Work in each department.....	4
2.2 Technical Specification.....	5
2.3 Schematic Layout.....	6
2.4 Detailed explanation of each stage of production.....	7
Chapter 3 Introduction of Project.....	8
3.1 Project Summary.....	8
3.2 Purpose.....	8
3.3 Objectives.....	10
3.4 Scope.....	11
3.5 Technology and Literature Review.....	13
3.6 Project Planning.....	15
3.6.1 Project Justification.....	17
3.6.2 Cost estimation of the project.....	18
3.6.3 Roles and Responsibilities.....	18
3.6.4 Group Dependencies.....	19
3.7 Project Scheduling.....	20

Chapter 4 System Analysis.....	24
4.1 Study of current system.....	24
4.2 Problems and weakness of current system.....	25
4.3 Requirements of new system.....	25
4.4 System Feasibility.....	27
4.4.1 Does the system contribute to the overall objective of the organizations?.....	28
4.4.2 Can the system be implemented using the current technology and within the given cost and schedule constraints.....	28
4.4.3 Can the system be integrated with other systems which are already in place?.....	28
4.5 Process in new system.....	29
4.6 Features of new system.....	29
4.7 List main modules.....	29
Chapter 5 System Design.....	32
5.1 Block Diagram.....	32
5.2 Flow Chart Diagram.....	33
5.3 ER Diagram.....	34
5.4 Use Case Diagram.....	35
5.5 Sequential Diagram.....	36
Chapter 6 Implementation.....	37
6.1 Implementation Platform.....	37
6.2 Module Specification.....	38
6.3 Outcomes.....	40
6.4 Screenshots.....	42
Chapter 7 Testing.....	53
7.1 Testing Techniques.....	53
7.2 Test Automation.....	53
7.3 Test Environment.....	53
7.4 Defect Management.....	54
7.5 Performance Testing.....	54
7.6 Security Testing.....	54
7.7 User Acceptance Testing(UAT).....	54

Chapter 8 Conclusion and Discussion.....	55
8.1 Overall analysis of internship.....	55
8.2 Photograph and date of surprise visit by institute mentor.....	56
8.3 Problems encountered and possible solutions.....	57
8.4 Summary of Internship.....	59
8.5 Limitations and Future Enhancement.....	60
Conclusion.....	62
References.....	63

CHAPTER 1: COMPANY OVERVIEW

1.1 ABOUT THE COMPANY

1.1.1 Who We Are:

Tech It Easy, established in 2020, is a leading software development company specializing in transportation management solutions and custom software development. In less than five years, we have proudly associated with 100+ satisfied clients, delivering smart, automated, and efficient technology solutions that help businesses streamline operations, enhance compliance, and improve productivity.

1.1.2 What We Do:

At Tech It Easy, we believe in building strong partnerships and helping businesses grow with the right technology. Whether you're looking for a robust transportation management system, a cutting-edge software solution, or digital transformation services, we are here to turn your vision into reality.

Our flagship products, Saral Transport and Saral Way, have transformed the logistics industry. Beyond transportation software, we offer custom software development, IoT solutions, mobile & web app development, DevOps, and application modernization for businesses across various industries.

We make technology simple, scalable, and efficient, enabling businesses to automate processes, reduce costs, and stay ahead in a digital world.

1.1.3 Why Choose Us:

We deliver innovative, scalable, and secure technology solutions tailored to drive business growth and efficiency.

- Expertise Across Industries – Proven solutions in logistics, enterprise, and digital transformation.
- Scalable & Secure Solutions – Future-proof applications designed for growth and efficiency.
- Customer-Centric Approach – We prioritize client satisfaction with personalized services.
- Cutting-Edge Technologies – We use the latest advancements in AI, cloud computing, and automation.
- End-to-End Services – From strategy to execution, we provide comprehensive technology solutions.

1.2 AIM AND OBJECTIVE OF THE INTERNSHIP

- An internship is a purposeful activity of the student set in a work environment in order to obtain learning outcomes within their curriculum. During the internship, the knowledge, skills, and attitudes learned in the program can be applied.
- The aim of the internship provides a direction to the activities, helps to focus on a result, and to assess the result achieved.
- Before going on the internship, two important factors guiding your development should be taken into account when formulating the aim:
 1. Connecting what you have learned (theoretical and practical knowledge on your subject field) with actual work experience, in order to complement your field specific skills and learn new ones.
 2. Apply and analyze at least one future skill.
- There can be one or two aims, but both development of field specific skills as well as future skills have to be represented.

CHAPTER 2

OVERVIEW OF THE DEPARTMENTS OF ORGANIZATION AND LAYOUT OF PRODUCTION/PROCESS BEING CARRIED OUT IN COMPANY

2.1 DEPARTMENTS OVERVIEW:

Operations Department:

- Manages daily transportation schedules and dispatching.
- Tracks vehicle movements, driver assignments, and delivery timelines.
- Ensures compliance with regulatory standards.

IT Department:

- Develops, maintains, and upgrades the TMS software.
- Handles database management, server maintenance, and security protocols.
- Ensures system integration with GPS, ERP systems, and third-party APIs.

Customer Support & Service:

- Handles client queries regarding shipments, delays, and tracking.
- Provides real-time updates and troubleshooting.

Finance and Accounts:

- Manages billing, invoicing, and financial reporting.
- Ensures proper recording of freight charges, driver payments, and fuel expenses.

Fleet Management:

- Monitors vehicle maintenance schedules.
- Tracks fuel consumption, vehicle utilization, and repairs.

Sales & Marketing:

- Focuses on client acquisition, partnerships, and promotions.
- Ensures customer satisfaction and service quality.

2.2 TECHNICAL SPECIFICATION OF TECHNOLOGIES USED IN DEPARTMENT

1. Booking & Order Management :

- Frontend: React.js for UI , HTML5, CSS3 for responsive design.
- Backend: FastAPI integrated with python for efficient data processing.
- Database: MySQL for structured data storage.

2. Route Planning & Optimization:

- Frontend Technologies:
 - React.js
- Backend Technologies:
 - Python
- Other:
 - MySQL
 - SQL Server

3. Consulting Services:

- SQL

4. Training and Development:

- Python

5. Sales and Marketing:

- Python
- SQL

6. Customer Support:

- Python
- SQL

2.3 SCHEMATIC LAYOUT

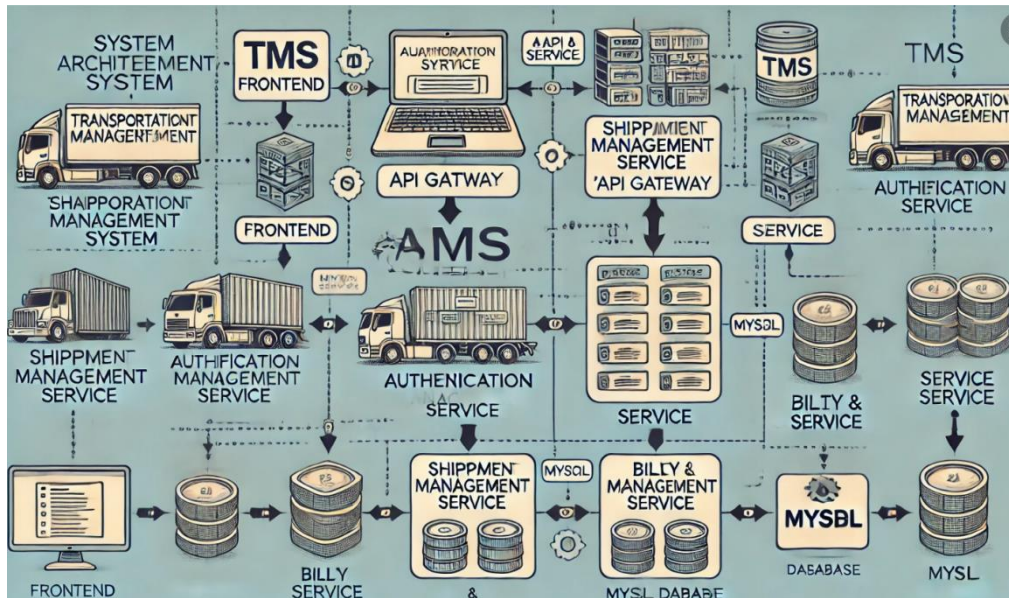


Fig 2.3.1 Sequence of Operation followed

This system architecture diagram illustrates the overall structure and flow of your Transportation Management System (TMS). At the core of the system is the TMS itself, which connects to various services and databases to manage transportation, shipment, and billing processes. The frontend interfaces serve as the primary interaction points for users, providing access to TMS functionalities. Notably, there are separate frontends for different modules, such as shipment management and billing services, ensuring a clear separation of concerns.

The API Gateway plays a crucial role in facilitating secure communication between services. Acting as a bridge, it routes requests efficiently while ensuring proper authentication. The Authentication Service further reinforces system security by verifying user identities before granting access to critical features. This integration of security layers ensures data integrity and protection.

The Shipment Management Service is responsible for handling shipment-related data, including tracking, status updates, and document uploads. This service directly interacts with the MySQL database to store and retrieve shipment records. Similarly, the Billing and Management Service manages financial data, such as invoices and payments, with its own MySQL database ensuring reliable storage.

2.4 DETAILED EXPLANATION OF EACH STAGE OF PRODUCTION

1. **Planning:** The first step is to define the project's objectives, scope, and resources. This phase involves identifying system requirements such as shipment tracking, POD management, and billing features. Feasibility analysis, budgeting, and team allocation are also part of this phase.
2. **Analysis:** In this stage, detailed requirements are gathered by consulting stakeholders like logistics managers and drivers. A comprehensive Software Requirement Specification (SRS) is created, and potential risks or dependencies are identified. For instance, in your TMS, this phase would involve determining how `pod_upload_date` integrates with the billing system.
3. **Design:** This phase focuses on creating technical blueprints for the system. System architecture, database schema design, and API endpoint planning take place here. For example, in TMS, you would design tables like `BiltyMaster`, `TripDetails`, and `Invoices`, ensuring proper relationships between them.
4. **Development:** During development, the actual coding takes place. Backend development using FastAPI, database setup with MySQL, and frontend design for dashboards and portals are key tasks. For instance, you might build an endpoint like `/upload_pod` to handle file uploads and update the `pod_upload_date` in the database.
5. **Testing:** This phase ensures that the system functions as intended. Unit testing, integration testing, and user acceptance testing (UAT) are performed. Tools like Postman for API testing and pytest for automated tests are commonly used. For example, you'd test if `pod_upload_date` updates correctly after successful uploads.
6. **Deployment:** Once testing is successful, the system is deployed on a cloud platform such as AWS or Azure. CI/CD pipelines are implemented for smoother updates, and server configurations are optimized to ensure secure database connections.

CHAPTER 3: INTRODUCTION OF PROJECT

This chapter starts with a high-level overview of the project. It describes the specific aims and objectives of the project.

3.1 PROJECT SUMMARY

The Transportation Management System (TMS) is a backend-powered solution designed to streamline logistics operations by efficiently managing the movement of goods. Developed using Python (FastAPI) for backend APIs and MySQL for data storage, the system ensures seamless tracking, updating, and reporting of transport activities.

Key modules include BiltyMaster, Media, and POD Upload, enabling real-time tracking of consignments, proof of delivery (POD) uploads, and historical data analysis. The system supports advanced features like filtering by date, company-wise reports, and data validation using `null_valid_upto` logic, making it both scalable and adaptable.

Integrated with a React.js frontend and a Telegram bot, users can interact with the system via a user-friendly interface and get updates or reports based on various filters like company, timeframe (yesterday, last week, or financial year), and more.

Overall, this TMS improves operational efficiency, provides better visibility into logistics data, and reduces manual intervention through automation and smart integration.

3.2 PURPOSE

1. **Performance Evaluation:** The TMS helps track key performance metrics such as delivery timelines, route efficiency, and fuel consumption. For instance, by monitoring `pod_upload_date`, the system can identify delays in proof-of-delivery submissions, allowing managers to take corrective actions.
2. **Quality Assurance:** Ensuring data accuracy and system reliability is crucial. The TMS can implement automated testing to verify that `pod_upload_date` updates correctly only when the uploaded file meets specific criteria. This

reduces manual errors and ensures data consistency.

3. **Customer Experience Enhancement:** By , providing features like real-time shipment tracking, automated delivery status notifications, and easy access to PODs, the TMS enhances customer satisfaction. For example, sending an automatic email when a shipment reaches its destination keeps clients informed.
4. **Data-Driven Decision Making:** The TMS can generate detailed reports on delivery patterns, driver performance, and shipment delays. For instance, analyzing data from BiltyMaster can reveal which routes consistently experience delays, enabling better route planning.
5. **Competitive Advantage:** Implementing features like route optimization, dynamic pricing based on traffic conditions, or automated dispatching can give the business a strategic edge. For example, a TMS that suggests faster delivery routes based on real-time traffic data can improve delivery speed and reduce costs.
6. **Compliance and Risk Management:** The TMS can track important compliance-related data such as vehicle insurance expiry, driver certifications, and pending POD submissions. Automated reminders for approaching deadlines help business avoid penalties and ensure operational safety.

3.3 OBJECTIVES

- The primary objective of a Transportation Management System (TMS) is to optimize logistics operations by improving the efficiency of shipment planning, vehicle allocation, and route management. By leveraging automated processes and data-driven insights, a TMS ensures that deliveries are faster, more reliable, and cost-effective. For instance, route optimization algorithms can help identify the most efficient paths, reducing fuel consumption and delivery times.
- Another key objective is enhancing shipment visibility. A TMS provides real-time tracking capabilities that keep both customers and logistics managers informed about shipment statuses. This improved visibility helps identify delays early, allowing proactive decision-making. For example, automated alerts can notify stakeholders about shipment milestones, improving overall transparency.
- Ensuring delivery accuracy is also a crucial goal. The TMS reduces manual errors by automating data entry processes such as POD uploads and billing updates. By maintaining accurate records, businesses can prevent costly mistakes and improve accountability. For instance, ensuring that `pod_upload_date` updates correctly upon successful file uploads helps maintain precise delivery records.
- Cost reduction is another core objective of a TMS. By analyzing delivery data and optimizing routes, businesses can minimize fuel expenses, reduce idle time, and maximize vehicle utilization. Additionally, a TMS helps in identifying under performing routes or unnecessary trips, enabling managers to implement cost-saving measures.

3.4 SCOPE

1. **Functionality:** The TMS must offer essential features like shipment booking, dispatch management, POD upload tracking, and automated invoicing. Advanced features such as route optimization, driver performance tracking, and GPS integration should be included to improve logistics efficiency. For instance, implementing an automated `pod_upload_date` update ensures accurate delivery timelines.
2. **User Roles and Permissions:** The system should provide role-based access control (RBAC) to restrict functionalities based on user roles. For example, admins can manage system configurations, dispatchers can assign shipments, and drivers can access POD upload functionality. This ensures secure and efficient workflow management.
3. **Mystery Shopping Workflow:** To assess service quality, the TMS can include a mystery shopping module. This allows businesses to assign evaluators who discreetly assess delivery processes, driver behavior, and shipment conditions. The system can log observations and generate quality control reports.
4. **Feedback Management:** The TMS should incorporate a feedback portal where customers can rate their delivery experience and report issues. Feedback data can be analyzed to identify service gaps, helping the business enhance its logistics processes.
5. **Analytics and Reporting:** Comprehensive reporting tools should be integrated to track KPIs such as delivery performance, shipment delays, driver efficiency, and fuel consumption. Dashboards with real-time insights empower managers to make data-driven decisions. For instance, tracking delayed `pod_upload_date` entries can help pinpoint recurring issues.

