CHINMAYA INSTITUTE OF TECHNOLOGY KANNUR

SYNC CITY

PRESENTED BY

Mr. ABHILASH P

Reg No : **C3GMCA2001**

THIRD SEMESTER MCA (2023-2025)

SCHOOL OF COMPUTER SCIENCE AND

INFORMATION TECHNOLOGY

DECLARATION

I **ABHILASH P**, 3rd semester MCA student of Chinmaya Institute of Technology under Kannur University do here by declare that the project entitled **“SYNC CITY”** is the original work carried out by me under the supervision of **ASST PROF. SHAJEER M P**, Chinmaya Institute of Technology towards partial fulfillment of the requirement of M.C.A Degree, and no part there of which has been presented for the award of any other degree.

Signature of the Student

Place : Kannur

Date :

CHINMAYA INSTITUTE OF TECHNOLOGY

KANNUR

SCHOOL OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY

<u>CERTIFICATE</u>

This is to certify that the main project "**SYNC CITY**" submitted in partial fulfillment of the requirement for the award ofn MCA Degree under kannur university is a result of bonafide work carried out by **Mr. ABHILASH P** University **Register Number: C3GMCA2001** third semester in the year 2023-2025.

External Examiner:

1.

2.

Faculty Incharge:

Place: **Head of the Department:**

Date:

ACKNOWLEDGEMENT

I would like to express my deepest gratitude to all those who have helped and supported me during the development of this project. First and foremost, I extend my sincere thanks to my project guide, Asst. Prof. **Shajeer M P**, for his invaluable guidance, encouragement, and constructive feedback throughout the project. His expertise and suggestions have played a vital role in shaping the project and ensuring its success. I would also like to thank the MCA department for providing me with the necessary resources and environment to complete this project successfully. The facilities and infrastructure provided were essential in carrying out the research and development work. Special thanks to my friends and colleagues for their support, ideas, and continuous motivation throughout the duration of this project. Their encouragement and feedback have been greatly appreciated. This project would not have been possible without the support of all these individuals, and I am deeply grateful for their assistance

ABHILASH P

ABSTRACT

The Sync City project is a comprehensive smart city platform designed to integrate multiple urban services into a seamless digital ecosystem. It aims to enhance the daily lives of residents and visitors by providing convenient access to essential services such as healthcare, transportation, marketplace, and event management. The platform allows users to book doctor appointments through a token-based system, reducing waiting times and streamlining healthcare access. It features a tradesman marketplace where skilled workers can showcase their services, making it easy for users to find and hire them. Sync City also includes a goods marketplace connecting farmers directly with buyers, promoting sustainable trade practices. Businesses can advertise their deals on a city-wide offers page, and users can book tickets for city events with ease. The platform supports reliable taxi and driver booking services, ensuring accessible transportation for all. Built using Python Django with an SQL database, Sync City leverages modern web technologies to provide a robust and user-friendly experience. By integrating these diverse functionalities, the platform fosters efficiency, accessibility, and growth in urban environments. This innovative solution embodies the vision of a connected, smart city that caters to the evolving needs of its inhabitants.

TABLE OF CONTENTS

LIST OF DIAGRAMS

1. INTRODUCTION

Sync City is a smart city platform designed to integrate multiple essential urban services into one comprehensive system, enhancing convenience and connectivity for residents and visitors. The platform offers a variety of functionalities aimed at streamlining daily tasks and promoting community engagement. Key functionalities include a **Hospital Management System** for token-based doctor appointment booking, a **Tradesman Marketplace** where tradespeople can advertise and offer their services, and a **City-Wide Offers Page** that enables businesses to promote discounts and deals effectively. Additionally, Sync City facilitates **Event Ticket Booking** for easy access to city events, **Taxi and Driver Booking** for efficient transportation, and a **Goods Marketplace** that connects farmers and buyers, supporting local trade. Developed using Python Django and powered by an SQL database, Sync City seamlessly integrates these features to create a smarter and more efficient urban environment for everyone.

2. BACKGROUND STUDY

2.1 EXISTING SYSTEM AND PROPOSED SYSTEM

2.1.1 EXISTING SYSTEM

Multiple services such as hospital management, local offers, event information, tradesman, taxi, and goods booking were managed through separate, disconnected platforms. Currently, users have to switch between different websites or apps to find the services they need. This can be confusing and time-consuming.

2.1.2. PROPOSED SYSTEM

The proposed system for the Sync City is designed to create a unified platform that integrates multiple city services to enhance efficiency, convenience, and accessibility for residents and visitors. The system includes a hospital management module that allows users to book doctor appointments using a token-based system, reducing waiting times and improving the overall healthcare experience. It also features a tradesman marketplace where service providers can advertise their expertise, enabling users to easily find and hire professionals for various tasks. the platform includes a city-wide offers page that allows businesses to promote their deals and discounts, fostering local commerce and benefiting consumers. Sync City further simplifies urban life with an event ticket booking system, making it easier for users to attend city events. The taxi and driver booking feature ensures reliable and accessible transportation services, while the goods marketplace connects farmers directly with buyers, promoting sustainable trade practices.

3. PROBLEM FORMULATION

3.1 MAIN OBJECTIVE

The Sync City project aims to create a unified platform to integrate and manage essential services across a city efficiently. This smart city platform is designed to streamline various operations, providing a seamless experience for users by addressing healthcare, commerce, transportation, and event management needs. Sync City encompasses key features such as token-based doctor appointment booking, tradesman services, city-wide offers, event ticketing, taxi and driver bookings, and a goods marketplace to enhance convenience and accessibility for residents and visitors.

The specific objectives of the project include:

- Developing a user-friendly login page that allows users to sign up and access personalized features.
- Creating an admin panel for managing and overseeing platform operations effectively.
- Implementing a shopping cart functionality for goods and services.
- Integrating an online payment system to facilitate secure and convenient transactions.

3.2 METHODOLOGY

Agile methodology is an ideal choice for the Sync City project as it offers a flexible and iterative approach to development, perfectly suited for the diverse and complex features of the platform. By breaking down the project into smaller, manageable tasks, Agile enables the development and testing of core functionalities like token-based doctor appointment booking, tradesman services, and event ticketing in incremental stages. This approach allows for continuous improvement and the early delivery of essential features, ensuring they can be refined based on real-time feedback. The user-centric nature of Agile emphasizes collaboration and responsiveness, making it easier to incorporate user feedback to enhance the platform's usability and personalization. For example, features like the shopping cart and goods marketplace can be tested in sprints to ensure they meet user needs effectively. Regular sprint reviews and testing help maintain the quality of the platform by identifying and addressing potential issues promptly, mitigating risks early in the development cycle.

Furthermore, Agile fosters teamwork and efficient resource utilization by focusing on prioritized tasks and ensuring the development team remains aligned with the project goals. This dynamic and adaptable approach ensures that Sync City evolves continuously, delivering a robust, high-quality platform that meets the diverse needs of city residents and visitors.

Flexibility: Agile allows you to adapt to changes during development, which is ideal for web-based projects where new requirements or improvements may arise.
Incremental Development: You can build your project in small, manageable iterations, such as user authentication, antique listing, admin verification, and transaction handling.

User-Centric Approach: Agile emphasizes regular feedback and collaboration, even if formal feedback features are not included in the project.

Frequent Testing: Testing is integrated into each iteration, ensuring that issues are identified and resolved early.

Team Collaboration: If you're working in a team, Agile promotes clear communication and efficient task management.

Software Requirement Analysis

This technique is additionally recognized as a feasibility study. The want for this software was investigated. A Software Requirement Specification file was once created at the give-up step. It holds all data required to develop the venture of this which consists of functional and non-functional requirements. The requirement gathering manner is intensified and targeted specifically on software. The facts are for the software, as nicely as required function, behavior, performance, and interfacing are nicely understood.

System Analysis and Design

In this phase, of the software development process, the software's usual shape is defined. The logical gadget of the product is developed in this phase. A dataflow graph that shows the drift of statistics thr the machine is created. This phase produces a graph and a low-degree design. The high level of excessive level diagram comprises the modules to be generated and their purpose. The low-level sketch consists of an extra elaborated view of the modules which consists of the methods to be defined.

Code Generation

The sketch created in the diagram segment ought to be translated into a machine-readable form. The code generation step performs this task. Since this project processing entails connection to the dataset.

Testing

After the code used to be generated, the software program trying out begins. Errors in the program drift used to be identified in this step.

3.3 PLATFORM

Platform types form the foundation for the architecture, design, and implementation of the Sync City platform. The tools and technologies used in the development of this software application are as follows:

- **Operating System** – Windows
- **Technology** – Python Django

3.3.1 Python

Python Django is a high-level Python web framework that simplifies the development of robust and scalable web applications. It is designed to facilitate the rapid development of secure and maintainable websites by providing built-in tools and conventions. The key components of the Python Django stack include:

- **Django –** A powerful web framework that handles application logic, routing, and ORM (Object-Relational Mapping) for database operations.
- **Python –** A versatile programming language used for writing the backend logic of the application.
- **SQL Database –** The database layer used to store structured data efficiently, enabling secure and scalable data management.

The Django framework follows the Model-View-Template (MVT) architecture, where:

- **Model** represents the database structure and handles data management.
- **View** processes user requests and sends responses.
- **Template** defines the presentation layer for displaying the front-end interface.

Python Django is an ideal platform for developing the Sync City project, as it ensures seamless integration between components, supports rapid development, and offers robust security features. The framework’s scalability and flexibility make it well-suited for handling the diverse functionalities of the platform, including token-based hospital bookings, tradesman service listings, and city-wide offers management. This comprehensive stack provides a cohesive foundation for delivering a high-quality, user-focused solution.

3.3.2 ARCHITECTURE

1. Model Layer

The **Model Layer** in Django is the data-handling component of the architecture, representing the database schema and the relationships between data entities. This layer uses Django’s built-in ORM (Object-Relational Mapping) to interact with the database, allowing developers to perform CRUD operations without writing raw SQL queries. For the Sync City project, the Model Layer defines key entities such as Users, Hospitals, Tradesmen, Events, Drivers, Shops, and Farmers. These entities are structured in tables to store and manage information like appointments, service listings, ticket bookings, and goods details efficiently.

2. View Layer

The **View Layer** handles the business logic and acts as the bridge between user requests and the data stored in the Model. It processes HTTP requests, retrieves or updates data via the Model, and sends the processed data to the Template for presentation. In Sync City, the View Layer manages functionalities such as processing doctor appointment requests, listing tradesman services, and generating personalized user experiences. It ensures that user actions are effectively translated into meaningful outcomes.

3. Template Layer

The **Template Layer** is the presentation component responsible for rendering data into a user-friendly interface. It dynamically displays the content processed by the View Layer using Django’s templating engine. In Sync City, the Template Layer ensures an intuitive and interactive frontend where users can perform actions such as browsing city-wide offers, booking tickets, or accessing their dashboard. This layer is designed to provide a seamless and visually appealing user experience.

4. Controller Layer (Middleware)

The **Controller Layer** (referred to as Middleware in Django) acts as the intermediary between the user’s request and the application. It processes HTTP requests and responses, manages authentication and authorization, and handles security features like preventing cross-site request forgery (CSRF). For Sync City, the Controller Layer ensures secure user interactions and data exchange, such as verifying login credentials, ensuring role-based access for Admins, and maintaining the integrity of sensitive operations like payment processing.

By utilizing this architecture, the Sync City platform is designed to deliver an efficient, scalable, and secure application while maintaining a clear separation of concerns to simplify development and maintenance.

4. SYSTEM ANALYSIS AND DESIGN

4.1 FEASIBILITY ANALYSIS

A feasibility analysis is conducted to identify the software solution that achieves optimal performance at minimal cost. The primary objective of this analysis is to determine whether it is functionally and technically feasible to develop the proposed Sync City platform. The feasibility study examines the system's operational effectiveness, its impact on stakeholders, the ability to meet user requirements, and efficient utilization of resources. It helps establish the scope of the project and ensures that the system aligns with its intended objectives. The study addresses three critical aspects:

- **Operational feasibility**
- **Technical feasibility**
- **Economic feasibility**

Operational Feasibility

The operational feasibility of Sync City evaluates its scope to ensure smooth operation once implemented. This analysis considers whether the system will meet the needs of the city's residents, businesses, and administrators effectively. Factors include user-friendliness, management support, and ensuring the system does not cause harm or inefficiencies. The platform's design makes it highly accessible, allowing users with basic technical knowledge to navigate and use the system. Features like intuitive interfaces and role-specific functionalities ensure operational success and wide acceptance among stakeholders.

Technical Feasibility

Technical feasibility involves analyzing the technical specifications and infrastructure required for Sync City. The system is built on a robust stack, including Python Django for backend operations, SQL for database management, and modern frontend technologies, ensuring technical reliability. The development environment and deployment infrastructure are adequate to support the project’s requirements. With its well-validated design and emphasis on security, Sync City is technically feasible, meeting industry standards for accuracy, reliability, and ease of access.

Economic Feasibility

Economic feasibility focuses on assessing the cost-effectiveness of the Sync City project. The platform is designed to be scalable and cost-efficient, utilizing open-source technologies and resources effectively. Anticipated savings in time and effort for users, along with the streamlined service delivery, justify the initial investment. Maintenance and operational costs are kept low through automated processes and error reduction mechanisms. Thus, the project is economically viable and can be implemented within the available budget.

Behavioral Feasibility

Behavioral feasibility examines user acceptance and adaptability to the system. Sync City's intuitive user interface and clearly defined roles ensure a positive response from diverse user groups, including residents, businesses, and city administrators. Minimal training is required due to its simple and interactive design. By solving real-world problems such as appointment booking, tradesman discovery, and event ticketing efficiently, Sync City ensures user satisfaction and acceptance across all levels. This makes the project behaviorally feasible and likely to be embraced by its target audience.

4.2 REQUIREMENT ANALYSIS

Requirement analysis is a crucial step in the development of the Sync City platform, aimed at understanding and documenting the needs and expectations of stakeholders. It ensures that the system's design and functionality align with the intended objectives, providing a seamless and efficient solution for city-wide services. The requirements are divided into functional and non-functional categories to ensure a comprehensive understanding of the project’s scope.

1. Functional Requirements

Sync City must support core functionalities such as token-based doctor appointment booking, tradesman service listings, city-wide offers management, event ticket booking, taxi/driver hiring, and a goods marketplace for farmers. Each feature must be user-friendly, efficient, and fully integrated to provide a unified platform experience.

2. Non-Functional Requirements

The platform must ensure high scalability to handle increasing user demands, robust security protocols to protect sensitive data, and optimal performance for smooth operation. It should also support responsive design to ensure accessibility across devices like desktops, tablets, and smartphones.

3. User Requirements

The system must cater to various stakeholders, including users, administrators, tradesmen, drivers, shops, and farmers, providing them with specific roles and access to their respective functionalities. It must also enable intuitive navigation and streamlined processes for actions like booking, registration, and payment.

4. Technical Requirements

The development of Sync City relies on Python Django for backend development, SQL for database management, and modern web technologies for frontend responsiveness. The system must also integrate payment gateways and APIs for features like location-based services and notifications.By systematically analyzing these requirements, the project aims to deliver a platform that meets stakeholder expectations and ensures a smooth, user-centric experience.

4.3. SYSTEM REQUIREMENT SPECIFICATIONS

4.3.1 Sof twareRequirements

Front End: HTML, CSS, JavaScript

- **Backend Framework:** Python Django
- **Database:** SQL
- **Operating System:** Windows
- **Technology:** Django Framework
- **IDE:** PyCharm / Visual Studio Code
- **Packages Used:** Django Rest Framework, Pillow, and Django-Allauth

4.3.2 Hardware Specifications

- **Processor:** Intel Core i5 or Ryzen 5
- **Storage:** SSD, Minimum 512GB
- **Graphics Card:** Integrated or AMD Radeon (optional)
- **RAM (Minimum):** 8GB

4.3.3 Functional Requirements

Core Functionalities of Sync City

1. **Hospital Management System:**

   The platform includes a token-based system for doctor appointment bookings, allowing users to schedule consultations efficiently. This reduces waiting times and ensures a streamlined healthcare experience for residents. Hospitals can manage appointment schedules, view patient bookings, and handle cancellations through the system.

2. **Tradesman Marketplace:**

   Sync City provides a platform for tradesmen to advertise their services, such as plumbing, electrical work, or carpentry. Users can browse available tradesmen, view ratings and reviews, and book services directly through the system. This feature fosters a convenient way for service providers to connect with potential clients.

3. **City-Wide Offers and Event Ticket Booking:**

   Businesses can promote deals and discounts on a dedicated offers page, enhancing local commerce. Additionally, the platform enables users to book tickets for city events, providing a one-stop solution for entertainment planning. Event coordinators can use the system to manage ticket sales and attendee information efficiently.

4. **Taxi and Goods Marketplace:**

   Users can book taxis or drivers for reliable and convenient transportation across the city. The goods marketplace connects farmers directly with buyers, allowing the purchase and sale of fresh produce through the platform. This feature supports sustainable trade practices and benefits both consumers and producers

4.3.4 Non-Functional Requirements

• **Accuracy**: Accuracy in functioning and the nature of simple needs to be maintained by using the system.

• **Speed**: The machine should be successful in offering speed.

• **Low cost**: This gadget is very less expensive to implement and is also user-friendly.

• **Less Time consuming:** It makes use of very less time compared to the existing system.

• **User Friendly**: This proposed machine is relatively consumer pleasant they enable the creation of a correct environment.

4.3.5 Quality Requirements

• **Scalability**: The software will meet all of the user requirements.

• **Maintainability**: The gadget needs to be maintainable. It should preserve backups to atone for device failures and need to log its things to do periodically.

• **Reliability**: The appropriate threshold for downtime have to be large as possible. i.e. mean time between screws-ups has to be massive as possible. And if the machine is broken, the time required to get the device back up once more needs to be minimum.

• **Availability**: This machine is without difficulty reachable as the core requirements in constructing the software is effortlessly obtained.

• **High- Functionality**: This machine is fantastically purposeful in all environments since they are fantastically adaptable.

4.4 ACTOR IDENTIFICATION

There are three types of users in this system. They are:

1. Admin
2. User
3. Worker
4. Hospital
5. Driver
6. Event coordinator
7. shop
8. Farmer

4.4.1 ADMIN

Admin has Full control over the system and can perform the following:

- Approve hospital
- View rating
- View booking

- manage services
- block worker
- verify farmer
- view event coordinator
- view shop and verify

4.4.2 User

- find services
- booking services
- report fake services
- search hospital
- booking doctor
- view hospital
- view nearby vehicle
- view driver details
- view booking vehicle
- view previous booking

4.4.3 Worker

- create profile
- view booking
- view review and rating
- view payment
- manage own services

4.4.4 Hospital

- doctor management
- schedule management
- view booking

4.4.5 Driver

- create profile
- view bookings
- view previous booking

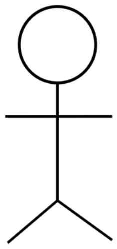
4.5 USECASE DIAGRAM

A approach for identifying, outlining, and organising system requirements is called a use case. The use case consists of a number of potential interactions between users and systems in a specific environment that are connected to a specific objective. All system operations that are significant to users should be included in the use case. An assortment of potential outcomes connected to a specific objective might be thought of as a use case.

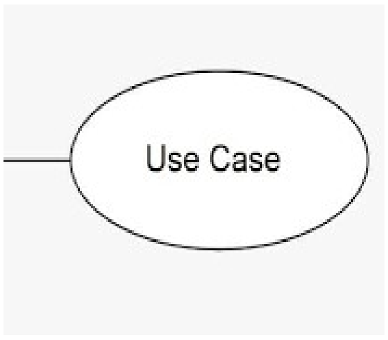
A use case often consists of the following three components:

- Actors: Users who engage in system interaction are actors.
- System: Use cases record the functional requirements that define the system's intended behaviour.
- Use cases, which describe the steps and variations involved in achieving a goal, are often started by a user.

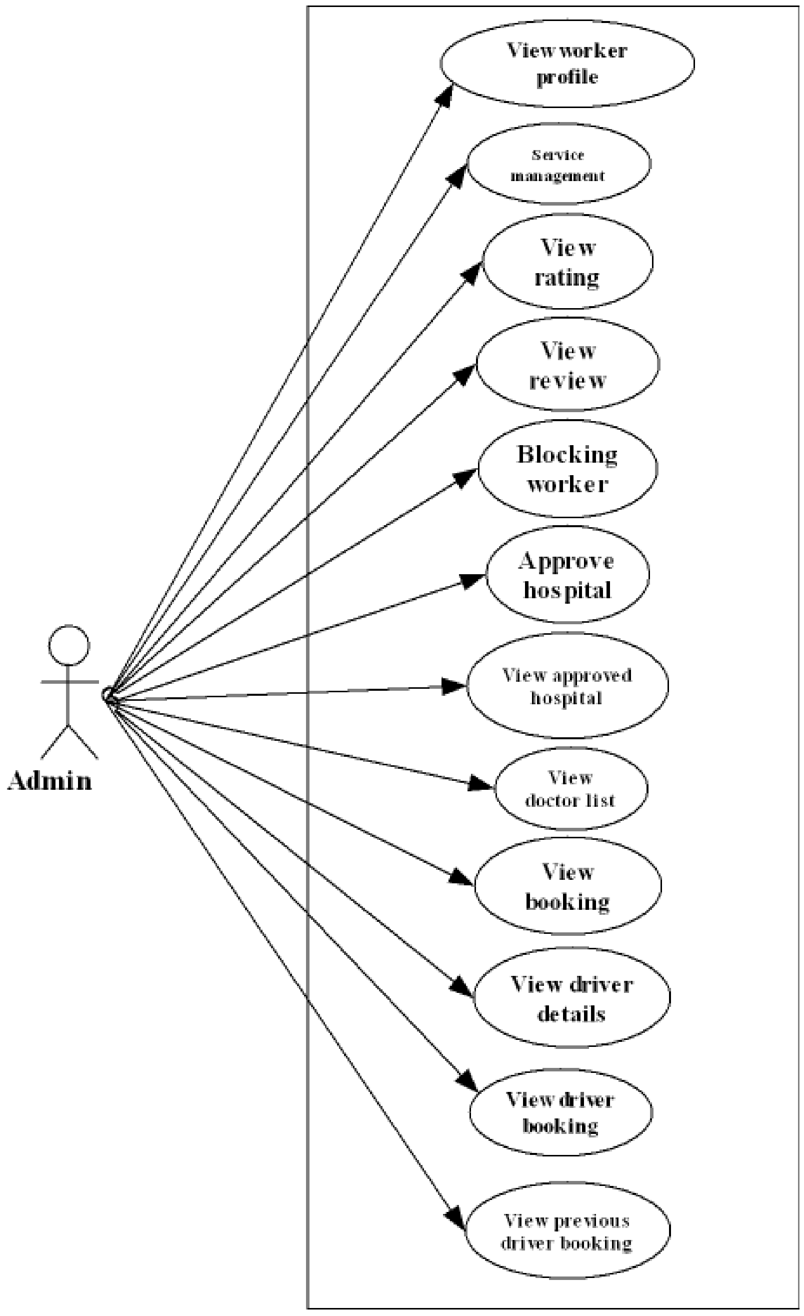
An Actor in the usecase is represnted as :