6. **Integration:** The TMS should seamlessly connect with third-party systems like ERP, accounting software, and GPS tracking solutions. For example, integrating with billing platforms can automate invoice generation after a successful POD submission, ensuring faster financial processes.
7. **Compliance and Security:** To maintain data integrity and meet regulatory standards, the TMS must include encryption protocols, data access controls, and audit logs. Automated reminders for vehicle insurance expiry, driver license renewals, and POD deadlines can mitigate compliance risks.
8. **Training and Support:** The system should include a knowledge base, user guides, and live support to assist staff during on boarding and day-to-day use. Regular training modules can help employees adapt to new system updates or feature enhancements.
9. **Scalability and Flexibility:** The TMS should be designed to handle business growth. Using modular development with FastAPI ensures new features can be integrated without disrupting core functionalities. Whether adding new shipment zones or integrating advanced reporting tools, scalability is crucial.
10. **Continuous Improvement:** To stay competitive, the TMS must support continuous enhancement through user feedback, performance monitoring, and technology updates. Regular code optimizations, database improvements, and feature upgrades should be part of the development roadmap. For example, refining the pod_upload_date logic to improve accuracy or adding AI-based route optimization can significantly boost efficiency over time.

3.5 TECHNOLOGY AND LITERATURE REVIEW

FRONT END:

REACT.JS

Using React.JS for your TMS frontend, the focus should be on building dynamic, efficient, and user-friendly interfaces. Leveraging React's component-based architecture will help you efficiently manage complex UI elements like shipment tracking dashboards, POD upload forms, and data visualizations. If you'd like guidance on structuring your React components, implementing state management, or integrating your FastAPI backend.

BACK END:

A **Transportation Management System (TMS)** is a crucial software solution that optimizes the planning, execution, and tracking of freight movements. The backend of a TMS is the backbone that ensures data handling, business logic, and system functionality operate seamlessly. This section outlines the key technologies and literature references used in developing a robust TMS backend.

1. **Technology stack for TMS Backend:** The TMS backend is typically built using a combination of technologies that ensure scalability, security, and efficiency.

a) Programming Language: Python

- Python offers a rich ecosystem of libraries and frameworks that make it deal for building scalable web applications.
- Popular frameworks like FastAPI, Django, or Flask enable rapid development with features like dependency injection, data validation, and high performance.

b) Database Management: MySQL

- MySQL is a reliable relational database used to manage structured data like shipment records, invoices, and delivery tracking.
- Features like Joins, Stored Procedures, and Triggers are essential for managing complex logistics data.

c) ORM(Object-Relational Mapping): SQLAlchemy

- SQLAlchemy simplifies database interactions by allowing developers to write Pythonic code instead of raw SQL queries.
- It offers robust query handling, transactions, and schema migrations.

d) API Framework: FastAPI

- FastAPI is known for its speed, automatic Swagger documentation, and async support, making it perfect for real-time tracking and heavy data operations.

e) Authentication and Authorization

- Secure authentication can be implemented using JWT (JSON Web Tokens) for user session and role-based access control(RBAC).

f) Task Scheduling and Automation

- Celery is a popular tool for handling background tasks such as email notifications, periodic data synchronization, or automated report generation.

Literature Review

Academic research and best practices have greatly influenced modern TMS backend designs. Key references include:

- Micro services Architecture for Scalable Logistics Platforms: A study on decomposing TMS functionalities into independent services for better scalability.
- Database Optimization Techniques for Supply Chain Management Systems: Research on indexing strategies, query optimization, and caching solutions for faster data retrieval.
- FastAPI for High-Performance Web Applications: Articles showcasing FastAPI's performance benefits in data-intensive systems like TMS.
- AI-Driven Route Optimization Models: Studies leveraging machine learning for dynamic route planning and predictive analysis in transportation systems.

3.6 PROJECT PLANNING

1. Requirement Gathering:

- Identify project goals, key features, and user roles.
- Understand business logic (e.g., shipment tracking, POD uploads, etc.).
- Define data flow, API endpoints, and database schema.

Deliverables:

- Requirements document
- API endpoint list
- Database schema design

2. System Design:

- Design the backend architecture using **FastAPI**.
- Plan database structure with tables like BiltyMaster, Media, Branch, CompanyMaster, etc.
- Design ERD (Entity-Relationship Diagram) for MySQL database.
- Create a folder structure for backend code.

Deliverables:

- ERD diagram
- Backend architecture plan
- Folder structure design

3. Database Setup:

- Set up MySQL database with proper indexing and relationships.
- Create tables with necessary fields like bilty_id, pod_upload_date, etc.
- Implement sample data insertion for testing.

Deliverables:

- Database schema implementation
- SQL scripts for creating tables and inserting sample data

4. Backend Development:

- Develop core business logic using **FastAPI**.
- Implement endpoints for:
 - ◆ Shipment creation, updates, and deletion
 - ◆ POD upload and retrieval
 - ◆ Delivery tracking and status updates
- Develop pagination, sorting, and filtering mechanisms.
- Implement error handling and logging using **Loguru**.

Deliverables:

- Core API development
- CRUD operations for major entities
- Error handling implementation

5. Integration & Testing:

- Connect frontend (React.js) with the FastAPI backend.
- Perform unit testing for individual functions and endpoints.
- Conduct integration testing for end-to-end functionality.
- Use tools like **Postman** or **Swagger** for API testing.

Deliverables:

- Fully functional integration
- Test cases with results
- Bug fixes and optimizations

6. Deployment & Documentation:

- Deploy the backend on a server.
- Document API endpoints with detailed usage instructions.
- Provide database backup scripts and deployment steps.

Deliverables:

- Deployed backend with live endpoints
- API documentation
- Final Project Report

7. Maintenance & Support:

- Monitor system performance.
- Handle bug fixes and feature enhancements.
- Provide support for user queries and improvements.

3.6.1 Project Justification

The Transportation Management System (TMS) project is designed to address key challenges in logistics such as shipment tracking delays, mismanaged POD (Proof of Delivery) documents, and inefficient data retrieval.

By developing the backend using FastAPI, MySQL, and integrating React.js for the frontend, this system aims to:

- Improve shipment tracking accuracy by replacing `Media.created_date` with `BiltyMaster.pod_upload_date`.
- Automate shipment management processes to reduce manual effort.
- Provide fast and efficient data retrieval with optimized queries and pagination.

This solution enhances operational efficiency, ensures accurate delivery tracking, and offers a scalable architecture for future upgrades — making it a cost-effective and reliable solution for transportation management.

3.6.2 Cost estimation of the project

1. Labor Cost:

- Developers: Backend (FastAPI, Python), Frontend (React.js), and Database experts.
- QA Testers: Ensuring functionality, performance, and security.
- Project Manager: Overseeing timelines, tasks, and communication.

2. Equipment Cost:

- Hardware: Servers, storage devices, and networking tools.
- Software Licenses: IDEs (e.g., PyCharm), database tools (e.g., MySQL Workbench), and design tools (e.g., Figma).
- Cloud Services: Hosting, deployment, and database storage on platforms like AWS or Digital Ocean.

3. Trip/Logistics Cost:

- On-site Visits: For client meetings, requirement gathering, or deployment.
- Transportation: For attending stakeholder discussions or training sessions.

4. Training Cost:

- Training end-users on system usage, data entry, and report generation.

5. Maintenance & Support:

- Regular updates, bug fixes, and performance optimizations.

3.6.3 Roles and Responsibilities

During my backend development project at **Tech It Easy**, I played a key role in developing and enhancing the **Transportation Management System (TMS)**. The project aimed to improve shipment tracking, streamline POD (Proof of Delivery) management, and optimize data retrieval processes.

A successful project requires a well-defined structure where each team member understands their roles and responsibilities. Each individual contributed significantly to achieving the project goals. Below are the defined roles and responsibilities for the TMS project:

As a Backend Developer, my responsibilities included:

- Developing core business logic using FastAPI.
- Implementing data retrieval functions by replacing `Media.created_date` with `BiltyMaster.pod_upload_date` for improved POD tracking accuracy.
- Writing optimized SQL queries for fetching shipment details and managing large data sets.
- Ensuring the backend API endpoints were secure, efficient, and scalable.
- Collaborating with frontend developers to ensure seamless data integration.

3.6.4 Group Dependencies

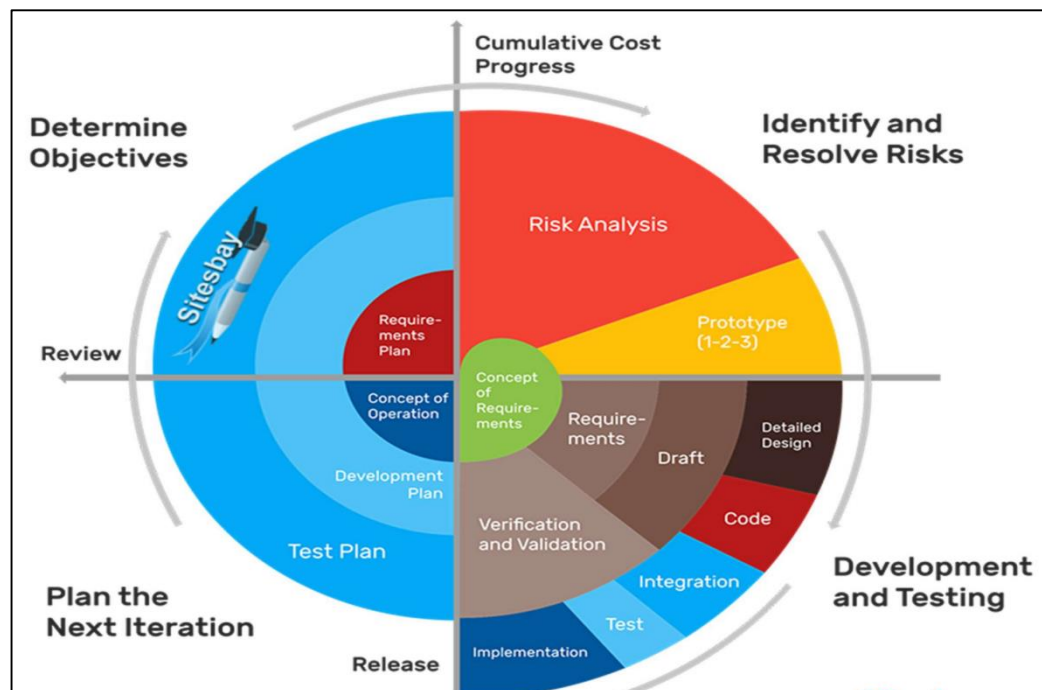
- Database Team: Coordination required for schema design, especially for tables like `BiltyMaster`, `Media`, and POD-related tables.
- Frontend Team: Provides API contracts (request/response formats), handles integration with the `React.js` UI.
- Bot Integration Team: Shares API endpoints for Telegram bot interactions (e.g., `null_valid_upto` logic and data filtering).
- Backend Team: Requires clear API endpoints that match user inputs like company selection, timeframe filters, etc. Provides logic for fetching, updating, or transforming data.
- Database Team: Needs structured data to display clean and meaningful summaries via bot messages.

3.7 PROJECT SCHEDULING

Week 1	<ul style="list-style-type: none"> • Diving into MySQL Fundamentals. • Mastering joins and operations. • Performed everything to the live database.
Week 2	<ul style="list-style-type: none"> • Understood the transportation system. • Learned about how it works, the process, different types and etc. • Learned about git.
Week 3	<ul style="list-style-type: none"> • Got the whole idea of FastAPI and CRUD operations. • Got the full backend of the software and the process. • Worked in live backend and made API and debugged the results.
Week 4	<ul style="list-style-type: none"> • Installed Conda in the system. • Setup conda in the backend. • Made my own local host database and made APIs using Get, Post, Put, Delete operations.
Week 5	<ul style="list-style-type: none"> • Started work in the live database. • Made tale and performed the get, put, post, update and delete operations. • Performed this operations with the use of loops and if-else condition.
Week 6	<ul style="list-style-type: none"> • Learned to make new branches. • Made a code in which image will get by bilty id then it will scan with the QR if it is not scanned it will scan from the OCR. • Learned how to push the changes permanently in backend. • Added rupee only in the bill statement PDF after total amount.

Week 7	<ul style="list-style-type: none"> Added new column scanned_from in the table of database. In image scan table added scanned_from having value of OCR, Manual and QR. Made changes in upload image by bilty_no added “B-QR” and “B-OCR”, it will differentiate the bulk image. Made changes in bill generation pdf.
Week 8	<ul style="list-style-type: none"> Added mr_no column from the money_receipt_details table with some condition. Made a new API named get_past_rate_for_pay_type. Added data_from column in the tbb_billing_statement.
Week 9	<ul style="list-style-type: none"> Made a new table in backend and database. Made API and CRUD to get the data by mf_no. Added created_by and created_from in the upload_pod_app API and database. Made API to get last_mf_no and to update the mf_id of marfatiya table if is_deleted is 0 to 1 and delete that column mf_id from the bilty_marfatiya table.
Week 10	<ul style="list-style-type: none"> Made an API in which data from the database of table marfatiya is changed or updated from the mf_id. And that mf_id record deleted from the bilty_marfatiya table. Made API to get all the data of gri, bilty and trip from the gri_no. Got knowledge about telegram bot and learned how to make bots in telegram with the got father.

Week 11	<ul style="list-style-type: none">• Worked on the telegram chatbot project.• After the installation, made an API to make report of e-waybills PDF in e-waybill software backend.• Added this API in the telegram project so that after clicking on saral e-way user receive e-waybill report PDF.
Week 12	<ul style="list-style-type: none">• Made tbb_billing_statement API and changed the print positions.• Made pod-receive-report API and get the all pod_receive_report data.



3.1 Spiral Model

CHAPTER 4: SYSTEM ANALYSIS

4.1 STUDY OF CURRENT SYSTEM

Before embarking on the development of a Transportation Management System(TMS), it's essential to conduct a thorough study of the current system or processes in place. This study provides valuable insights into existing workflows, pain points, and areas for improvement. Here's how the study of the current system for transportation management may be conducted:

- 1. Identify Stakeholders:** Begin by identifying all stakeholders involved in the transportation management process. This may include logistic companies, transportation service providers, drivers, warehouse managers, dispatchers, and customers expecting timely deliveries.
- 2. Document Current Processes:** Document the current transportation management processes and workflows in detail. This includes how shipment request are received, how delivery routes are planned, how vehicle assignments are managed, and how tracking, invoicing, and reporting are conducted.
- 3. Gather Feedback:** Interview key stakeholders to gather feedback on the current transportation management process. This includes logistic companies assessing their operational efficiency, drivers reporting on delivery challenges, and dispatchers sharing their experiences with route planning and task assignments.
- 4. Analyze Data and Reports:** Review existing data and reports generated from transportation activities. Analyze trends, patterns, and anomalies to identify areas of strength and areas for improvement. Look for common themes or recurring issues that may indicate systemic problems.
- 5. Assess Technology Stack:** Evaluate the technology stack currently used to support transportation management activities. This includes any software applications, tools, or platforms used for route optimization, vehicle tracking,

order management, and reporting. Assess the effectiveness and limitations of these technologies.