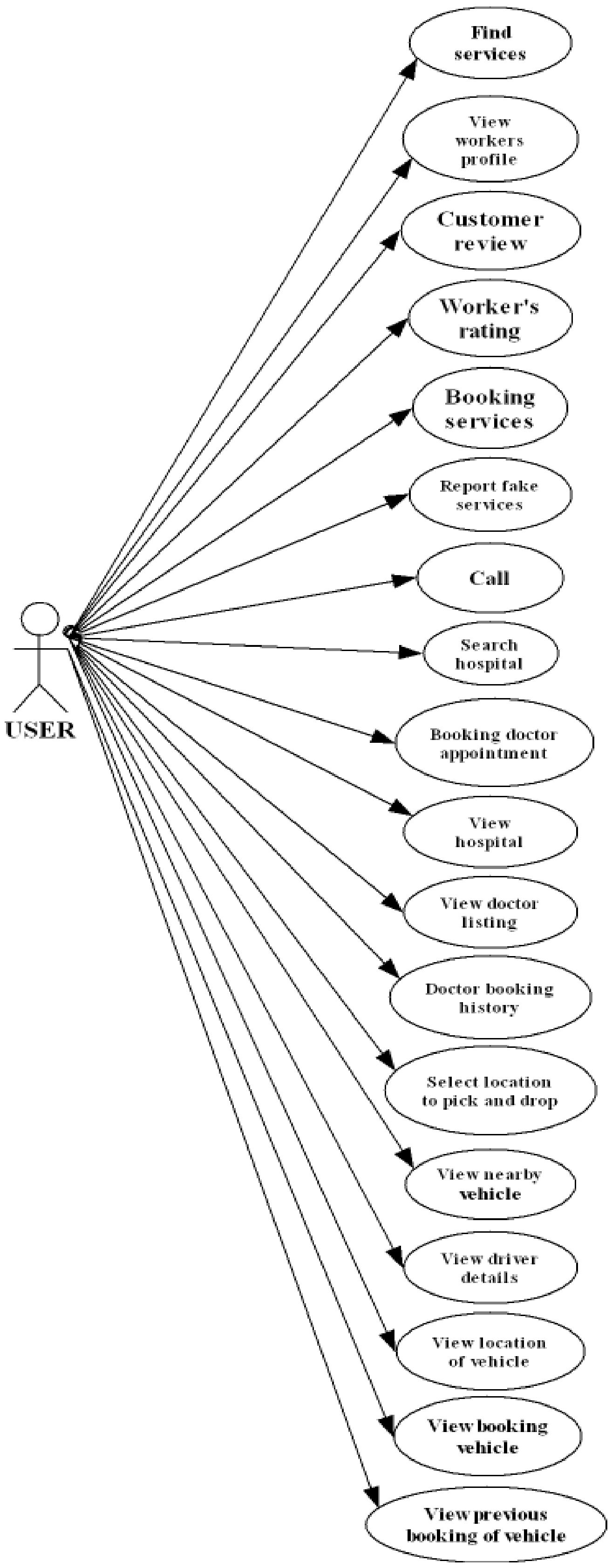
A Usecase is represented as :



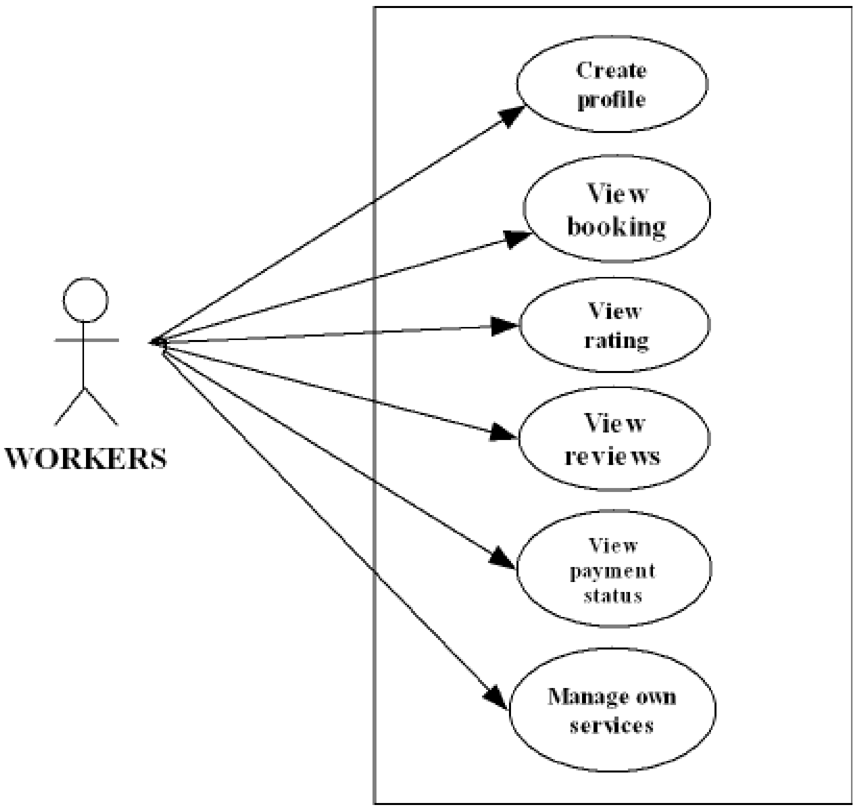
ADMIN:


Admin
View worker profile
Service management
View rating
View review
Blocking worker
Approve hospital
View approved hospital
View doctor list
View booking
View driver details
View driver booking
View previous driver booking

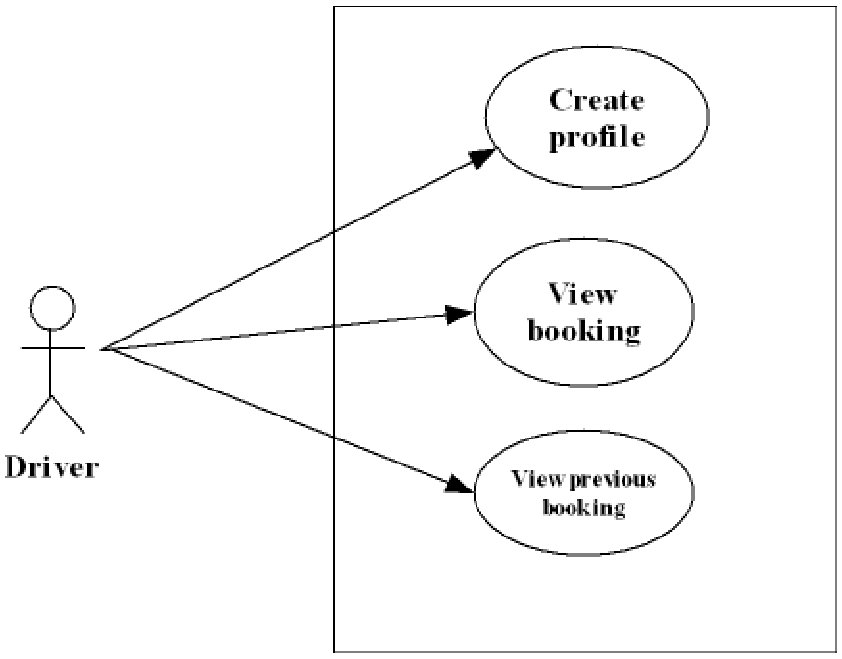
USER:


USER
Find services
View workers profile
Customer review
Worker's rating
Booking services
Report fake services
Call
Search hospital
Booking doctor appointment
View hospital
View doctor listing
Doctor booking history
Select location to pick and drop
View nearby vehicle
View driver details
View location of vehicle
View booking vehicle
View previous booking of vehicle

WORKER:



DRIVER:

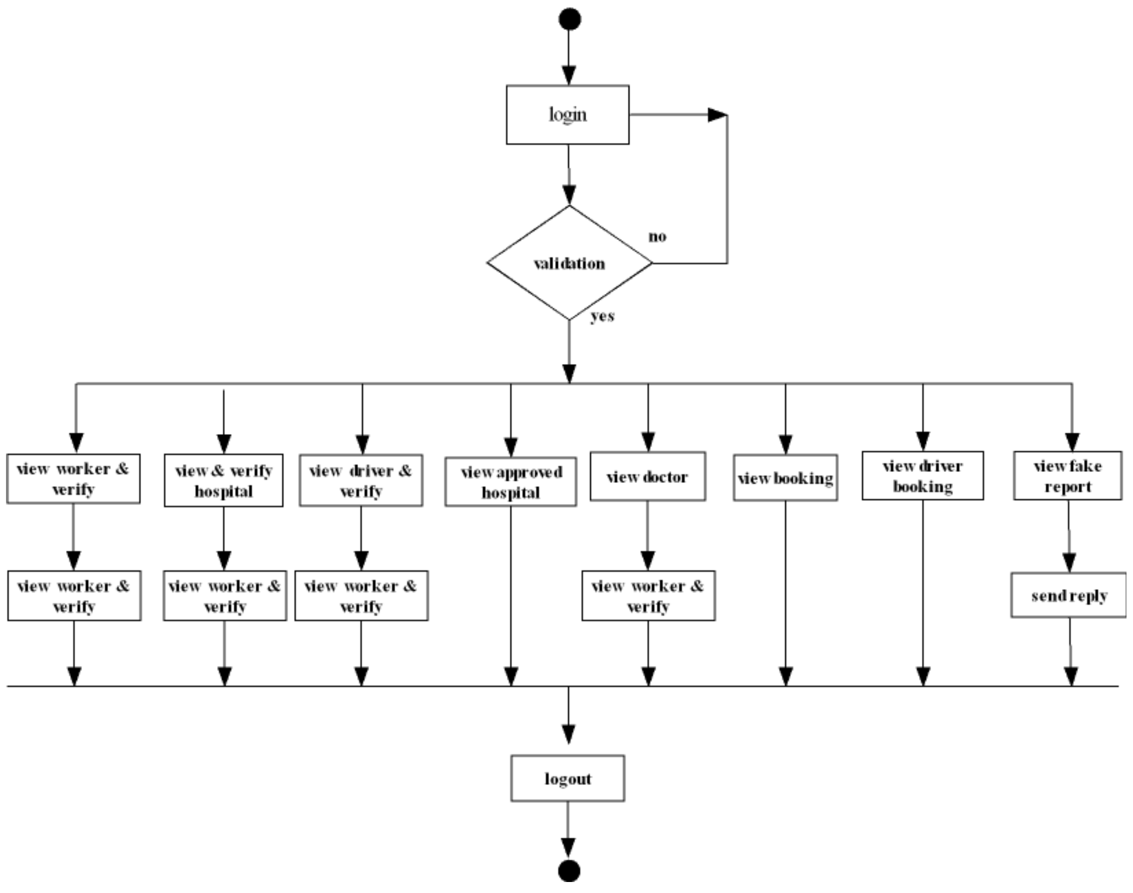


4.6 ACTIVITY DIAGRAM

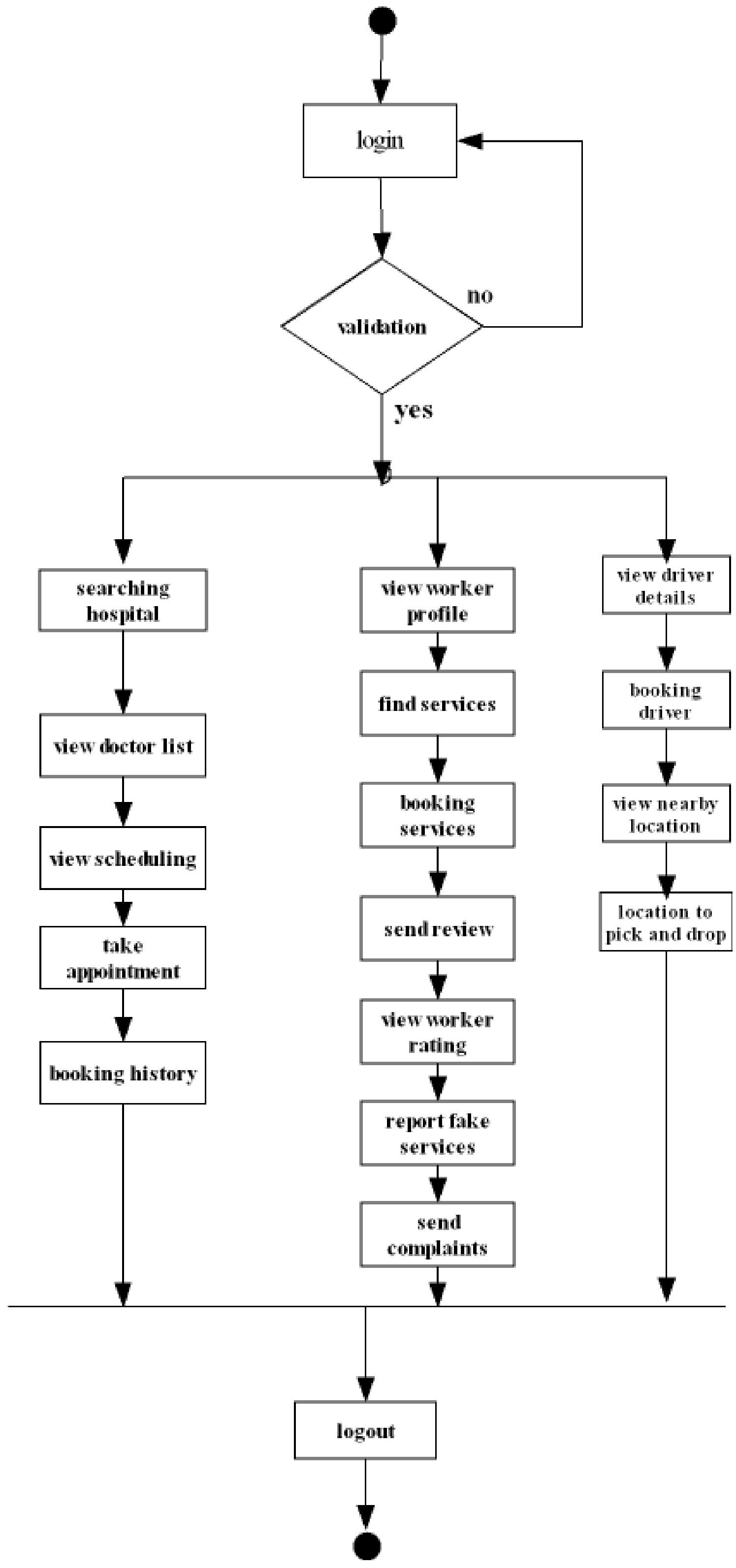
A key UML layout for describing the dynamic system components is an activity diagram. A glide chart is used as the activity design to represent the transition from one pastime to another. The action might be referred to as a system operation. As a result, the manipulator moves from one operation to another. This drift may be contemporaneous, branching, or sequential. Activity diagrams use unique aspects like fork, be a part of, etc. to provide various types of drift control. Activity is a precise system operation. Activity diagrams are now utilised to create the executable system using forward and reverse engineering methodologies, in addition to helping to visualise a device's dynamic nature.

- Draw the undertaking drift of a system.
- Describe the sequence from one endeavor to another.
- Describe the parallel, branched, and concurrent float of the system.

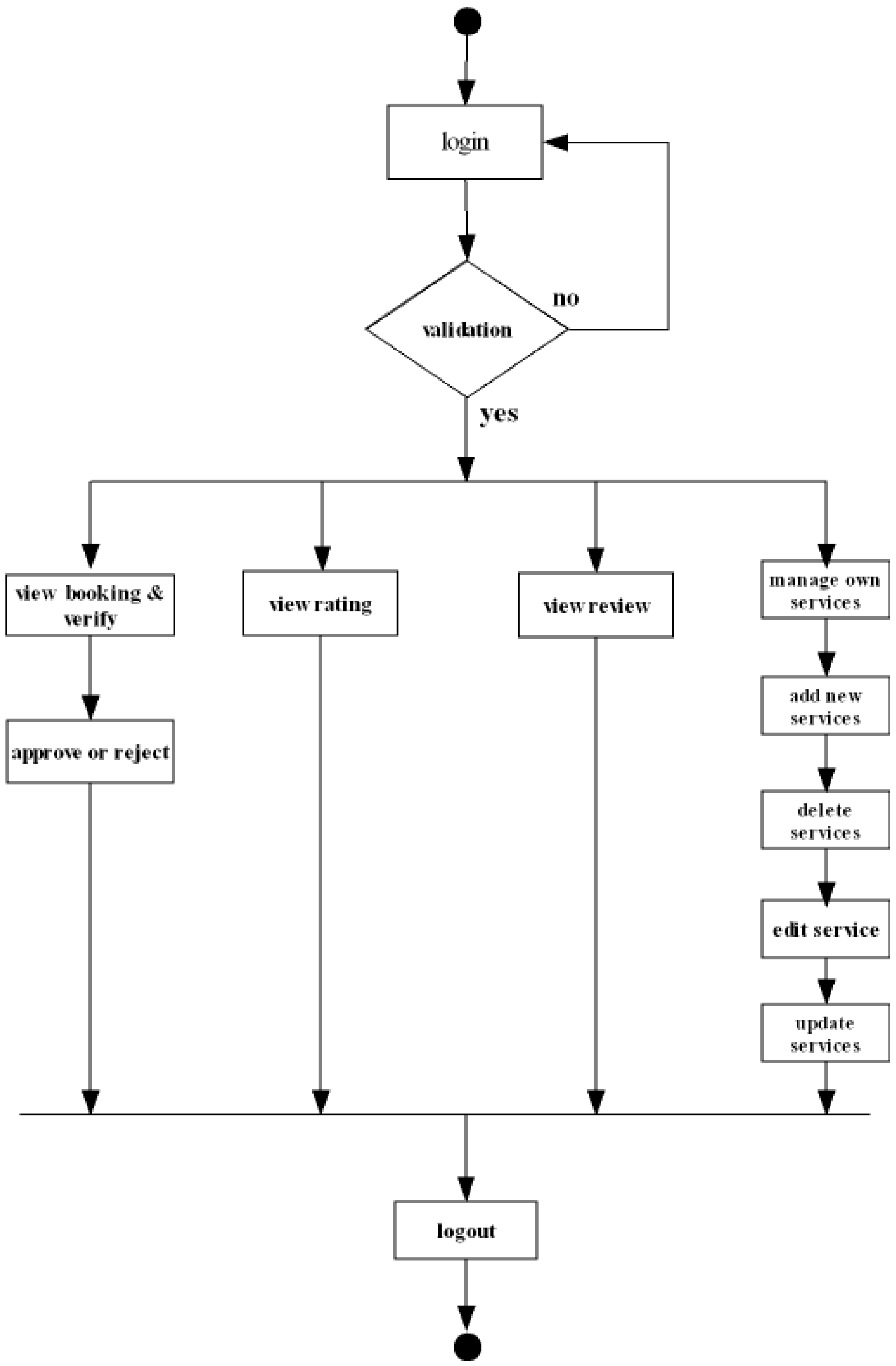
4.6.1 ADMIN


login
validation
no
yes
view worker & verify
view & verify hospital
view driver & verify
view approved hospital
view doctor
view booking
view driver booking
view fake report
view worker & verify
view worker & verify
view worker & verify
view worker & verify
send reply
logout