- 6. Identify Challenges and Pain Points:** Identify challenges, pain points, and bottlenecks in the current transportation management process. This may include issues such as manual route planning, delay updates on delivery status, difficulty tracking, POD(Proof of Delivery) documents, or inefficiencies in communication between stakeholders.
- 7. Evaluate Compliance and Quality:** Assess the level of compliance with industry standards, safety regulations, and internal guidelines. Evaluate the accuracy of delivery records, timeliness of shipments, and reliability of the data collected throughout the process.
- 8. Document Findings:** Document the findings of the study, including observations, feedback from stakeholders, data analysis results, and identified challenges and pain points. Use this documentation as a basis for identifying requirements and designing solutions for the transportation management system.

4.2 PROBLEMS AND WEAKNESS OF CURRENT SYSTEM

The current transportation system faces several issues, including manual processes, lack of real-time tracking, and poor communication between stakeholders. Data is scattered across systems, making reporting and analysis inefficient. Proof of Delivery (POD) management is delayed and unorganized. There's minimal automation, leading to high operational effort and errors. Additionally, compliance and quality control are difficult to maintain, and outdated technology limits the system's scalability and efficiency.

4.3 REQUIREMENTS OF NEW SYSTEM

When conducting a study for the development of a new Transportation Management System(TMS), it's crucial to thoroughly analyze the requirements, objectives, and constraints of the project. Here's how the study of the new system for transportation management may be conducted:

- 1. Define Objectives:** Clearly define the objectives of implementing the new TMS. Determine the goals you aim to achieve, such as improving delivery efficiency, enhancing shipment tracking, or increasing operational transparency.
- 2. Identify Stakeholders:** Identify all stakeholders who will be involved in the new TMS. This includes logistics companies, transportation service providers, drivers, warehouse managers, dispatchers and customers expecting timely deliveries.
- 3. Gather Requirements:** Conduct interviews, surveys, and workshops with stakeholders to gather requirements for the new TMS. Identify functional requirements related to shipment assignment, route planning, vehicle tracking, data analysis, reporting, and user roles and permissions. Gather non-functional requirements such as security, scalability, usability, and performance.
- 4. Analyse current Processes:** Analyze the current transportation management processes and workflows to identify areas for improvement. Compare the current processes with the desired objectives of the new system and identify gaps or inefficiencies that need to be addressed.
- 5. Review Industry Standards:** Research industry best practices and standards for transportation management systems. Review existing TMS platforms and solutions to understand common features, functionalities, and user experiences. Identify opportunities to incorporate the industry best practices into the new system.
- 6. Assess Technology Stack:** Evaluate the technology stack required to support the new TMS. Determine the software applications, tools, and platforms, tools, and platforms needed for task management, data collection and analysis, reporting, communication, and user interface design. Assess the feasibility and compatibility of different technologies.
- 7. Consider Compliance and Quality:** Ensure that the new TMS complies with industry regulations and standards. Incorporate mechanisms for quality assurance and compliance monitoring to ensure that data is captured accurately and securely.
- 8. Design System Architecture:** Design the system architecture for the new TMS. Define the overall structure, components, and interactions of the system, including databases, servers, APIs, user interfaces, and integrations with external systems.

4.4 SYSTEM FEASIBILITY

System feasibility assessment is essential to determine whether the proposed Transportation Management System(TMS) project is technically, economically, and operationally feasible. Here's how the feasibility of the project can be evaluated:

1. **Technical Feasibility:**

- Evaluate the technical requirements and capabilities needed to develop and implement the TMS.
- Assess the availability of necessary hardware, software, and technology infrastructure to support the system.

2. **Economic Feasibility:**

- Conduct a cost-benefit analysis to assess the economic viability of the project.
- Estimate the development costs, including hardware, software, labor, and other expenses.

3. **Operational Feasibility:**

- Evaluate the practicality and usability of the TMS from an operational perspective..

4. **Legal and Regulatory Feasibility:**

- Ensure compliance with legal and regulatory requirements governing data privacy, security, and transportation standards.
- Assess any potential legal or regulatory barriers that may affect the development and implementation of the TMS..

5. **Schedule Feasibility:**

- Evaluate the project timeline and schedule to determine whether the proposed timeline is realistic and achievable.
- Consider factors such as resource availability, dependencies, and potential risks that may impact the project schedule.

4.4.1 Does the system contribute to the overall objectives of the organization?

Yes, the proposed Transportation Management System (TMS) significantly contributes to the overall objectives of the organization. It enhances operational efficiency, ensures real-time tracking and visibility, improves data accuracy, and streamlines shipment and delivery management. With modules for user control, POD verification, reporting, communication, and integration, the system supports better decision-making, ensures regulatory compliance, and promotes customer satisfaction. It also reduces manual work, minimizes delays, and aligns with the organization's goal of delivering reliable, scalable, and technology-driven logistics solutions.

4.4.2 Can the system be implemented using the current technology and within the given cost and schedule constraints

Yes, the proposed Transportation Management System (TMS) can be implemented using the current technology stack, which includes FastAPI for backend, MySQL for database management, and React.js for frontend development. These technologies are widely used, open-source, and cost-effective, reducing overall project expenses.

Moreover, based on the feasibility analysis, the system can be developed within the estimated budget and timeline, provided that resources are allocated efficiently, and risks are managed properly. With proper planning, collaboration, and phased development, the project is technically, economically, and operationally viable.

4.4.3 Can the system be integrated with other systems which are already in place?

Yes, the Transportation Management System (TMS) is designed with integration in mind. It includes a dedicated Integration Module that allows seamless connectivity with existing systems such as ERP solutions, GPS tracking tools, accounting software, and third-party logistics platforms. The use of RESTful API endpoints ensures interoperability and smooth data exchange, enabling the TMS to complement and enhance the functionality of systems already in place without disrupting current operations.

4.5 PROCESS IN NEW SYSTEM

The new system enables user registration, shipment creation, and automated task assignment to drivers. It offers real-time tracking, optimized route planning, and POD submission with digital verification. Communication tools and feedback options improve coordination, while built-in analytics provide performance insights. The system supports quality checks, admin control, user support, and integration with external tools like ERP, GPS, and accounting systems.

4.6 FEATURES OF NEW SYSTEM

The system offers role-based user access, shipment creation and tracking, and digital POD uploads. It includes real-time updates, automated route planning, and performance reporting. Users receive alerts and messages, while admins perform quality checks. The system also supports integration with external tools and provides user support and training resources.

4.7 LIST MAIN MODULES

1. User Management Module:

- Registration and authentication of users (logistics companies, drivers, dispatchers, and administrators).
- User profile management.
- Role-based access control to define permissions and privileges for different user roles.

2. Shipment Management Module:

- Creation and management of shipment requests by administrators or business users.
- Assignment of shipment tasks to registered drivers.
- Task scheduling, including shipment deadlines and priorities.
- Tracking of shipment status and progress.

3. POD (Proof of Delivery) Submission Module:

- Submission of POD documents by drivers after completing assigned deliveries.
- Capture of POD data, including signatures, timestamps, and multimedia evidence(photos, receipts,etc.).
- Validation and verification of submitted POD by administrators.

4. Data Analysis Module:

- Analysis of transportation data to identify trends, patterns, and insights.
- Generation of reports and dashboards to visualize shipment performance .
- Comparative analysis of performance metrics across different locations, time periods,or criteria.

5. Feedback and Communication Module:

- Communication channels for logic companies, drivers, and administrators.
- Feedback mechanisms for stakeholders to provide comments, suggestions, and complaints.
- Notifications and alerts for shipment assignments, deadlines, and updates.

6. Reporting Module:

- Generation of comprehensive reports based on shipment and delivery data.
- Customization of report templates and formats.
- Exporting and sharing of reports with stakeholders.

7. Administration and Configuration Module:

- Configuration of system settings and parameters.
- Management of user roles, permissions, and access control lists.
- Maintenance of master data, including business profiles, driver profiles, and delivery criteria.

8. Quality Assurance Module:

- Quality control mechanisms to ensure the accuracy and reliability of delivery data.
- Review and validation of delivery records by administrators or quality assurance teams.
- Resolution of discrepancies or issues identified during quality checks.

9. Training and Support Module:

- Provision of training materials and resources for drivers and dispatchers.
- User guides, FAQs, and knowledge base for self-help support.
- Helpdesk or customer support ticketing system for resolving user inquiries and issues.

10. Integration Module:

- Integration with external systems or platforms, such as ERP systems, GPS tracking systems, or accounting tools.
- API endpoints for data exchange and interoperability with third-party applications.

CHAPTER 5: SYSTEM DESIGN

5.1 BLOCK DIAGRAM

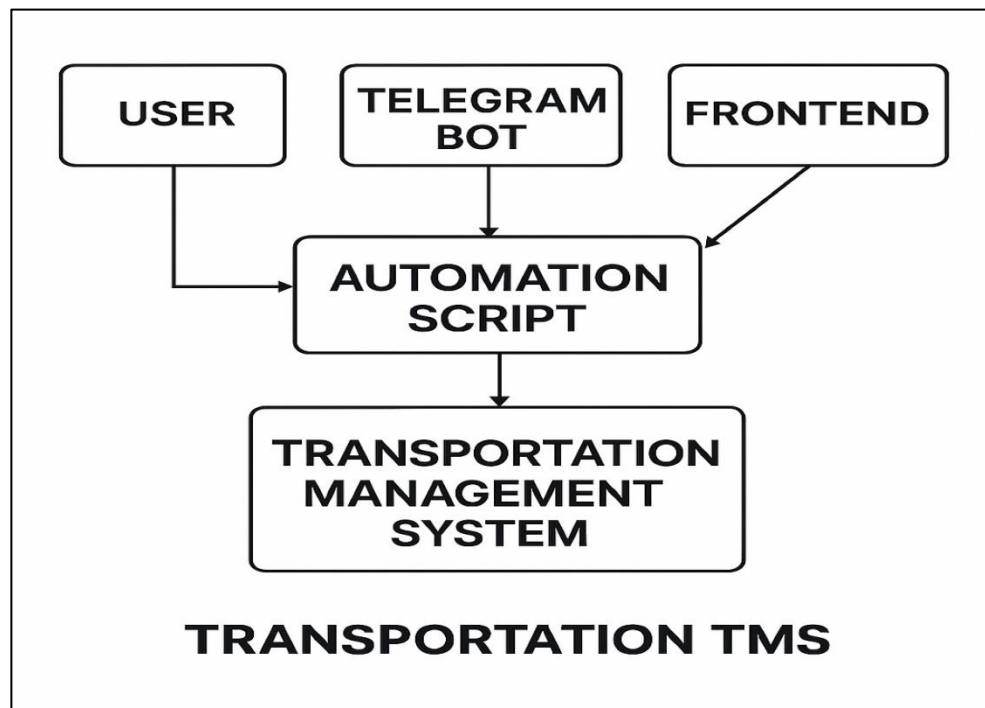


Fig 5.1 Block Diagram

User (Dispatcher / Admin / Logistics Company)

Acts as the main actor who interacts with the system. The user initiates actions such as shipment creation, driver assignment, and monitoring delivery progress.

TMS Web Interface (Frontend - React.js)

A user-friendly interface where users can input shipment details, view dashboards, manage users, and track deliveries. It serves as the bridge between the user and the backend system.

TMS Backend API (FastAPI)

Handles all core logic and processes behind the scenes. It manages:

- Shipment assignment
- POD uploads
- Real-time data updates
- Route optimization