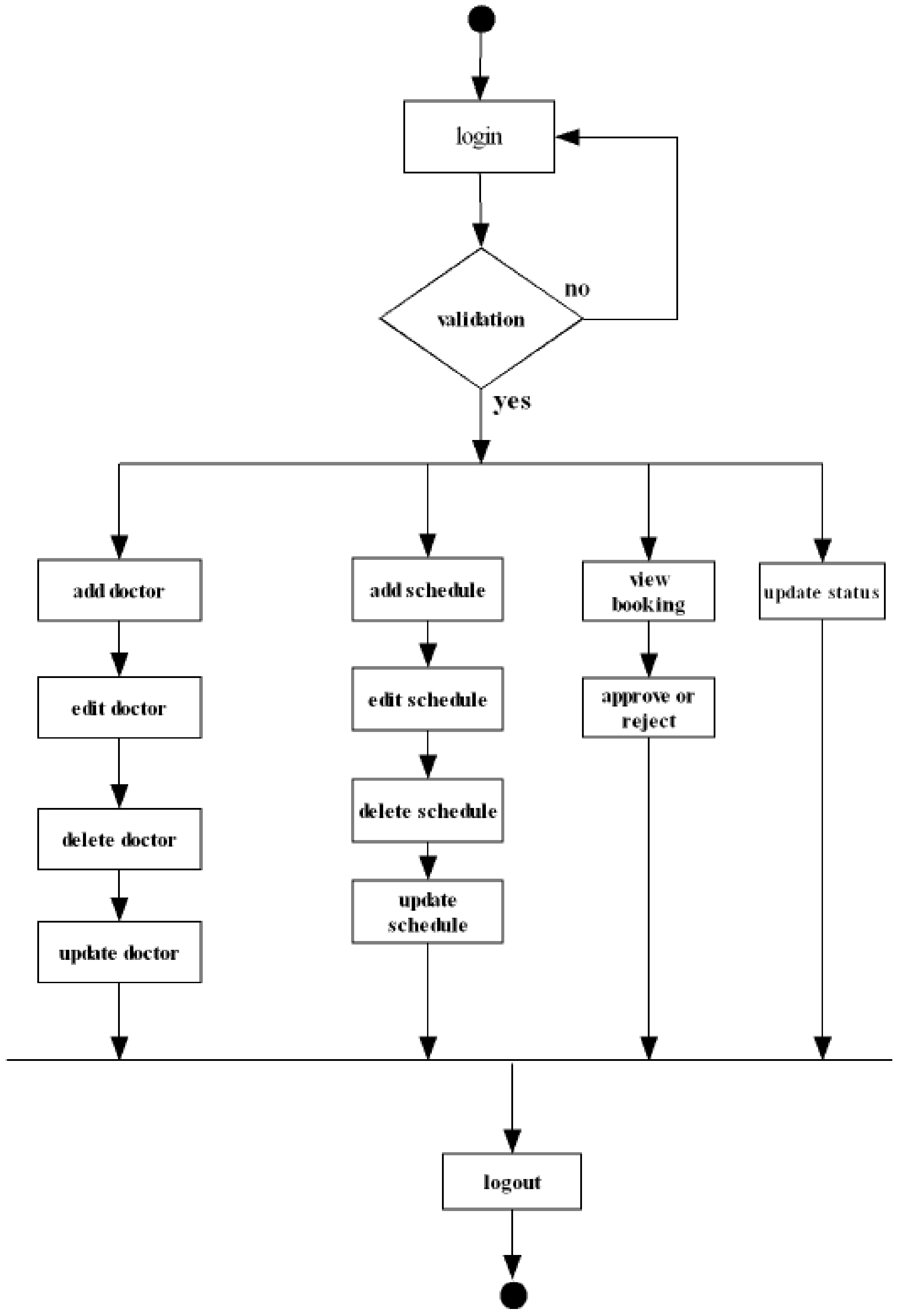
4.6.2 USER


login
validation
no
yes
searching hospital
view doctor list
view scheduling
take appointment
booking history
view worker profile
find services
booking services
send review
view worker rating
report fake services
send complaints
view driver details
booking driver
view nearby location
location to pick and drop
logout

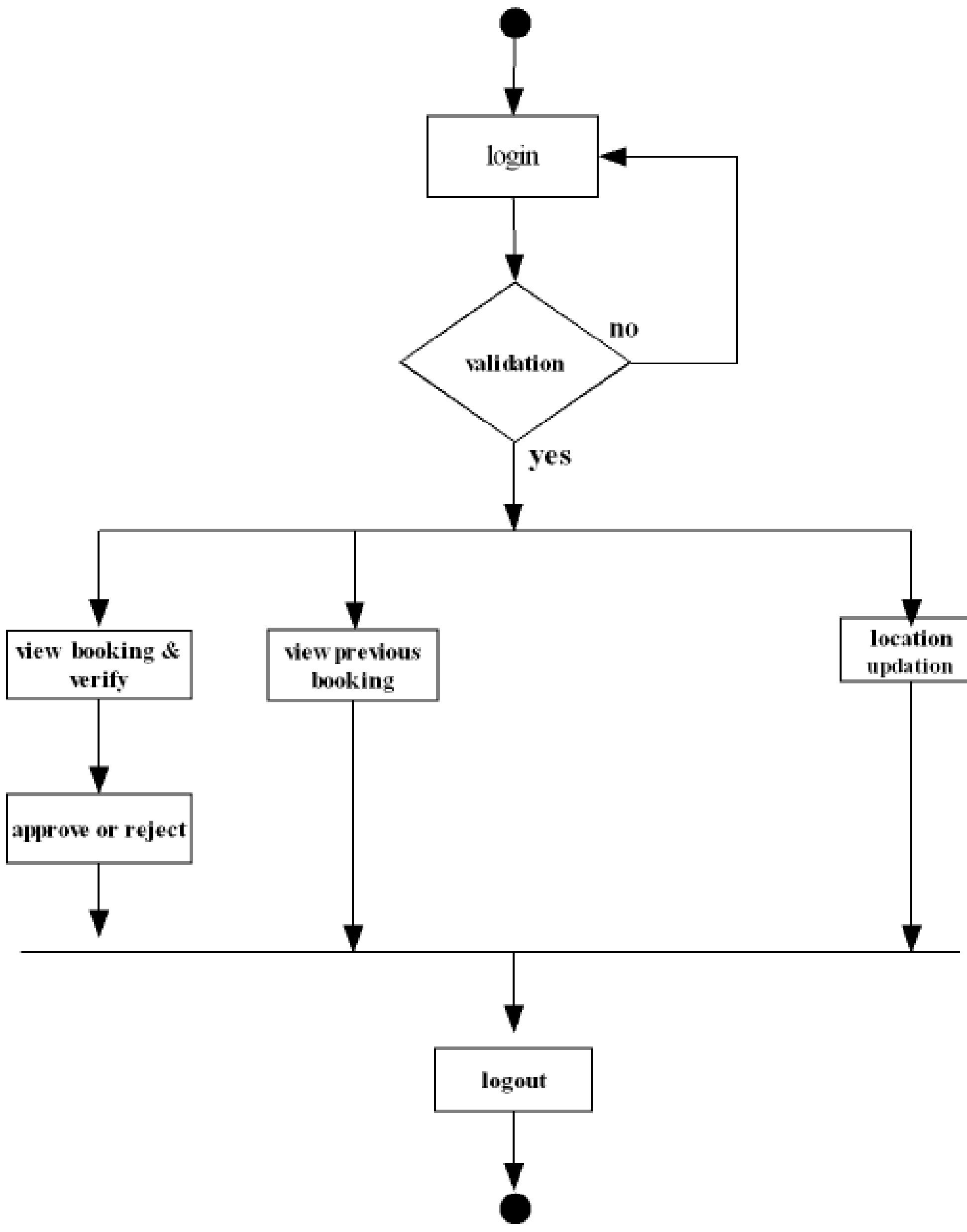
4.6.3 worker


login
validation
no
yes
view booking & verify
approve or reject
view rating
view review
manage own services
add new services
delete services
edit service
update services
logout

4.6.4 Hospital



4.6.2 Driver


login
validation
no
yes
view booking & verify
view previous booking
location updation
approve or reject
logout

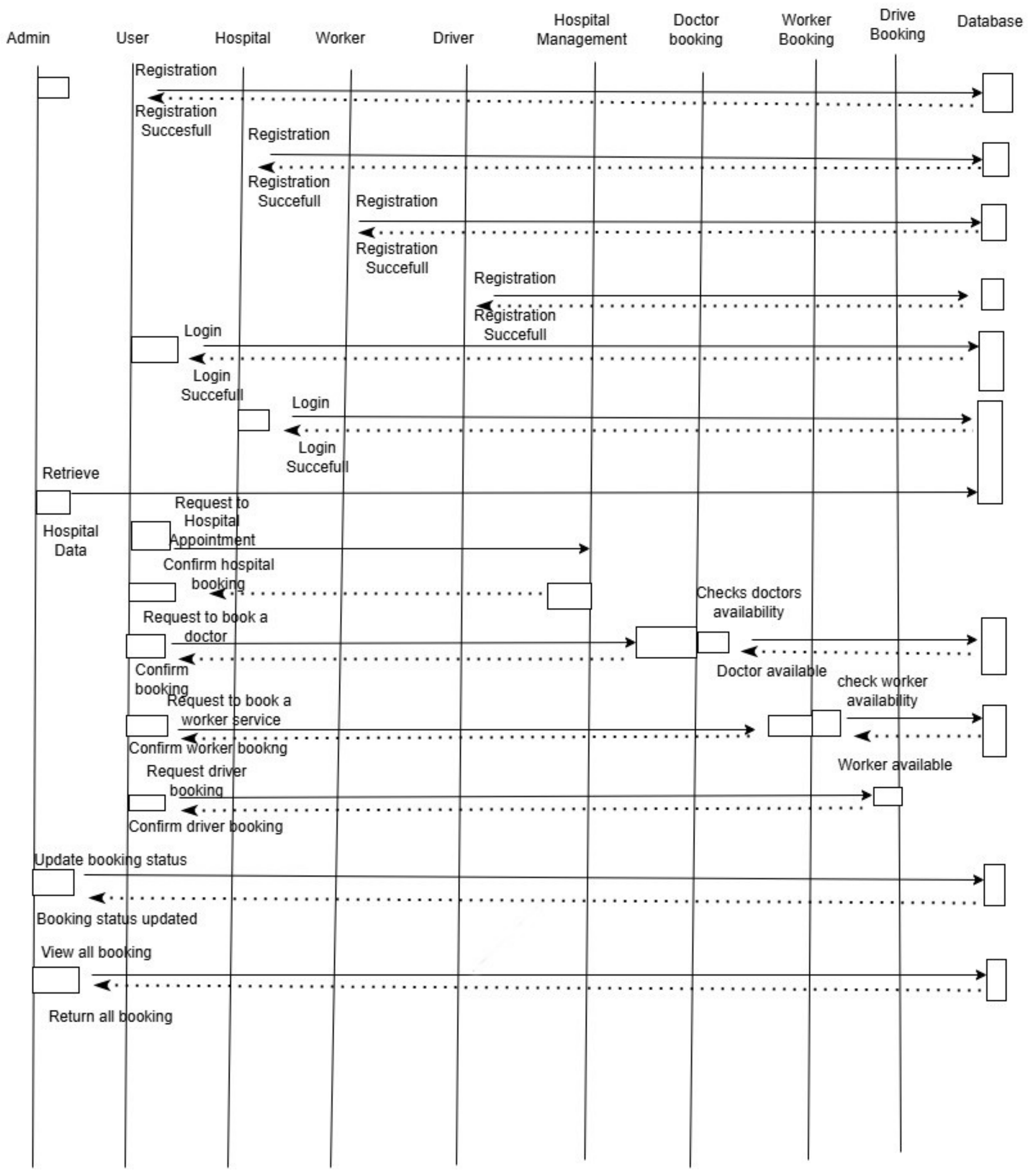
4.7 SEQUENCE DIAGRAM

A sequence diagram is used to describe the glide of messages, events, and actions between objects and to show concurrent processes and activations. This additionally shows time is not without problems depicted in different diagrams. The sequences that sequence diagrams are normally used in the course of analysis and graphs to report and apprehend the logical flow of our system. The sequence diagram's key parts are:

Participant: Acting element or thing in the diagram

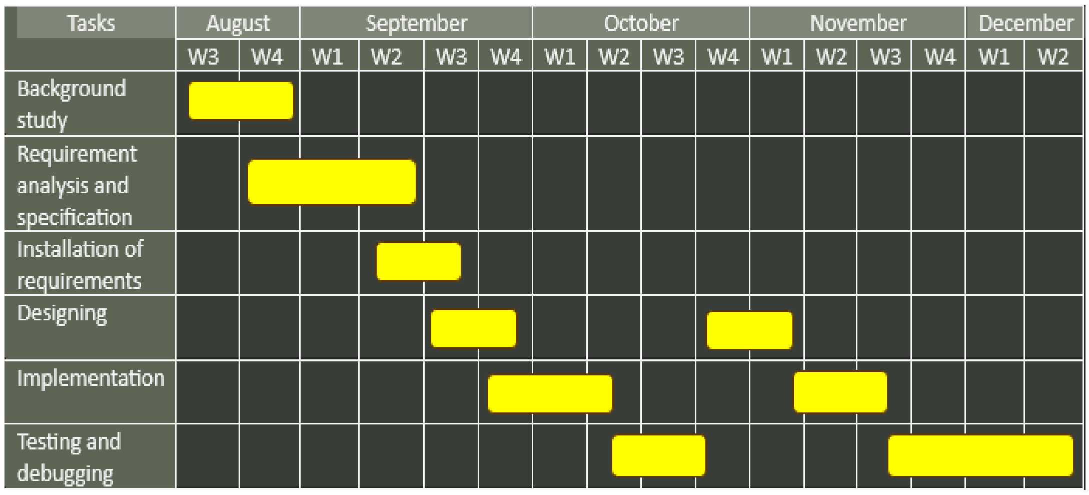
Message: communication between participant objects

Axes: horizontal axes represent which object/participant is acting and vertical represent time.


Admin
User
Hospital
Worker
Driver
Hospital Management
Doctor booking
Worker Booking
Drive Booking
Database
Registration
Registration Succesfull
Registration
Registration Succefull
Registration
Registration Succefull
Registration
Registration Succefull
Login
Login Succefull
Login
Login Succefull
Retrieve
Hospital Data
Request to Hospital Appointment
Confirm hospital booking
Request to book a doctor
Checks doctors availability
Doctor available
Confirm booking
Request to book a worker service
check worker availability
Confirm worker bookng
Worker available
Request driver booking
Confirm driver booking
Update booking status
Booking status updated
View all booking
Return all booking

4.8 GANTT CHART

Below depicts the Gantt chart for The System , according to this gantt chart, I completed the system. This is the representation of the timeallotted for every task I needed to complete for the proper implementation and flow of the system



According to this chart, we will take more time for the requirement analysis and specification. Taking this much time helps us to design and implement the process easier and with less timing. We use agile methodology for implementing our system. It helps to implement new requirements at any point in time.

5. SYSTEM DESIGN AND IMPLEMENTATION

The overall structure of the software is established during this phase of the development process. In this stage, the product's logical structure is developed. It is developed as a data flow diagram that shows how data moves through the system. This stage results in both a high-level and low-level design. The high-level design includes the modules that will be developed and what they will be used for. A more detailed picture of the modules that includes the methods to be specified makes up the low-level design.

5.1 OVERALL DESIGN

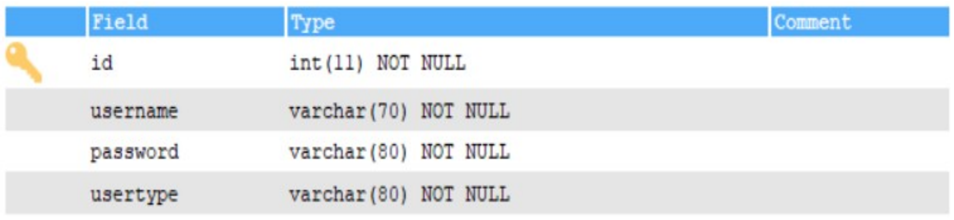
The overall design of the Sync City platform is structured to integrate various city services into a single cohesive system. The application follows a three-tier architecture comprising a front-end user interface, a middle-tier application logic layer, and a back-end database. The front-end, built using HTML, CSS, and JavaScript, ensures a user-friendly interface for seamless navigation and interaction. The back-end, powered by Python Django, handles data processing, authentication, and server-side operations, providing secure and scalable functionality. SQL serves as the database for managing and storing all application data, including user profiles, bookings, services, and transactions. APIs built with Django Rest Framework facilitate smooth communication between the front-end and back-end, enabling real-time updates and data exchange. The design ensures modularity, where each service—like hospital management, tradesman marketplace, and taxi booking—functions independently yet integrates seamlessly within the platform. This approach ensures scalability, maintainability, and ease of adding future enhancements to the system

5.2 DATABASE DESIGN

The database design for Sync City is structured using a relational SQL-based schema to ensure efficient data management and scalability. Key entities like users, hospitals, tradesmen, events, shops, drivers, and farmers are organized into distinct tables, with attributes tailored to their roles. Relationships such as one-to-many and many-to-many are defined using foreign keys to handle interactions like appointments, service bookings, and purchases. Constraints like primary keys and validation rules are implemented to maintain data integrity and prevent duplication. The design ensures flexibility for future enhancements while incorporating mechanisms for data security, backup, and recovery.

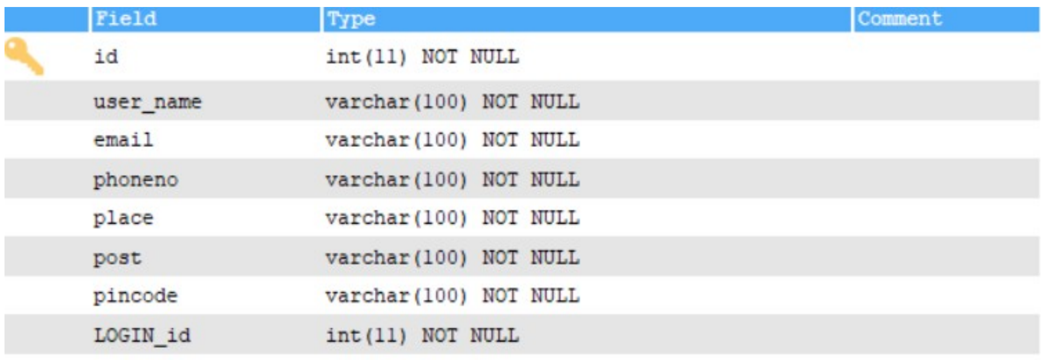
COLLECTIONS :

LOGIN



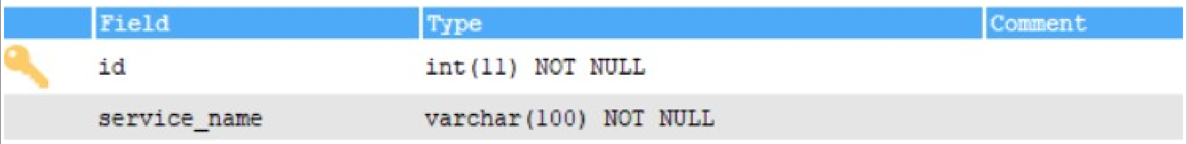
	Field	Type	Comment
🔑	id	int(11) NOT NULL	
	username	varchar(70) NOT NULL	
	password	varchar(80) NOT NULL	
	usertype	varchar(80) NOT NULL	

USERS



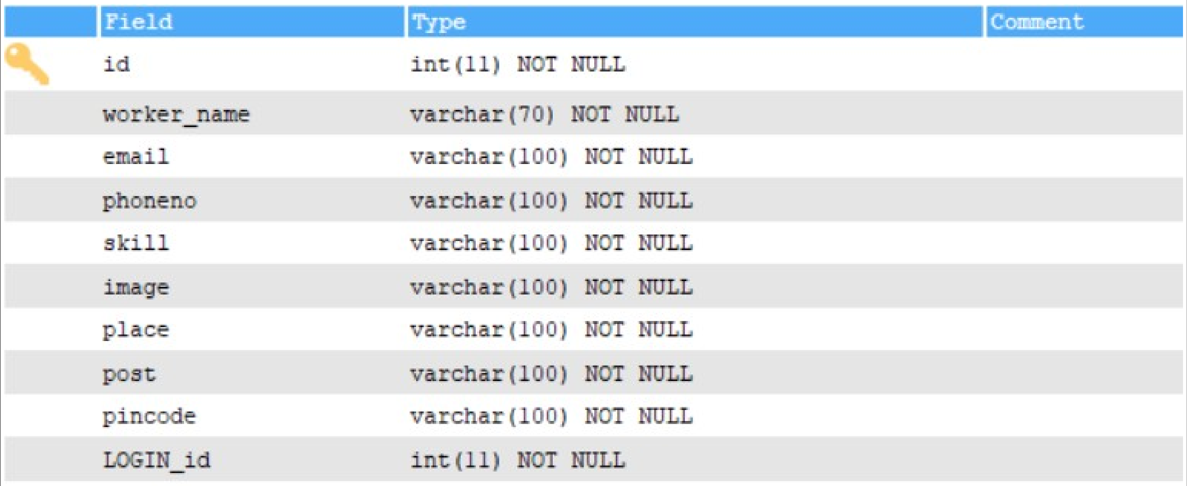
	Field	Type	Comment
🔑	id	int(11) NOT NULL	
	user_name	varchar(100) NOT NULL	
	email	varchar(100) NOT NULL	
	phoneno	varchar(100) NOT NULL	
	place	varchar(100) NOT NULL	
	post	varchar(100) NOT NULL	
	pincode	varchar(100) NOT NULL	
	LOGIN_id	int(11) NOT NULL	

Service



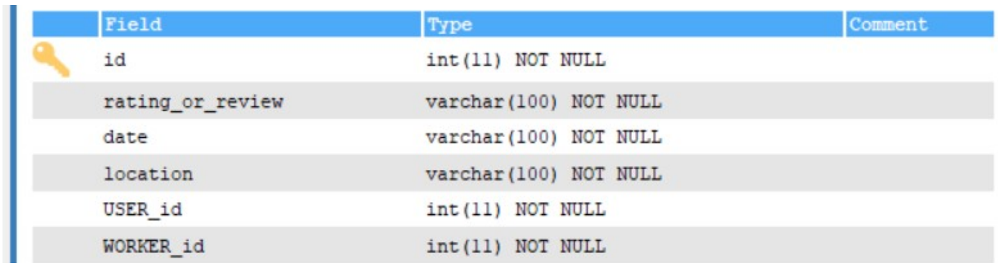
	Field	Type	Comment
🔑	id	int(11) NOT NULL	
	service_name	varchar(100) NOT NULL	