5.2 FLOW CHART DIAGRAM

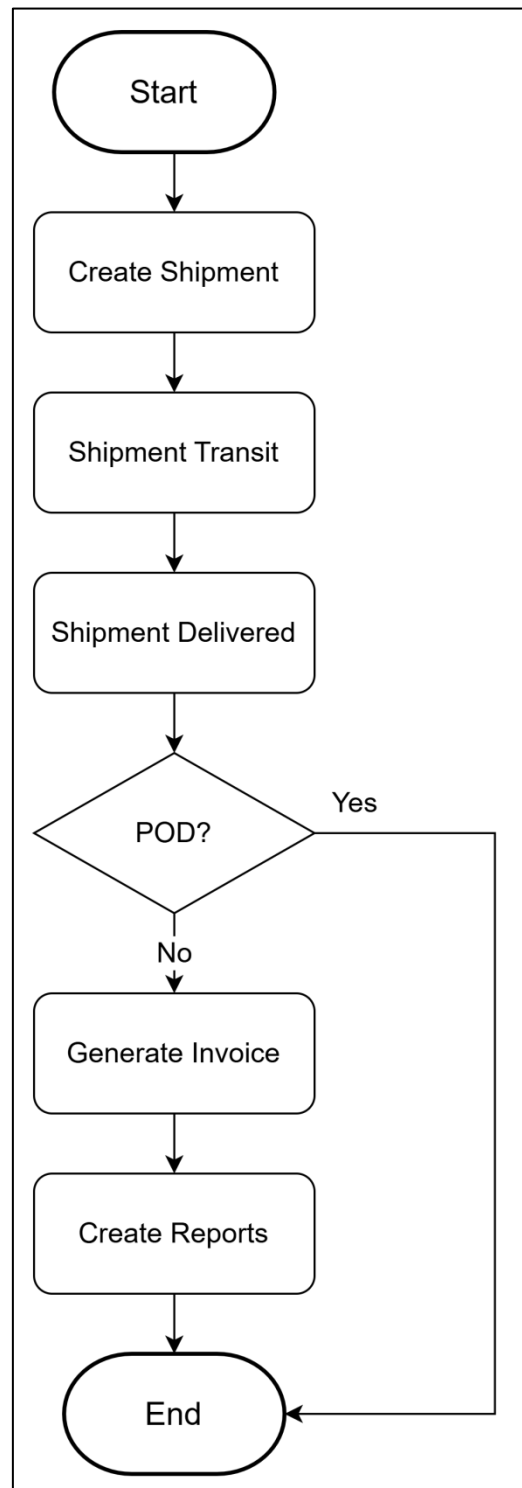


Fig 5.2 Flow Chart Diagram

5.3 ER DIAGRAM

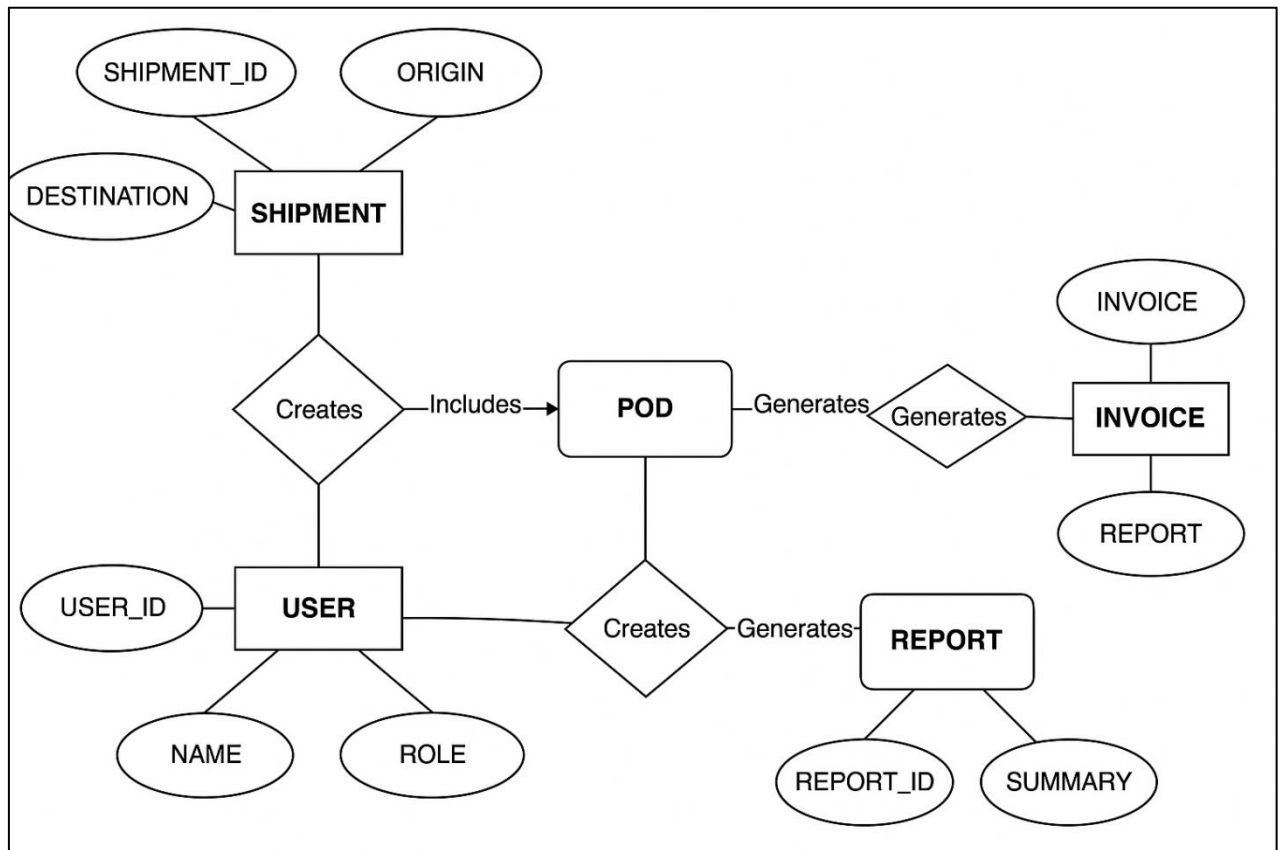


Fig 5.3 ER Diagram

5.4 USE CASE DIAGRAM

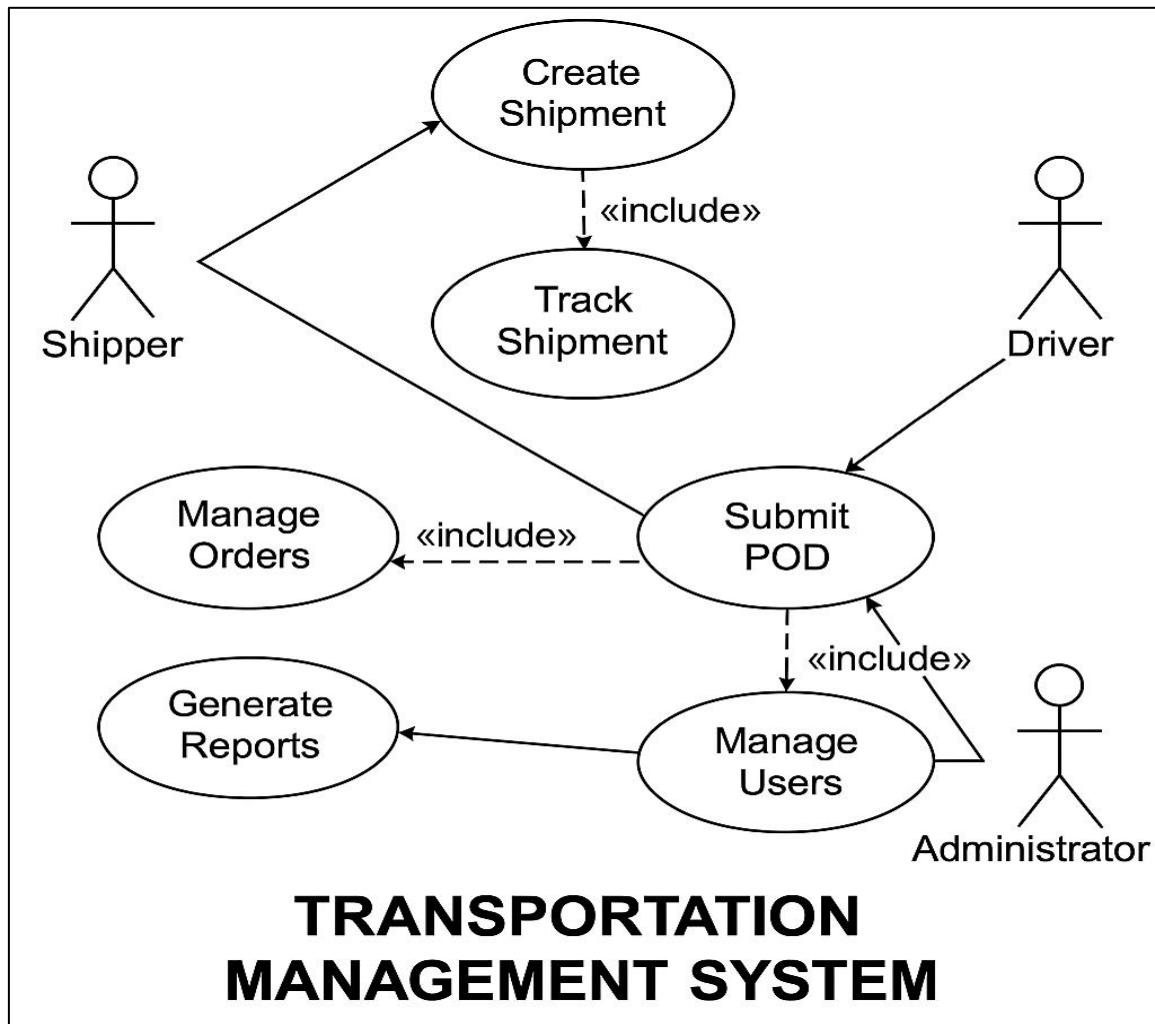


Fig 5.4 Use Case Diagram

5.5 SEQUENTIAL DIAGRAM

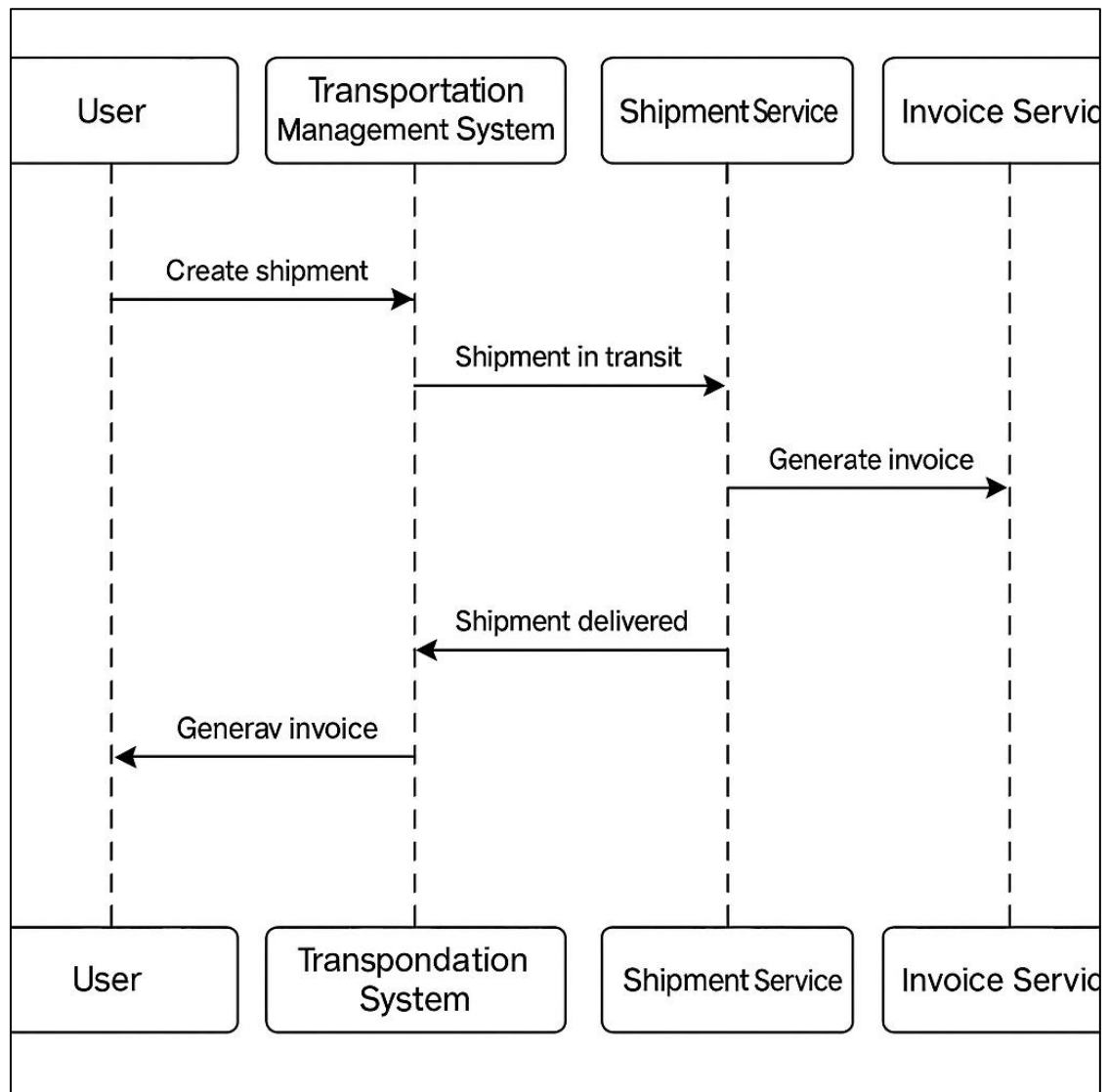


Fig 5.5 Sequential Diagram

CHAPTER 6: IMPLEMENTATION

6.1 IMPLEMENTATION PLATFORM

- **Frontend** : React.js
- **Backend** : FastAPI
- **Database** : MySQL
- **Bot** : Telegram Bot
- **Automation** : Python Script
- **Deployment** : Swagger UI

Field	Datatype	Constraint
id	Integer	Primary Key, Not Null
bilty_id	Integer	Null
booking_chalan_id	Integer	Null

6.1.1 bilty_chalan table from RCC database

Field	Datatype	Constraint
id	Int	Primary Key, Not Null
item_id	Int	Null
bilty_id	Int	Null
unit	Varchar(10)	Null
pkgs	Int	Null
weight	Float	Null
rate	Float	Null
amount	Float	Null
packing_type	Int	Null
truck_size	Varchar(20)	Nul

6.1.2 item-bilty table from RCC database

Field	Datatype	Constraint
Id	Int	Primary Key, Not Null
name	Varchar(100)	Null
parent_menu_id	Int	Null
is_main	Tinyint(1)	Null
display_name	Varchar(50)	Null

6.1.3 menu table from RCC database

Field	Datatype	Constraint
id	Int	Primary Key, Not Null
bilty_no	Int	Null
booking_chalan_id	Int	Null
bilty_date	Datetime	Null
ewb_no	Bigint	Null

6.1.4 separate_bilty_chalan table from RCC

6.2 MODULE SPECIFICATION

Authentication Module

The Authentication Module forms the backbone of security in the TMS. Using FastAPI, the system supports secure user login, registration, and access control through OAuth2 and JWT-based authentication. Users such as dispatchers, admins, and drivers authenticate using a username-password combination. Passwords are securely stored using hashing algorithms like bcrypt. On successful login, a JWT token is issued, which is required for subsequent requests to protected endpoints. The system also supports role-based access control, ensuring that only users with proper privileges can access sensitive features such as shipment management or report generation.

Shipment Management

The Shipment Management Module serves as the core operational layer of the TMS, where logistics activities such as creating shipment requests, assigning drivers, updating shipment statuses, and tracking deliveries are handled. It allows dispatchers or admins to input shipment details, select preferred delivery windows, and match available drivers and vehicles for optimal efficiency. The shipment data is dynamically linked with user and vehicle information, ensuring that all parties involved are informed in real time. Each shipment goes through lifecycle stages such as "Created," "In-Transit," and "Delivered," which are updated based on the driver's activity and admin inputs. This helps maintain complete visibility into ongoing logistics operations.

POD(Proof of Delivery) Module

The POD (Proof of Delivery) Module facilitates secure and verifiable confirmation of successful deliveries. Drivers can upload PODs using a mobile-friendly interface that supports image capture, digital signatures, and automatic timestamping. These submissions are reviewed by the admin or dispatcher for authenticity before marking the delivery as complete. The module ensures that every POD is uniquely tied to its corresponding shipment and driver, maintaining accountability. The system also flags anomalies like delayed uploads or mismatched delivery times, enabling faster resolution of disputes or delivery issues.

FastAPI

All interactions in the system are managed by the API Communication Module, built using FastAPI. The backend exposes well-structured RESTful APIs for frontend interfaces, mobile applications, and third-party systems. These APIs cover user registration, login, shipment creation, POD submission, and real-time updates. JWT tokens secure every endpoint, and FastAPI's integration with Pydantic ensures robust request validation and response formatting. These APIs also support webhook callbacks and data syncing, enabling integrations with external ERP systems, GPS trackers, and analytics platforms.

Database and Logging Module

Data storage and system monitoring are handled by the Database and Logging Module, which uses MySQL or PostgreSQL to persist all critical information, including user details, shipment records, POD logs, and feedback.

Reporting and Analytics Module

The Reporting and Analytics Module offers detailed reports and dashboards on transport metrics such as delivery timeframes, driver performance, delayed shipments, and customer feedback. These reports are generated based on historical data and are presented in downloadable formats like PDF and Excel. The analytics engine can also trigger alerts based on performance thresholds or exceptions, enabling proactive decision-making.

Notification and Feedback Module

A Notification and Feedback Module ensures smooth communication within the system. Real-time notifications are sent to users regarding shipment assignments, delivery delays, and POD upload reminders. These notifications can be configured to be delivered via SMS, email, or push alerts. The system also allows users and clients to submit feedback after each delivery, which is logged and reviewed by the admin for service improvement.

Testing and Quality Assurance

The entire system is validated through the Testing and Quality Assurance Module. Using pytest and FastAPI's TestClient, the system undergoes rigorous unit testing and integration testing for all endpoints and business logic. Edge cases such as invalid inputs, expired tokens, and delivery mismatches are simulated and handled gracefully. During development, fake data and mock APIs are used to simulate real-world logistics scenarios without affecting production data. This module guarantees the robustness, scalability, and fault-tolerance of the TMS.