Worker



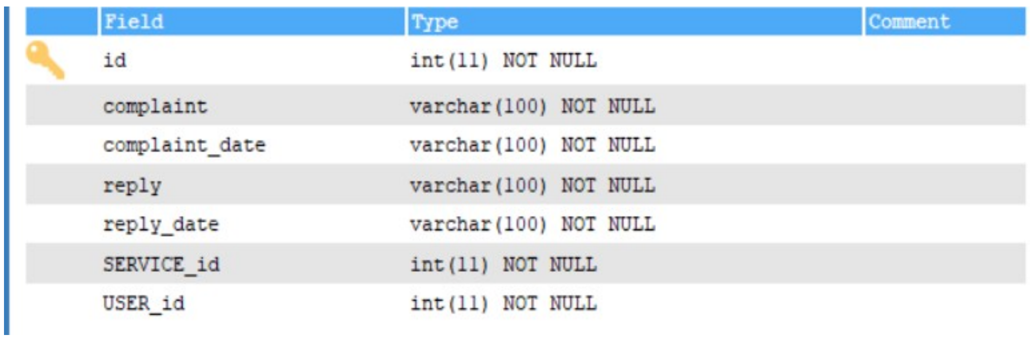
	Field	Type	Comment
🔑	id	int(11) NOT NULL	
	worker_name	varchar(70) NOT NULL	
	email	varchar(100) NOT NULL	
	phoneno	varchar(100) NOT NULL	
	skill	varchar(100) NOT NULL	
	image	varchar(100) NOT NULL	
	place	varchar(100) NOT NULL	
	post	varchar(100) NOT NULL	
	pincode	varchar(100) NOT NULL	
	LOGIN_id	int(11) NOT NULL	

Rating and review



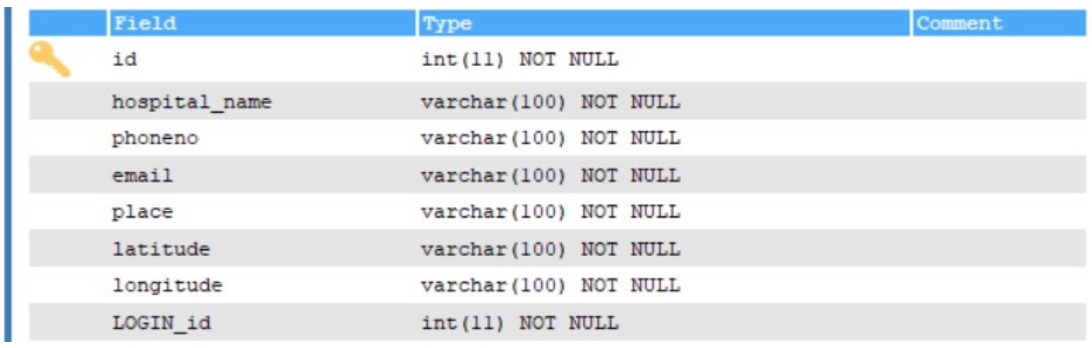
	Field	Type	Comment
🔑	id	int(11) NOT NULL	
	rating_or_review	varchar(100) NOT NULL	
	date	varchar(100) NOT NULL	
	location	varchar(100) NOT NULL	
	USER_id	int(11) NOT NULL	
	WORKER_id	int(11) NOT NULL	

Complaint



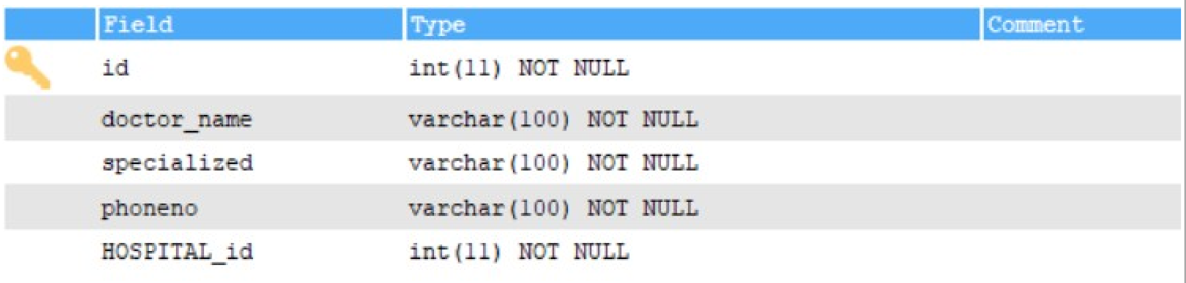
	Field	Type	Comment
🔑	id	int(11) NOT NULL	
	complaint	varchar(100) NOT NULL	
	complaint_date	varchar(100) NOT NULL	
	reply	varchar(100) NOT NULL	
	reply_date	varchar(100) NOT NULL	
	SERVICE_id	int(11) NOT NULL	
	USER_id	int(11) NOT NULL	

Hospital



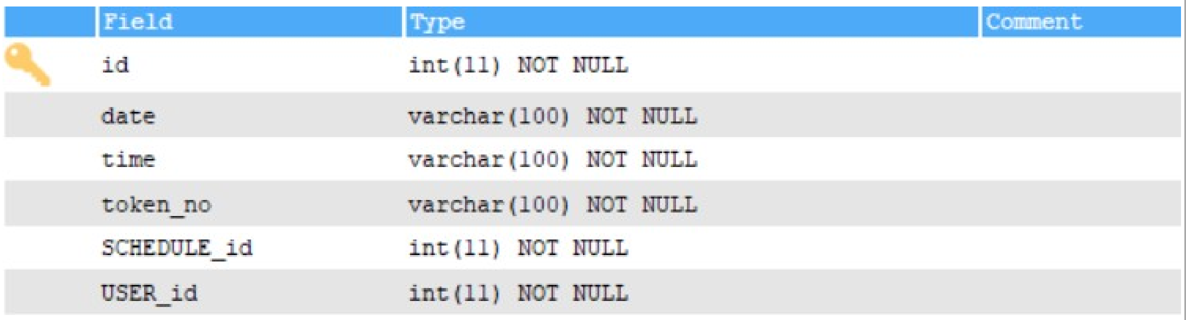
	Field	Type	Comment
🔑	id	int(11) NOT NULL	
	hospital_name	varchar(100) NOT NULL	
	phoneno	varchar(100) NOT NULL	
	email	varchar(100) NOT NULL	
	place	varchar(100) NOT NULL	
	latitude	varchar(100) NOT NULL	
	longitude	varchar(100) NOT NULL	
	LOGIN_id	int(11) NOT NULL	

Doctor



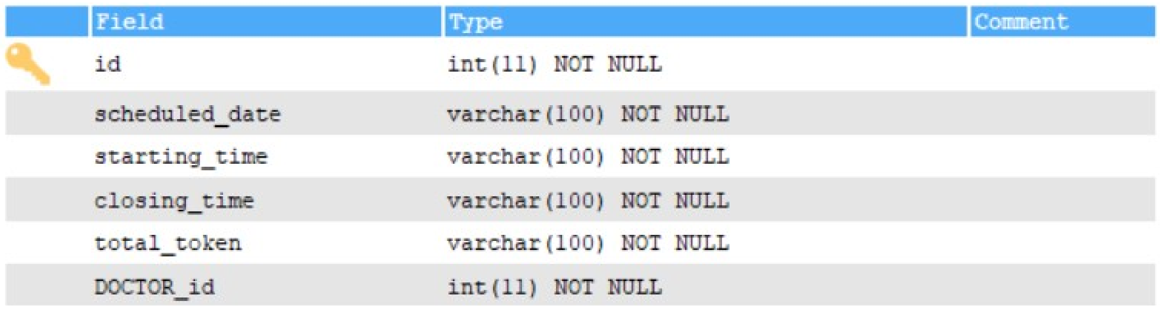
	Field	Type	Comment
🔑	id	int(11) NOT NULL	
	doctor_name	varchar(100) NOT NULL	
	specialized	varchar(100) NOT NULL	
	phoneno	varchar(100) NOT NULL	
	HOSPITAL_id	int(11) NOT NULL	

Doctor booking



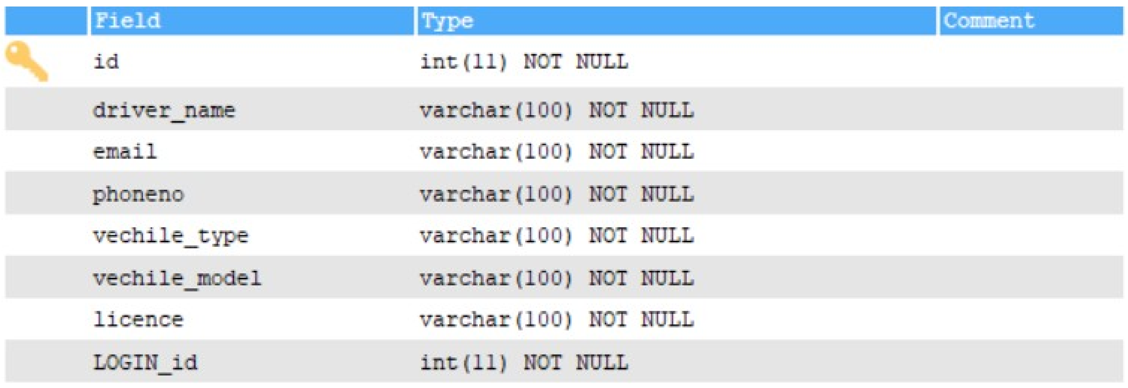
	Field	Type	Comment
🔑	id	int(11) NOT NULL	
	date	varchar(100) NOT NULL	
	time	varchar(100) NOT NULL	
	token_no	varchar(100) NOT NULL	
	SCHEDULE_id	int(11) NOT NULL	
	USER_id	int(11) NOT NULL	

Schedule



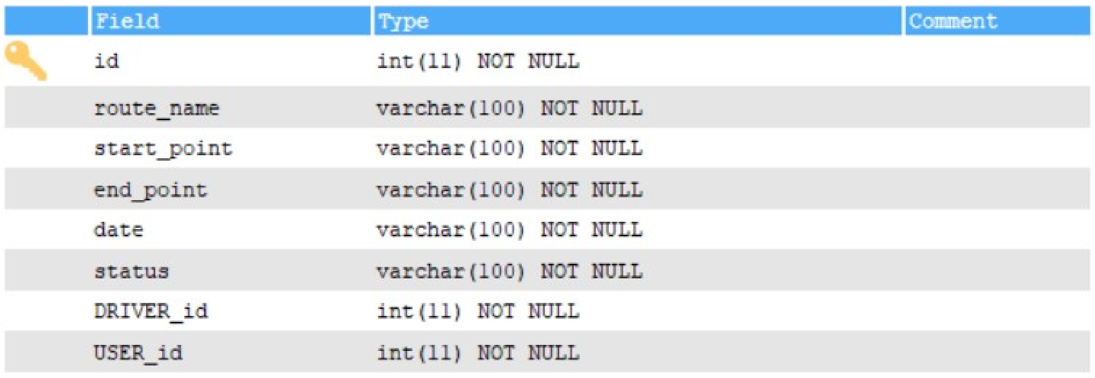
	Field	Type	Comment
🔑	id	int(11) NOT NULL	
	scheduled_date	varchar(100) NOT NULL	
	starting_time	varchar(100) NOT NULL	
	closing_time	varchar(100) NOT NULL	
	total_token	varchar(100) NOT NULL	
	DOCTOR_id	int(11) NOT NULL	