6.3 OUTCOMES

Efficient Transport Workflow Automation

The Transportation Management System (TMS) automates and simplifies the end-to-end process of managing transport operations, from creating shipments to final delivery with POD upload. By using FastAPI as the backend engine and integrating with a modern frontend (like React), the system eliminates manual bottlenecks, accelerates operations, and provides a centralized platform for dispatchers, drivers, and administrators. As a result, organizations benefit from improved efficiency and reduced paperwork, ultimately saving time and operational costs.

Real-Time POD Upload

Drivers can upload Proof of Delivery (POD) documents directly from the field using the portal, which are instantly updated in the system. This real-time data handling reduces delays in reporting, minimizes errors from manual entry, and ensures timely verification of completed deliveries.

Enhanced User Interaction and Remote Management

With a user-friendly web interface and secure backend access, users can manage transport activities remotely. Admins and dispatchers can assign shipments, monitor routes, and validate delivery statuses, while drivers receive their assignments and upload PODs without needing in-person coordination. This remote management capability greatly benefits teams operating across multiple geographic locations.

Secure Access and Role-Based Control

The system incorporates robust security using JWT tokens and role-based access controls to restrict user capabilities according to their responsibilities. Admins, dispatchers, and drivers have distinct roles, and access to sensitive actions is governed accordingly. Passwords are securely hashed, sessions are tracked, and all API interactions are validated—ensuring that only authorized actions are executed within the platform.

Comprehensive Data Logging and Operational Monitoring

Every action—from user login to POD uploads and shipment updates—is logged with accurate timestamps and statuses. This structured logging helps administrators audit the workflow, trace issues, and monitor the overall system performance. These logs also serve as a source of historical data for compliance checks and business intelligence analysis.

Scalability and Modular Architecture

The TMS is built with scalability in mind, allowing it to adapt as the transport network or number of users grows. Its modular backend makes it easy to plug in new features such as GPS tracking, SMS/email notifications, or analytics dashboards. Each module, like shipment management or POD handling, can be updated independently, ensuring long-term maintainability and flexibility for future enhancements.

6.4 SCREENSHOTS

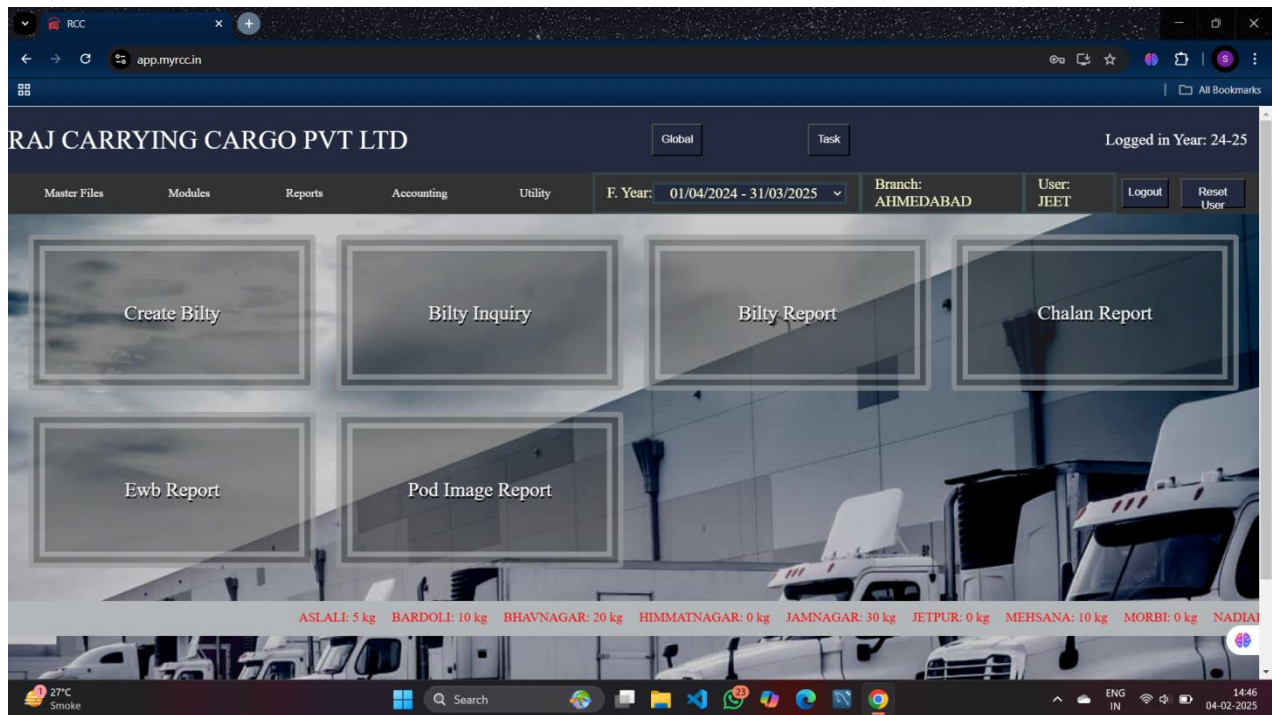


Fig 6.4.1 Software Dashboard

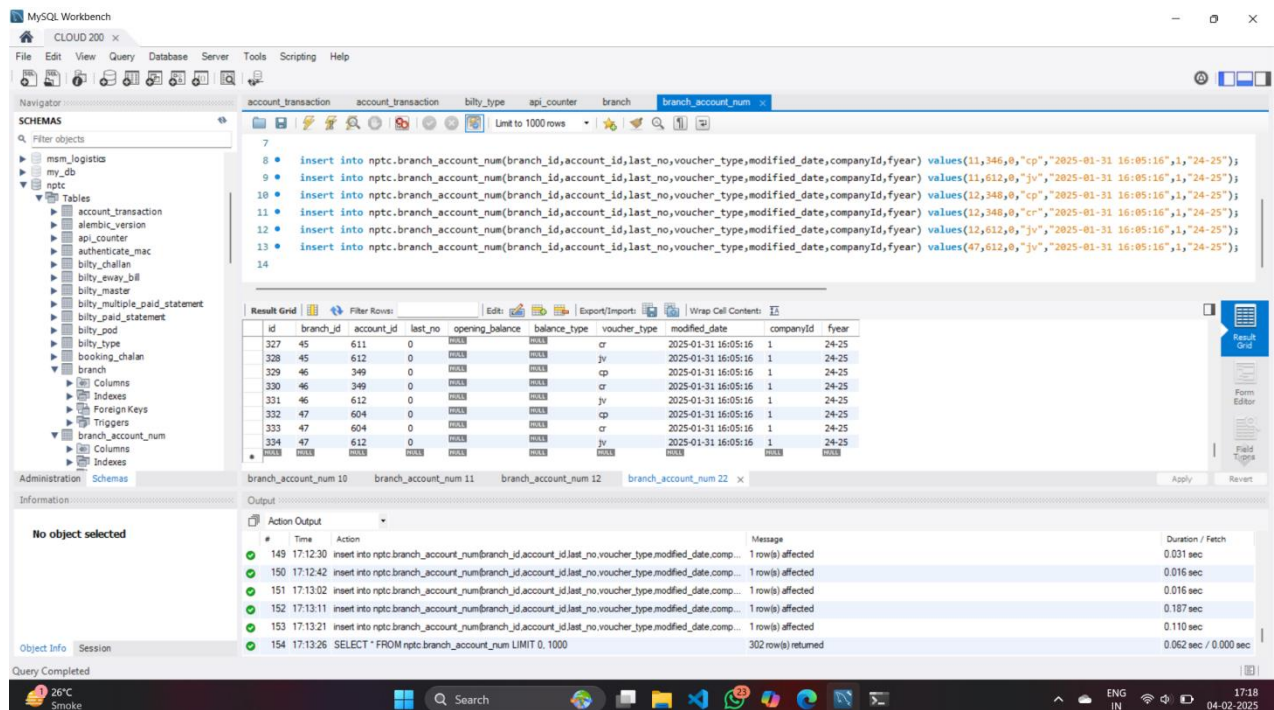


Fig 6.4.2 Table Creation with MySQL

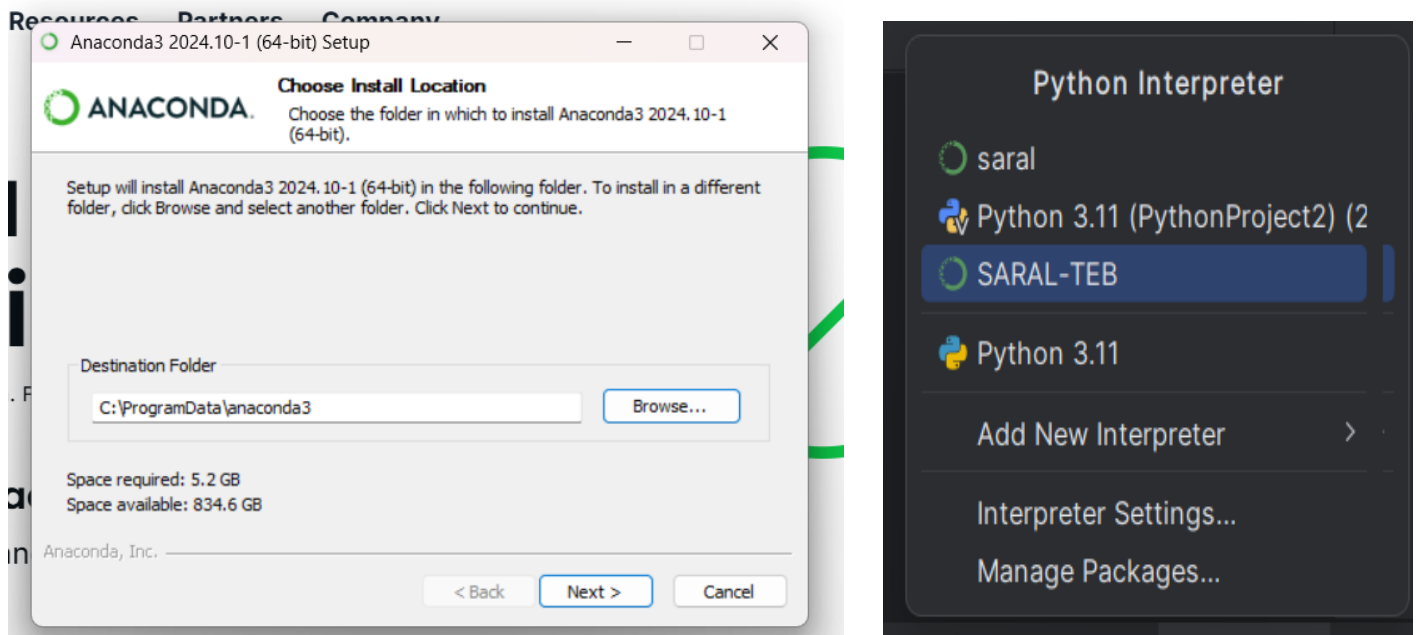


Fig 6.4.3 Installation and Setting Up Conda in the Backend

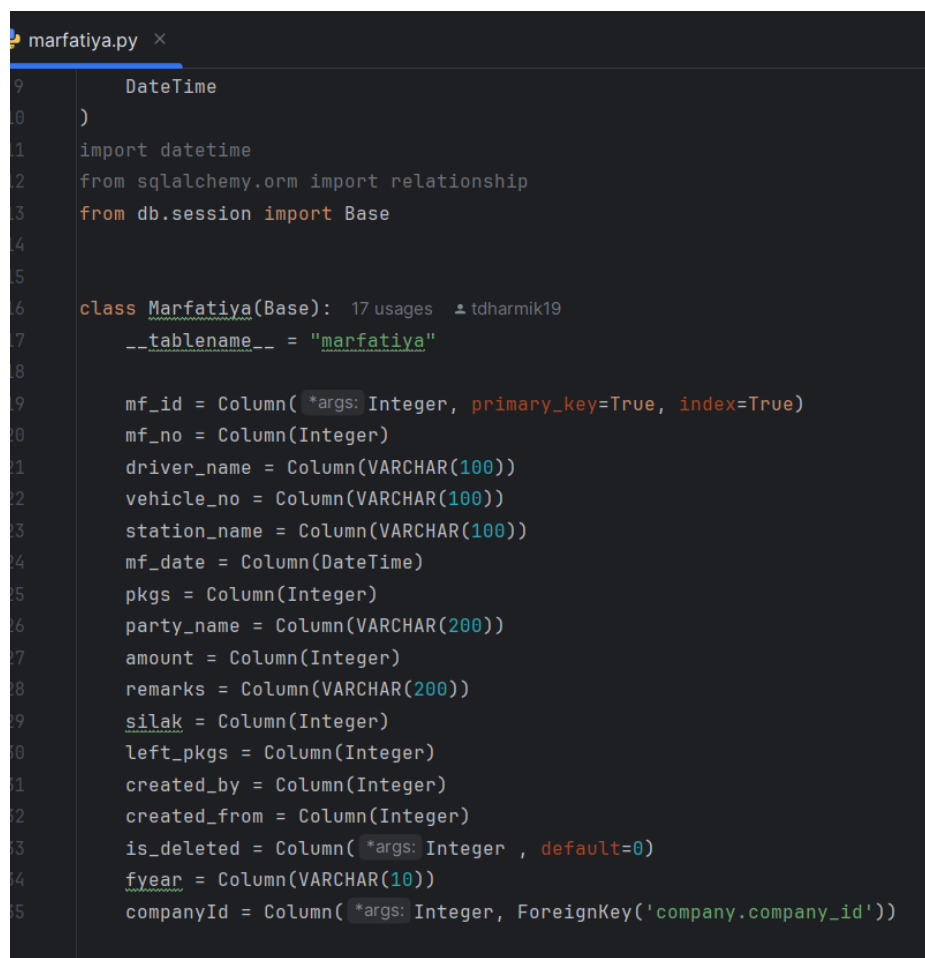
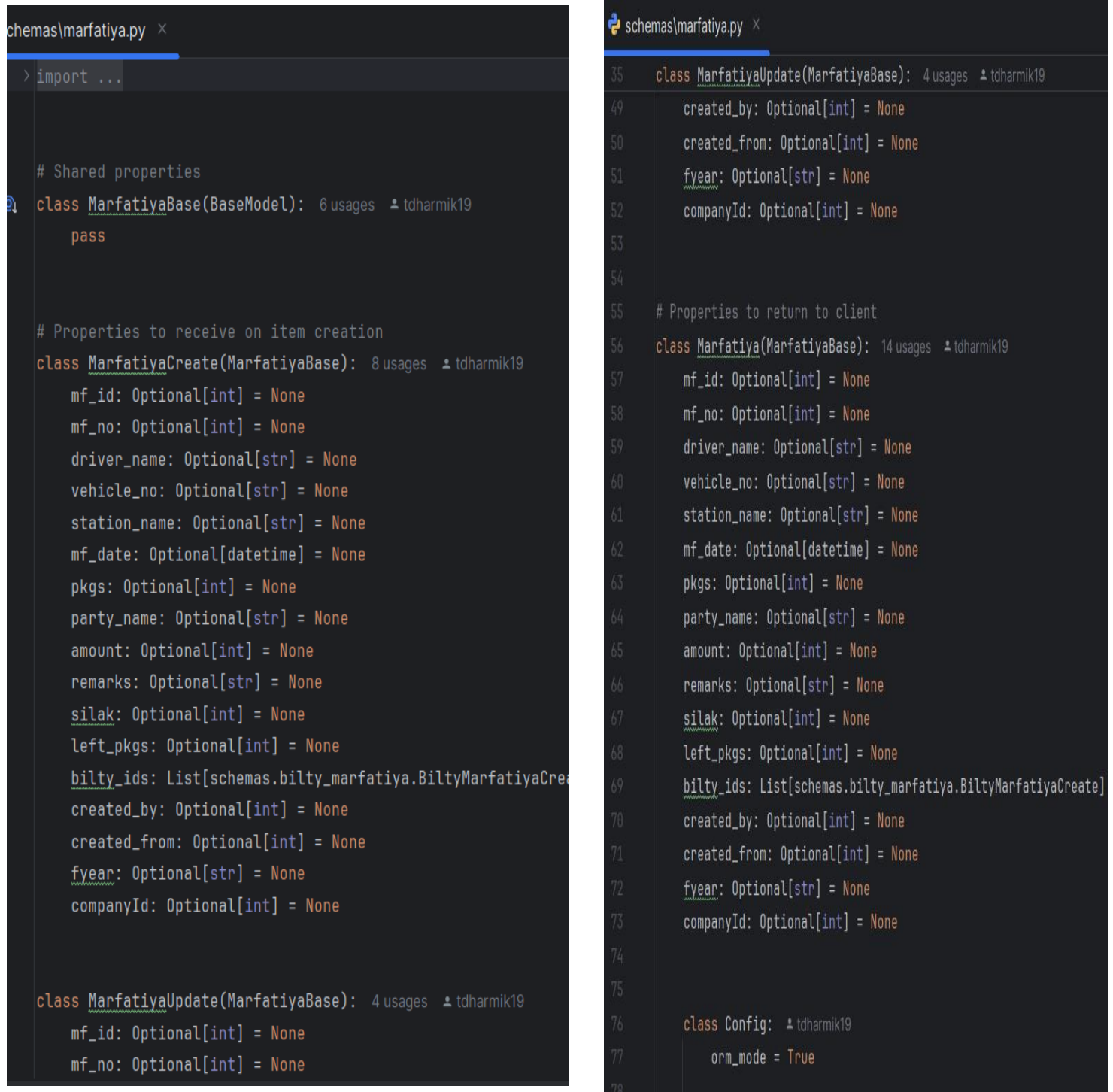


Fig 6.4.4 Addition of marfatiya table into Models of Backend



```

schemas\marfatiya.py x
> import ...

# Shared properties
class MarfatiyaBase(BaseModel): 6 usages  tdharmik19
    pass

# Properties to receive on item creation
class MarfatiyaCreate(MarfatiyaBase): 8 usages  tdharmik19
    mf_id: Optional[int] = None
    mf_no: Optional[int] = None
    driver_name: Optional[str] = None
    vehicle_no: Optional[str] = None
    station_name: Optional[str] = None
    mf_date: Optional[datetime] = None
    pkgs: Optional[int] = None
    party_name: Optional[str] = None
    amount: Optional[int] = None
    remarks: Optional[str] = None
    silak: Optional[int] = None
    left_pkgs: Optional[int] = None
    bilty_ids: List[schemas.bilty_marfatiya.BiltyMarfatiyaCreate]
    created_by: Optional[int] = None
    created_from: Optional[int] = None
    fyear: Optional[str] = None
    companyId: Optional[int] = None

class MarfatiyaUpdate(MarfatiyaBase): 4 usages  tdharmik19
    mf_id: Optional[int] = None
    mf_no: Optional[int] = None

schemas\marfatiya.py x
35 class MarfatiyaUpdate(MarfatiyaBase): 4 usages  tdharmik19
49     created_by: Optional[int] = None
50     created_from: Optional[int] = None
51     fyear: Optional[str] = None
52     companyId: Optional[int] = None
53
54
55 # Properties to return to client
56 class Marfatiya(MarfatiyaBase): 14 usages  tdharmik19
57     mf_id: Optional[int] = None
58     mf_no: Optional[int] = None
59     driver_name: Optional[str] = None
60     vehicle_no: Optional[str] = None
61     station_name: Optional[str] = None
62     mf_date: Optional[datetime] = None
63     pkgs: Optional[int] = None
64     party_name: Optional[str] = None
65     amount: Optional[int] = None
66     remarks: Optional[str] = None
67     silak: Optional[int] = None
68     left_pkgs: Optional[int] = None
69     bilty_ids: List[schemas.bilty_marfatiya.BiltyMarfatiyaCreate]
70     created_by: Optional[int] = None
71     created_from: Optional[int] = None
72     fyear: Optional[str] = None
73     companyId: Optional[int] = None
74
75
76 class Config:  tdharmik19
77     orm_mode = True
78

```

Fig 6.4.5 Addition of table into Schemas of Backend

```

schemas.tracking_status import TrackingStatus, TrackingStatusCreate, TrackingStatusUpdate
from schemas.transporter_master import TransporterMaster, TransporterMasterCreate, TransporterMasterUpdate
from schemas.trip_bhada import TripBhada, TripBhadaCreate, TripBhadaUpdate
from schemas.trip_master import TripMaster, TripMasterCreate, TripMasterUpdate, TripPayableCreate
from schemas.user import User, UserCreate, UserUpdate, UserPost
from schemas.user_branch import UserBranch, UserBranchCreate, UserBranchUpdate
from schemas.user_role import UserRole, UserRoleUpdate, UserRoleCreate
from schemas.vehicle_master import VehicleMaster, VehicleMasterCreate, VehicleMasterUpdate
from schemas.vehicle_type import VehicleType, VehicleTypeCreate, VehicleTypeUpdate
from schemas.vehicle_register import VehicleRegister, VehicleRegisterCreate, VehicleRegisterUpdate
from schemas.bilty_multiple_paid_statement import BiltyMultiplePaidStatementCreate, BiltyMultiplePaidStatement, BiltyMultiplePaidStatementUpdate
from schemas.branch_account_num import BranchAccountNum, BranchAccountNumCreate, BranchAccountNumUpdate
from schemas.authenticate_mac import AuthenticateBase, AuthenticateCreate, AuthenticateUpdate, AuthenticateRead
from schemas.separate_booking_challan import SeparateBookingChallanBase, SeparateBookingChallanCreate, SeparateBookingChallanUpdate, SeparateBookingChallanRead
from schemas.separate_bilty_challan import SeparateBiltyChallanBase, SeparateBiltyChallanCreate, SeparateBiltyChallanUpdate, SeparateBiltyChallanRead
from schemas.company_mail import CompanyMail, CompanyMailBase, CompanyMailUpdate, CompanyMailCreate
from schemas.pending_mr_register import PendingMrRegisterCreate, PendingMrRegisterUpdate, PendingMrRegisterBase, PendingMrRegisterRead
from schemas.last_branch import LastBranch, LastBranchUpdate, LastBranchBase, LastBranchCreate
from schemas.token_verification import TokenVerificationCreate, TokenVerificationUpdate, TokenVerificationBase, TokenVerificationRead
from schemas.mail_master import MailMaster, MailMasterUpdate, MailMasterCreate, MailMasterBase
from schemas.user_dashboard import UserDashboard, UserDashboardBase, UserDashboardCreate, UserDashboardUpdate
from schemas.hisab_statement import HisabStatement, HisabStatementCreate, HisabStatementUpdate, HisabStatementBase
from schemas.fleet_master import FleetMasterBase, FleetMasterCreate, FleetMasterUpdate, FleetMasterRead
from schemas.stock_register import StockRegisterCreate, StockRegisterUpdate, StockRegisterBase, StockRegisterRead
from schemas.marfatiya import MarfatiyaCreate, MarfatiyaUpdate, MarfatiyaBase, MarfatiyaRead
from schemas.bilty_marfatiya import BiltyMarfatiyaCreate, BiltyMarfatiyaUpdate, BiltyMarfatiyaBase, BiltyMarfatiyaRead

```

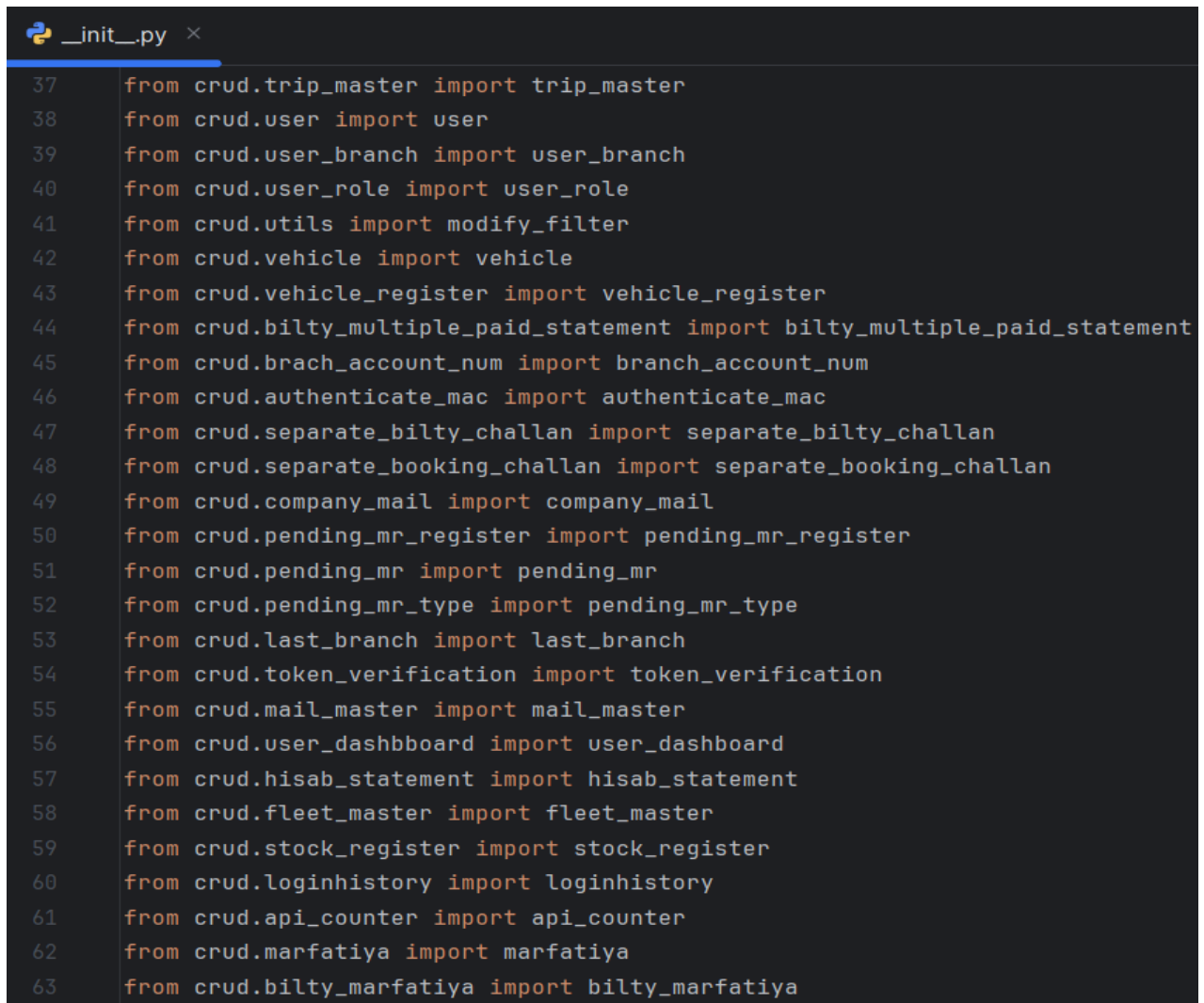
Fig 6.4.5 Addition of table in the init of schemas

```

models._init_.py
28 from models.bilty_paid_statement import BiltyPaidStatement
29 from models.mr_statement import MrStatement
30 from models.tbb_billing_statement import TbbBillingStatement
31 from models.eway_bill import EwayBill
32 from models.crossing_bill_inward import CrossingBillInward
33 from models.crossing_bill_outward import CrossingBillOutward
34 from models.pod_chalan import PodChalan
35 from models.charge_master import ChargeMaster
36 from models.group_master import GroupMaster
37 from models.subgroup_master import SubGroupMaster
38 from models.account_transaction import AccountTransaction
39 from models.narration_master import NarrationMaster
40 from models.vehicle_register import VehicleRegister
41 from models.bilty_multiple_paid_statement import BiltyMultiplePaidStatement
42 from models.branch_account_num import BranchAccountNum
43 from models.company_mail import CompanyMail
44 from models.pending_mr_register import PendingMrRegister
45 from models.last_branch import LastBranch
46 from models.token_verification import TokenVerification
47 from models.mail_master import MailMaster
48 from models.user_dashboard import UserDashboard
49 from models.hisab_statement import HisabStatement
50 from models.fleet_master import FleetMaster
51 from models.stock_register import StockRegister
52 from models.api_counter import ApiCounter, ApiCounterBase, ApiCounterCreate
53 from models.marfatiya import Marfatiya
54 from models.bilty_marfatiya import BiltyMarfatiya

```

Fig 6.4.6 Addition of table in the init of models

A screenshot of a code editor window titled '__init__.py' with a close button. The editor contains 27 lines of Python code, each starting with a line number from 37 to 63. Each line is an import statement from the 'crud' module, importing a specific class or function. The imports are: trip_master, user, user_branch, user_role, modify_filter, vehicle, vehicle_register, bilty_multiple_paid_statement, branch_account_num, authenticate_mac, separate_bilty_challan, separate_booking_challan, company_mail, pending_mr_register, pending_mr, pending_mr_type, last_branch, token_verification, mail_master, user_dashboard, hisab_statement, fleet_master, stock_register, loginhistory, api_counter, marfatiya, and bilty_marfatiya. The code is syntax-highlighted with orange for 'from', blue for 'import', and black for the module and class names.

```
37 from crud.trip_master import trip_master
38 from crud.user import user
39 from crud.user_branch import user_branch
40 from crud.user_role import user_role
41 from crud.utils import modify_filter
42 from crud.vehicle import vehicle
43 from crud.vehicle_register import vehicle_register
44 from crud.bilty_multiple_paid_statement import bilty_multiple_paid_statement
45 from crud.brach_account_num import branch_account_num
46 from crud.authenticate_mac import authenticate_mac
47 from crud.separate_bilty_challan import separate_bilty_challan
48 from crud.separate_booking_challan import separate_booking_challan
49 from crud.company_mail import company_mail
50 from crud.pending_mr_register import pending_mr_register
51 from crud.pending_mr import pending_mr
52 from crud.pending_mr_type import pending_mr_type
53 from crud.last_branch import last_branch
54 from crud.token_verification import token_verification
55 from crud.mail_master import mail_master
56 from crud.user_dashbboard import user_dashboard
57 from crud.hisab_statement import hisab_statement
58 from crud.fleet_master import fleet_master
59 from crud.stock_register import stock_register
60 from crud.loginhistory import loginhistory
61 from crud.api_counter import api_counter
62 from crud.marfatiya import marfatiya
63 from crud.bilty_marfatiya import bilty_marfatiya
```

Fig 6.4.7 Addition of table in the init of crud

```

api.py x
32 api_router.include_router(pod_chalan.router, prefix="/pod", tags=["Pod Chalan"])
33 api_router.include_router(crossing_outward.router, prefix="/crossing_outward", tags=["Crossing Outward"])
34 api_router.include_router(bilty_paid_statement.router, prefix="/paid_statement", tags=["Bilty Paid statement"])
35 api_router.include_router(mr_statement.router, prefix="/mr_statement", tags=["MR Statement"])
36 api_router.include_router(tbb_billing_statement.router, prefix="/tbb_billing_statement", tags=["Tbb Billing Statement"])
37 api_router.include_router(crossing_bill_inward.router, prefix="/crossing_bill_in", tags=["Crossing Bill Inward"])
38 api_router.include_router(crossing_bill_outward.router, prefix="/crossing_bill_out", tags=["Crossing Bill Outward"])
39 api_router.include_router(pod_statement.router, prefix="/pod_statement", tags=["POD Statement"])
40 api_router.include_router(group_master.router, prefix="/group", tags=["Group Master"])
41 api_router.include_router(subgroup_master.router, prefix="/subgroup", tags=["Sub Group Master"])
42 api_router.include_router(account_transaction.router, prefix="/account_trans", tags=["Account Transaction"])
43 api_router.include_router(narration_master.router, prefix="/narration", tags=["Narration"])
44 api_router.include_router(stock_change.router, prefix="/stock", tags=["Stock"])
45 api_router.include_router(vehicle_register.router, prefix="/vehicleregister", tags=["VehicleRegister"])
46 api_router.include_router(authenticate_mac.router, prefix="/authmac", tags=["Mac Authentication"])
47 api_router.include_router(mail.router, prefix="/mail", tags=["Send Mail"])
48 api_router.include_router(separate_ewb.router, prefix="/separate-ewb", tags=["Separate EWB"])
49 api_router.include_router(pending_mr_register.router, prefix="/pending_mr_payment", tags=["Pending Mr Payment"])
50 api_router.include_router(last_branch.router, prefix="/last_branch", tags=["Last Branch"])
51 api_router.include_router(token_verification.router, prefix="/token_verification", tags=["Token Verification"])
52 api_router.include_router(mail_master.router, prefix="/mail_master", tags=["Mail Master"])
53 api_router.include_router(user_dashboard.router, prefix="/user_dashboard", tags=["User Dashboard"])
54 api_router.include_router(fleet_master.router, prefix="/fleet_master", tags=["Fleet Master"])
55 api_router.include_router(stock_register.router, prefix="/stock_register", tags=["Stock Register"])
56 # api_router.include_router(backup.router, prefix="/backup", tags=["Backup"])
57 api_router.include_router(marfatiya.router, prefix="/marfatiya", tags=["Marfatiya"])

```

Fig 6.4.8 Addition of table in the init of API Endpoint

```

75
76 @router.put("/update_marfatiya") 1 usage (1 dynamic) 1 idharmik19
77 def update_marfatiya(
78     *,
79     db: Session = Depends(deps.get_db),
80     data: schemas.MarfatiyaUpdate,
81     mf_id: int
82 ) -> Any:
83     """
84     Update Marfatiya
85     """
86     try:
87         resp = crud.marfatiya.update_marfatiya(db=db, obj_in=data, mf_id=mf_id)
88         db.commit()
89         return {"status": True, "data": resp}
90     except Exception as e:
91         raise HTTPException(status_code=404, detail=str(e))
92
93
94

```

Fig 6.4.9 API Endpoint creation of update_marfatiya

```

83
84 def update_marfatiya(self, db: Session, obj_in: MarfatiyaUpdate, mf_id): 1 usage (1 dynamic) tdharmik19
85     try:
86         marfatiya_obj = db.query(Marfatiya).filter(*criteria: Marfatiya.mf_id == mf_id , Marfatiya.is_deleted == 0).first()
87
88         if not marfatiya_obj:
89             raise ValueError(f"mf_no {obj_in.mf_no} not found.")
90         for key, value in obj_in.dict().items():
91             setattr(marfatiya_obj, key, value)
92         db.flush()
93         bilty_data = [{"bilty_id": b.bilty_id, "packages": b.packages} for b in obj_in.bilty_ids]
94         bilty_list = crud.bilty_marfatiya.delete_and_update_bilty_marfatiya(db, mf_id=mf_id, obj_in=bilty_data)
95
96         db.commit()
97         db.refresh(marfatiya_obj)
98         response_data = jsonable_encoder(marfatiya_obj)
99         response_data["bilty_ids"] = bilty_list
100
101         return response_data
102
103     except Exception as e:
104         db.rollback()
105         raise e

```

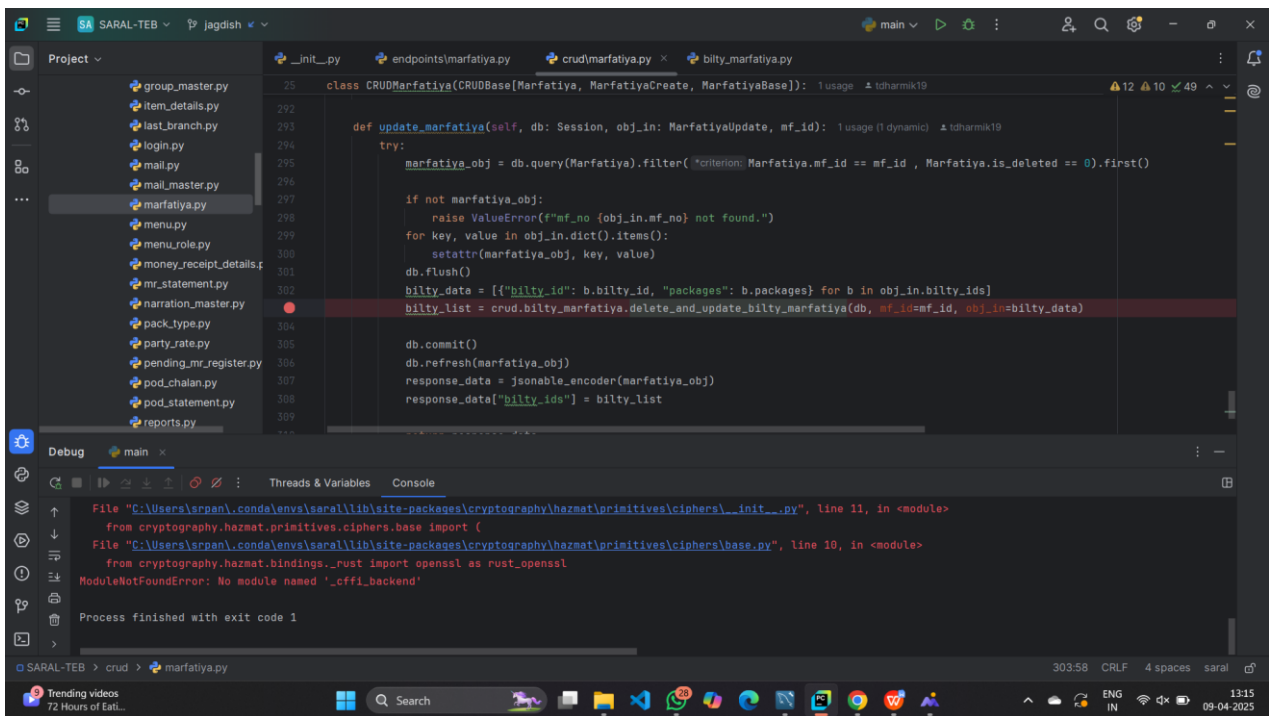
Fig 6.4.10 CRUD creation of update_marfatiya for the API

```

5 def delete_and_update_bilty_marfatiya(self, db: Session, mf_id: int, obj_in: List[Dict] = None): 1 usage (1 dynamic) tdharmik19
6     try:
7         obj_list = []
8
9         # Fetch and delete existing records
10        bilty_marfatiya_ids = self.get_bilty_marfatiya_by_mf_id(db=db, mf_id=mf_id)
11        self.remove_multi(db=db, ids=bilty_marfatiya_ids)
12
13        if not obj_in:
14            return obj_list
15
16        # Insert new records
17        for obj in obj_in:
18            obj_in_data = obj.copy()
19            obj_in_data["mf_id"] = mf_id # Assigning mf_id to new records
20            db_obj = self.model(**obj_in_data) # type: ignore
21            obj_list.append(db_obj)
22
23        db.add_all(obj_list)
24        db.flush()
25
26        for ob in obj_list:
27            db.refresh(ob)
28
29        return obj_list
30
31    except Exception as e:

```

Fig 6.4.11 CRUD creation of delete_and_update_bilty_marfatiya for the API



6.4.12 Debugging the errors

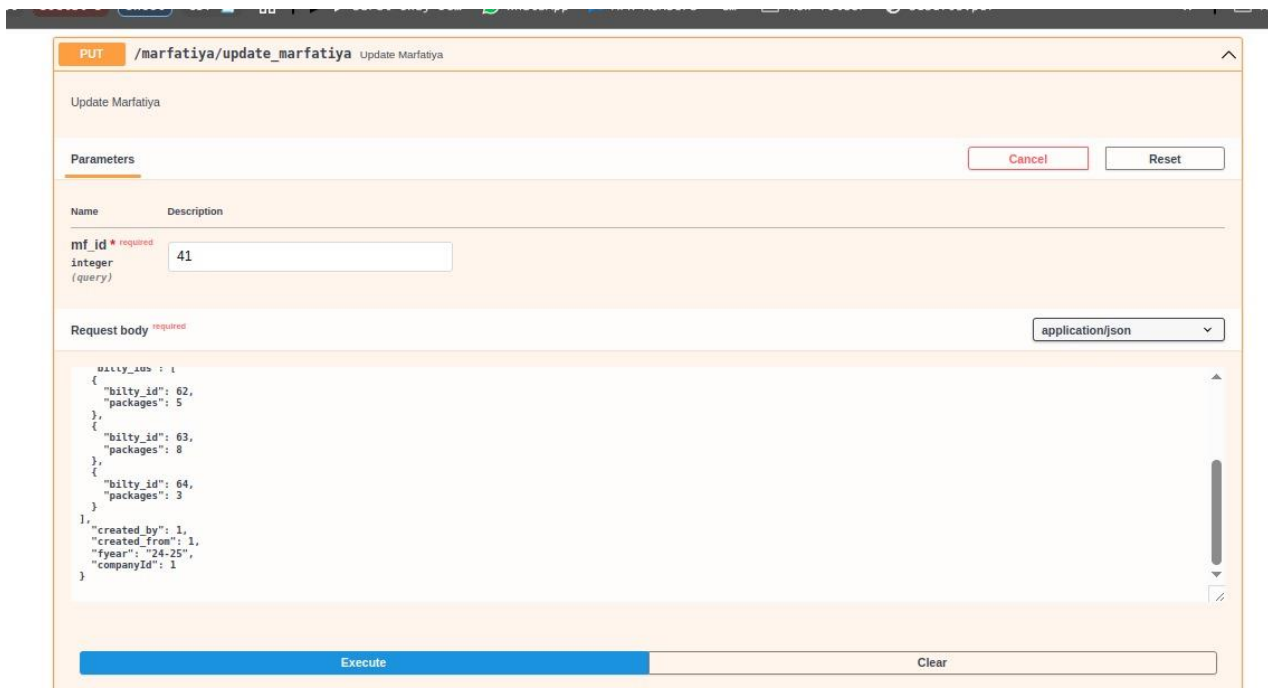


Fig 6.4.12 API in Swagger UI



Fig 6.4.13 Get the perfect data with the payload from the database

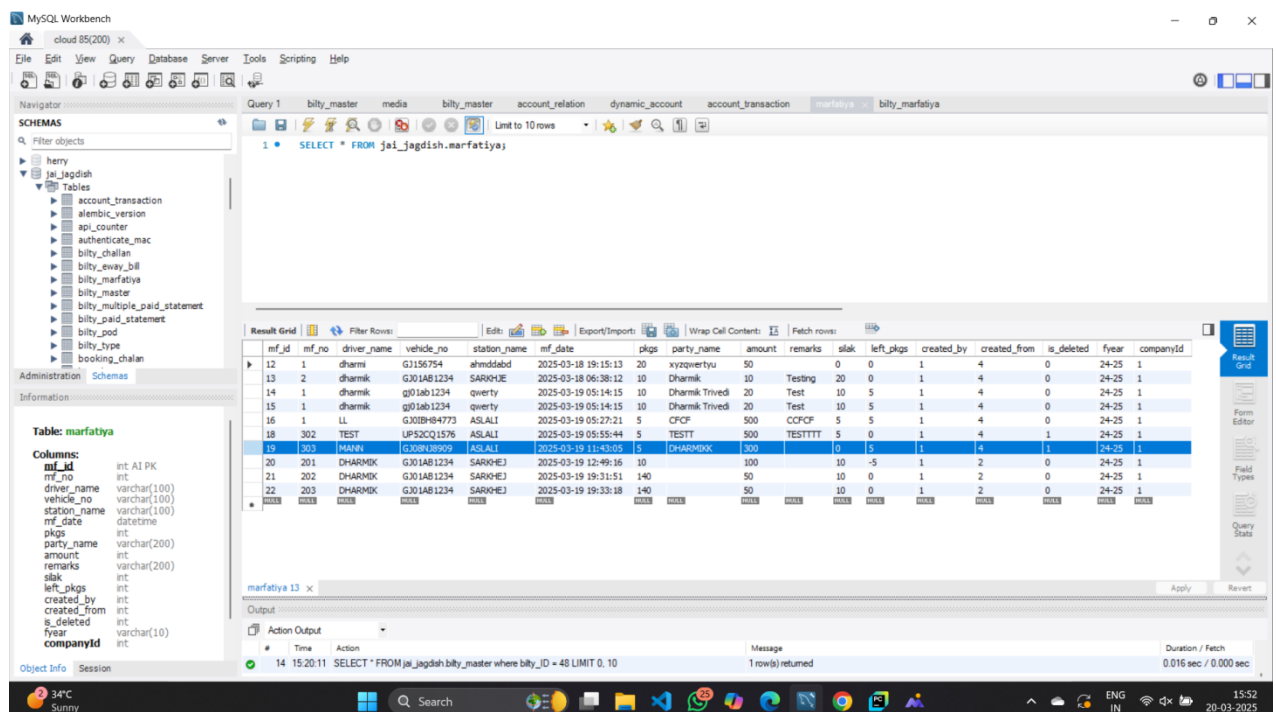


Fig 6.4.14 Changes done after the code run in marfatya table of database

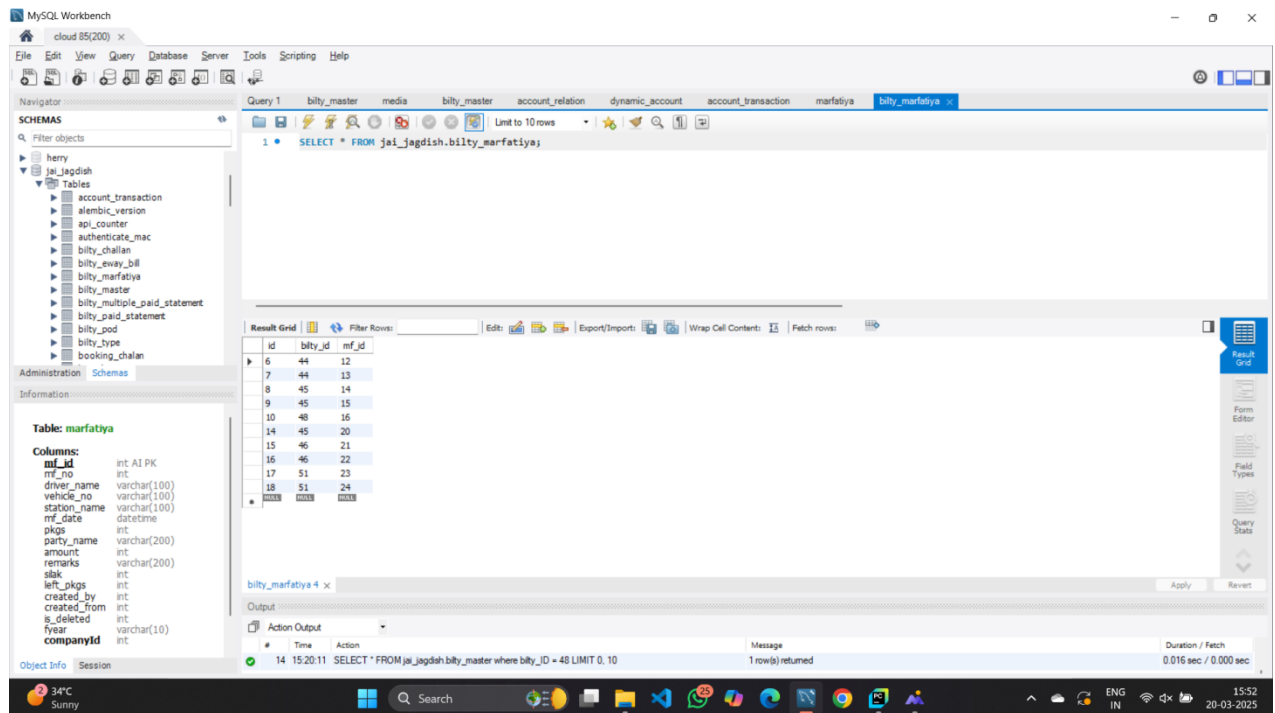


Fig 6.4.15 Changes in table bilty_marfatiya of database

```

Untracked files:
  (use "git add <file>..." to include in what will be committed)
      conda

(myenv) C:\Users\srpan\PycharmProjects\SARAL-TEB>git commit -m "added filter in ge_media_by_image_id is_deleted"
[RCC_run 5813ac5e] added filter in ge_media_by_image_id is_deleted
1 file changed, 1 insertion(+), 1 deletion(-)

(myenv) C:\Users\srpan\PycharmProjects\SARAL-TEB>git push
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 12 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 409 bytes | 136.00 KiB/s, done.
Total 4 (delta 3), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (3/3), completed with 3 local objects.
To a17b8101..5813ac5e RCC_run -> RCC_run

```

Fig 6.4.16 Final Push of the changes


```
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   crud/Media.py

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        conda

no changes added to commit (use "git add" and/or "git commit -a")

(myenv) C:\Users\srpan\PycharmProjects\SARAL-TEB>git add crud/Media.py

(myenv) C:\Users\srpan\PycharmProjects\SARAL-TEB>git status
On branch RCC_run
Your branch is up to date with 'origin/RCC_run'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   crud/Media.py
```

Fig 6.4.17 Final Push is done

CHAPTER 7: TESTING

This chapter details the testing process of the Transportation Management System (TMS) and outlines the results obtained during testing.

7.1 TESTING TECHNIQUES:

◆ **Black Box Testing:**

- Focuses on testing the TMS functionalities without examining the internal code.
- Ensures that shipment creation, route optimization, billing, and POD uploads perform as expected on input-output results.

◆ **White Box Testing:**

- Tests the internal logic, database queries, and code structures.
- Ensures optimal performance in route calculations, data validation, and record handling.

7.2 TEST AUTOMATION:

- ◆ **Automated Testing Tools:** Automating repetitive tasks like shipment creation, tracking updates, and report generation to improve efficiency.
- ◆ **Automation Frameworks:** Tools such as **Pytest**, **Selenium**, or **Postman** can be employed to automate API endpoints and UI functionality testing.

7.3 TEST ENVIRONMENT:

◆ **Environment Setup:**

- Establishing a test environment that closely mirrors the production setup.
- Ensuring database configurations, shipment data flow, and server environments are accurately replicated.

◆ **Dependencies Configuration:**

- Setting up FastAPI, MySQL databases, and third-party integrations like GPS services for location tracking.

7.4 DEFECT MANAGEMENT:

◆ Defect logging:

- Identifying and documenting issues like incorrect shipment status updates, failed POD uploads, or inaccurate billing.
- Using tools like JIRA, **Trello**, or **Bugzilla** to track defects efficiently.

◆ Defect Resolution Process:

- Assigning defects to developers with clear descriptions, expected outcomes and steps to reproduce for efficient resolution.

7.5 PERFORMANCE TESTING:

- ◆ **Load Testing:** Simulating high-volume shipment entries, multiple user logins, and bulk report generations to assess system behaviour under load.
- ◆ **Stress Testing:** Testing the system's performance by pushing it beyond its expected limits to identify breaking points.
- ◆ **Scalability Testing:** Ensuring the system can efficiently scale to handle increasing shipments, drivers, and customer requests.

7.6 SECURITY TESTING:

- ◆ **Penetration Testing:** Simulating cyber-attacks to identify potential security vulnerabilities in shipment data, user roles, and payment gateways.
- ◆ **Vulnerability Scanning:** Scanning the codebase, APIs, and endpoints to ensure data handling and compliance with security standards.

7.7 USER ACCEPTANCE TESTING(UAT):

- ◆ **End-User Testing:** Conducting controlled environment tests with logistics managers, dispatchers, and drivers to validate system functionality.
- ◆ **Feedback Integration:** Gathering feedback to confirm shipment tracking, POD uploads, and billings processes meet user expectations before deployment.

CHAPTER 8

CONCLUSIONS AND DISCUSSIONS

8.1 OVERALL ANALYSIS OF INTERNSHIP

- The development and implementation of the Transportation Management System (TMS) present a highly viable and practical solution in the logistics and supply chain domain. From the technical, economic, and operational perspectives, the project demonstrates strong feasibility and aligns well with the goals of optimizing transport workflows, enhancing data visibility, and enabling smoother coordination among stakeholders like transport companies, clients, and operators.
- From a **technical viability** standpoint, the system leverages modern and scalable technologies such as FastAPI for backend development, MySQL for structured data storage, and React.js for a responsive and user-friendly frontend. These technologies are not only robust and well-supported but also ideal for building modular and efficient web-based platforms. Integration capabilities are strong, allowing the TMS to connect with external systems or expand to include features like vehicle tracking, e-invoicing, and POD upload automation.
- The **operational viability** is also high, as the system is designed with real-world transportation workflows in mind. It includes modules for company management, bill generation, vehicle tracking, POD uploads, and report generation. These functionalities mirror the actual needs of logistics operators and transport firms, making the system directly usable and impactful. Moreover, the system supports role-based access, ensuring that operations are controlled and secure.
- Furthermore, the **scalability and maintainability** of the system ensure long-term sustainability. With a modular architecture, updates or additional modules can be implemented without affecting the core system. This makes the TMS a strong candidate for future-proofing transport operations across multiple organizations or branches.

8.2 PHOTOGRAPH AND DATE OF SURPRISE VISIT BY INSTITUTE MENTOR



Visit by institute mentor to the company on 24th March 2025

8.3 PROBLEMS ENCOUNTERED AND POSSIBLE SOLUTIONS

1. Complex Data Relationships

Problem:

During the development of the TMS, managing relationships between various entities such as companies, vehicles, drivers, bilties, and clients was challenging. The normalized database structure required careful design to avoid redundancy and ensure data integrity.

Solution:

A well-structured Entity-Relationship (ER) diagram was developed beforehand to model real-world relationships clearly. Foreign keys and indexing were applied in MySQL to maintain referential integrity and improve query performance. ORM tools like SQLAlchemy or Tortoise-ORM were also considered for simplifying the handling of complex joins and relationships in FastAPI.

2. Handling POD Uploads Efficiently

Problem:

Managing POD (Proof of Delivery) uploads and associating them accurately with the correct bilty records required robust file handling and validation.

Solution:

POD uploads were implemented using FastAPI's UploadFile with clear naming conventions and file storage structure. Each upload is linked to the corresponding bilty_id in the database. Additional validations were introduced to prevent duplicate uploads and ensure supported file formats only (e.g., PDF, JPEG, PNG).

3. Time-Based Data Filtering(Yesterday,Last week, Any date)

Problem:

Implementing accurate date-based filters for reports and dashboard metrics like “Yesterday,” “Last Week,” or “This Financial Year” introduced logic complexity.

Solution:

Helper utility functions were created to dynamically compute date ranges based on current system time. MySQL's BETWEEN and DATE() functions were used to fetch filtered data efficiently. Edge cases like month transitions and leap years were also handled in the backend.

4. Real-Time Status Updates

Problem:

There was a need for users to receive real-time updates on bilty creation, delivery status, or POD uploads, which posed challenges in a traditional request-response architecture.

Solution:

Periodic polling and WebSocket implementation were considered. Initially, REST APIs with refresh-based updates were used, and the system is structured to adopt WebSocket or third-party services like Pusher or Firebase for future real-time features.

5. Authentication & Role Management

Problem:

Managing multiple user roles (admin, operator, client, etc.) and protecting sensitive endpoints required a secure and scalable approach.

Solution:

JWT (JSON Web Tokens) authentication was integrated into FastAPI along with role-based access control. Middleware checks user permissions on each request, ensuring secure API usage.

6. Deployment & Hosting Issues

Problem:

Initial deployment faced challenges like database connectivity errors, CORS issues in frontend-backend interaction, and inconsistent server configurations.

Solution:

Docker was used to containerize the backend, ensuring consistent environments. Proper .env handling, CORS middleware in FastAPI, and structured logging helped trace and resolve errors quickly during deployment.

7. User Interface Complexity

Problem:

Designing a clean and intuitive interface in React.js for managing multiple modules (e.g., Company Master, Bilty Generation, POD Upload) was time-consuming.

Solution:

Component-based UI architecture was adopted using reusable cards, modals, and tables. Tailwind CSS and component libraries like shadcn/ui ensured fast development.

8.4 SUMMARY OF INTERNSHIP

During the course of the internship, I actively contributed to the design and development of a full-stack Transportation Management System (TMS) aimed at streamlining logistics operations such as vehicle tracking, bilty management, proof of delivery (POD) uploads, and automated reporting. The project was implemented using FastAPI for the backend, MySQL for data storage, and React.js for the frontend interface.

My responsibilities included designing the database schema, developing RESTful APIs, creating user authentication with JWT, and implementing role-based access for different types of users (e.g., admin, client, operator). One of the major highlights of the internship was building custom modules for company master data, bilty creation, POD upload, and reporting dashboards with time filters like "yesterday," "last week," and "this financial year."

I also worked on integrating file upload handling for PODs and ensuring that each file was securely linked to its corresponding bilty entry. Moreover, the system was designed to be scalable, modular, and secure, with thorough validation, error handling, and logging.

Throughout the internship, I improved my technical skills in API development, database design, authentication mechanisms, and modern frontend development. Additionally, I gained valuable experience in real-world problem-solving, teamwork, and software deployment strategies using containerization tools like Docker.

This internship not only deepened my understanding of full-stack development but also prepared me for real-time enterprise application development in the logistics and transportation domain.

8.5 LIMITATIONS AND FUTURE ENHANCEMENT

LIMITATIONS:

1. Dependence on Accurate Data Entry: The TMS heavily relies on accurate and timely data entry by users such as dispatchers, drivers, and warehouse staff. Errors in data input (e.g., incorrect delivery addresses, shipment details, or vehicle information) can lead to delays, mis-routing, or failed deliveries, impacting the overall efficiency of the system. Implementing data validation checks and automated error detection can help minimize this risk, but human error remains a potential challenge.

2. Network and Connectivity Issues: Since TMS systems often rely on real-time data synchronization, any network disruption or connectivity issue can impact the system's ability to update shipment statuses, track vehicles, or communicate with drivers effectively. This limitation can hinder real-time decision-making and lead to operational delays.

3. System Integration Challenges: Integrating the TMS with third-party platforms such as GPS tracking systems, financial software, or customer relationship management (CRM) tools may pose compatibility issues. Poor integration can result in data inconsistencies and limit the system's ability to deliver a seamless workflow.

4. User Adoption and Training: The effectiveness of the TMS depends on the ability of employees to efficiently use the platform. Without proper training, users may struggle with navigating the system, leading to errors, delays, and inefficient use of system features.

FUTURE ENHANCEMENT:**1. AI-Powered Route Optimization:**

- Integrate machine learning algorithms to predict optimal delivery routes based on real-time traffic data, weather conditions, and delivery priorities.
- Dynamic route adjustments can improve efficiency, reduce fuel costs, and enhance delivery timelines.

2. Predictive Maintenance Integration:

- Implement predictive maintenance features using IoT sensors to monitor vehicle health.
- This enhancement can alert fleet managers about potential issues before breakdowns occur, reducing downtime and maintenance costs.

3. Enhanced Real-Time Tracking System:

- Improve GPS tracking capabilities with live updates, geofencing alerts, and estimated time of arrival (ETA) notifications for better shipment visibility.
- Adding a driver's performance dashboard can improve accountability.

4. Automated Document Management:

- Introduce automated scanning and digital storage for critical documents like PODs (Proof of Delivery), invoices, and receipts.
- This can streamline record-keeping and minimize paperwork.

5. Integration with Third-Party Platforms:

- Enable seamless integration with ERPs, CRMs, and warehouse management systems to improve data flow across different platforms.
- This will enhance operational efficiency and improve inventory management.

6. Advanced Analytics and Reporting:

- Develop comprehensive data visualization dashboards to analyze shipment trends, fleet performance, and cost analysis.
- Predictive analytics can help identify inefficiencies and improve decision-making.

CONCLUSION

The Transportation Management System (TMS) developed during this project has proven to be a robust and efficient solution tailored to modern logistics needs. By leveraging powerful technologies such as **FastAPI**, **MySQL**, and **React.js**, the system successfully automates and streamlines key transportation processes, including bilty management, POD uploads, vehicle tracking, and report generation.

The modular architecture ensures that the platform is scalable and can easily be adapted for future enhancements or business-specific customizations. Features like secure user authentication, real-time data processing, and flexible time-based reporting empower logistics companies to manage their operations with greater accuracy, speed, and transparency.

Overall, the TMS not only reduces manual workload and human errors but also increases overall operational efficiency, making it a valuable asset for transportation and logistics firms looking to embrace digital transformation.

REFERENCES

- MySQL Tutorial : <https://www.w3schools.com/mysql/default.asp>
- Python Tutorial : <https://www.w3schools.com/python/default.asp>
- GitHub : <https://github.com/>
- FastAPI : <https://fastapi.tiangolo.com/>
- FastAPI CRUD Operations : <https://www.geeksforgeeks.org/fastapi-crud-operations/>