Driver



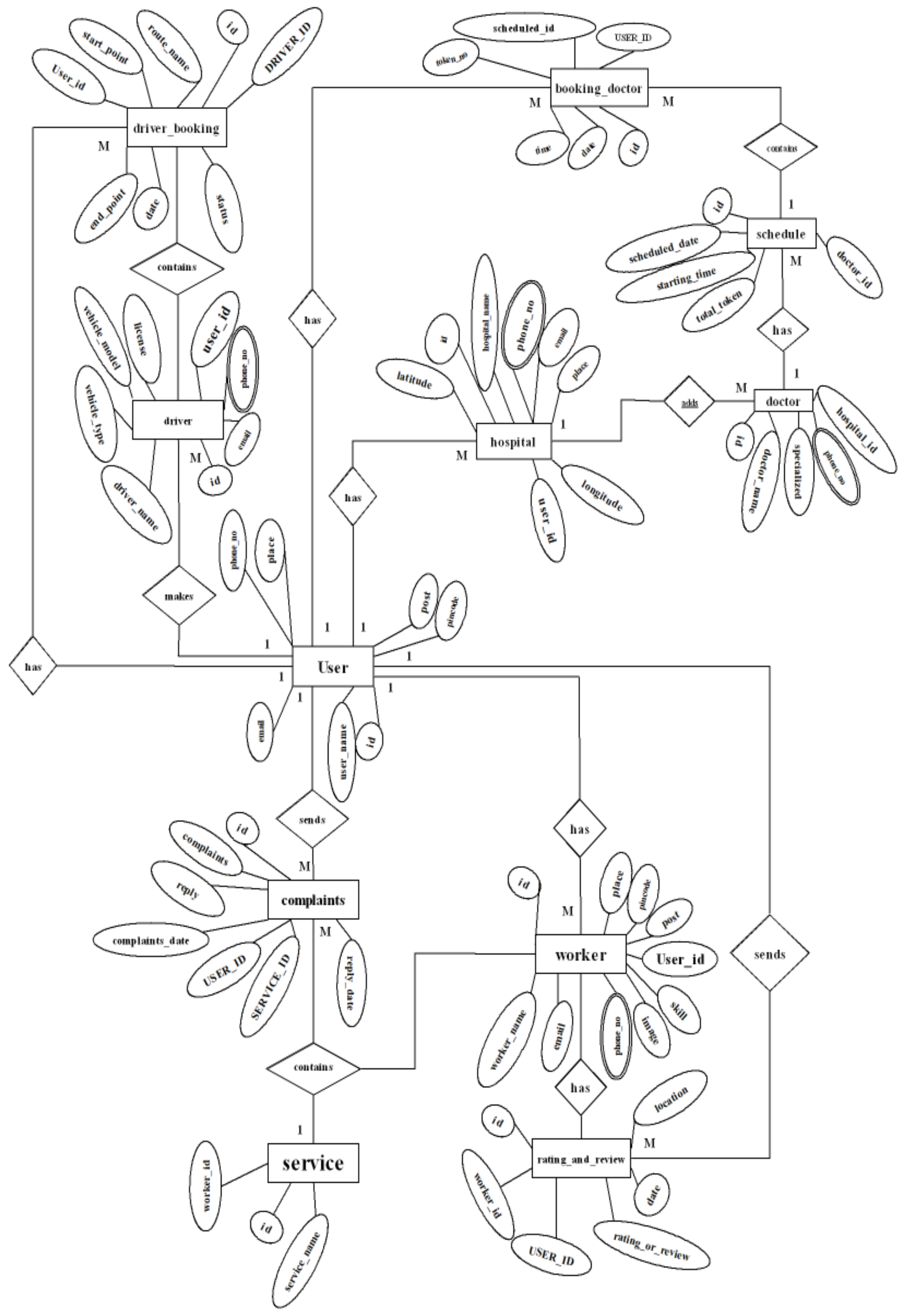
	Field	Type	Comment
🔑	id	int(11) NOT NULL	
	driver_name	varchar(100) NOT NULL	
	email	varchar(100) NOT NULL	
	phoneno	varchar(100) NOT NULL	
	vechile_type	varchar(100) NOT NULL	
	vechile_model	varchar(100) NOT NULL	
	licence	varchar(100) NOT NULL	
	LOGIN_id	int(11) NOT NULL	

Driver booking



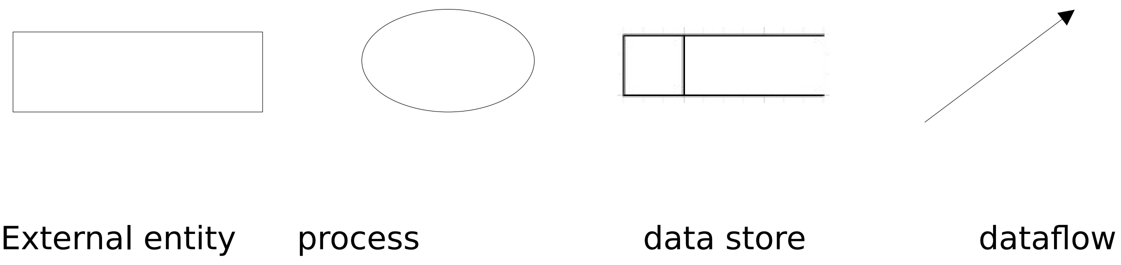
	Field	Type	Comment
🔑	id	int(11) NOT NULL	
	route_name	varchar(100) NOT NULL	
	start_point	varchar(100) NOT NULL	
	end_point	varchar(100) NOT NULL	
	date	varchar(100) NOT NULL	
	status	varchar(100) NOT NULL	
	DRIVER_id	int(11) NOT NULL	
	USER_id	int(11) NOT NULL	

5.3 ER DIAGRAM



5.4 DATA FLOW DIAGRAM

The "flow" of records through a facts system is graphically represented in a data flow diagram (DFD), which models its process elements. A DFD is typically used as an initial stage to develop a system overview that may then be expanded. A DFD outlines the kind of records that will be fed into and output from the system, where they will originate and end up, and where they will be stored. Below is a list of some of the notations used in the DFD.



Function Symbol:

A function has represented the usage of a circle. This image is known as a manner or a bubble. Bubbles are annotated with the names of corresponding functions.

External Entity Symbol:

An exterior entity such as users and criticism details is represented through a rectangle. The external entities are essentially those physical entities external to the software program system, which interact with the gadget via inputting records to the gadget or through eating the information produced by the system.

Data Flow Symbol:

A data flow symbol is either an arrow or a directed arc. This represents the information waft taking place between two processes or between an external entity and a system in route of the Data Flow Arrow. Data Flow symbols are annotated with corresponding record names.

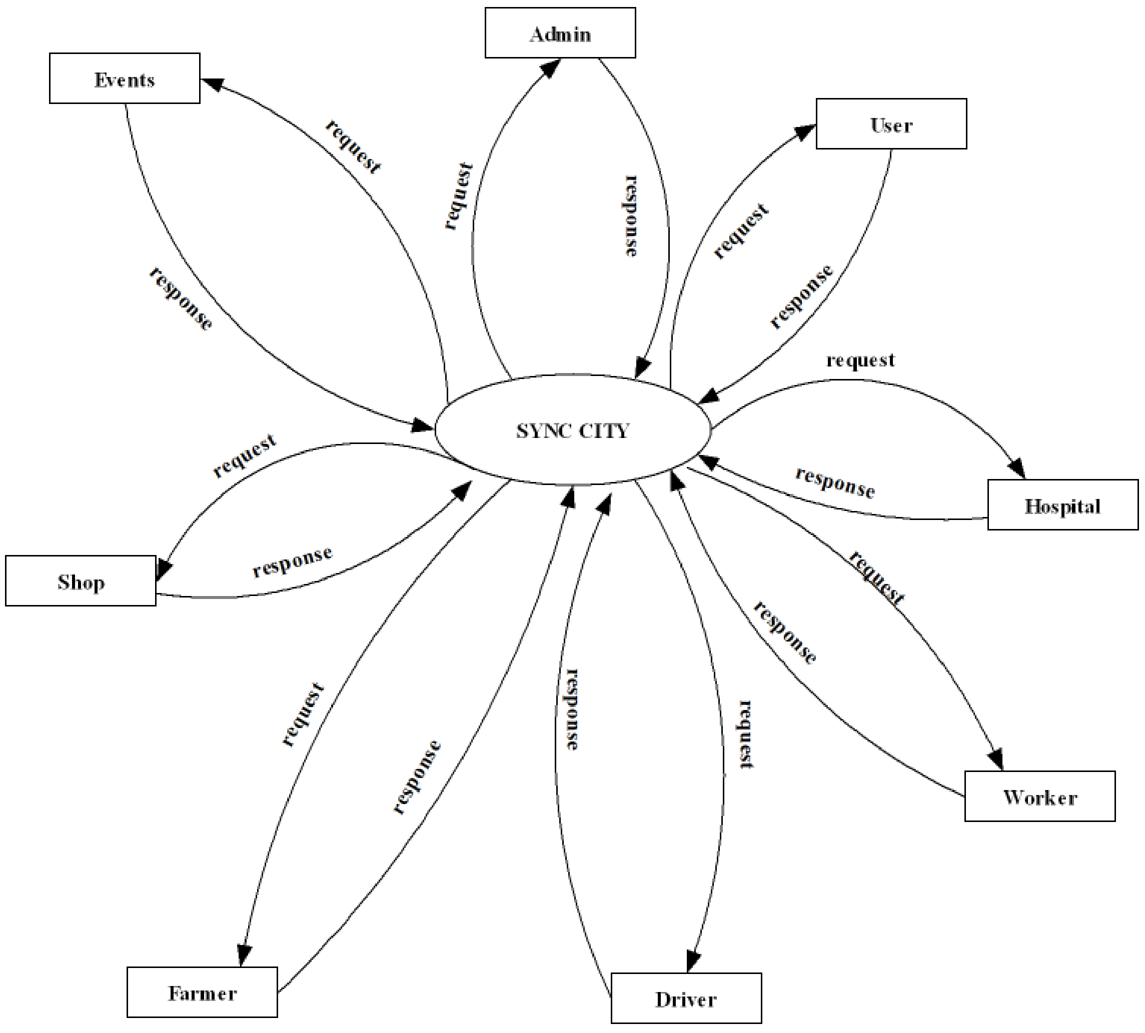
Data Store Symbol:

A Data Store represents a logical file; it is represented by the usage of two parallel lines. A logical file can characterize either Data Store Symbol, which can characterize either information structure or bodily file on disk.

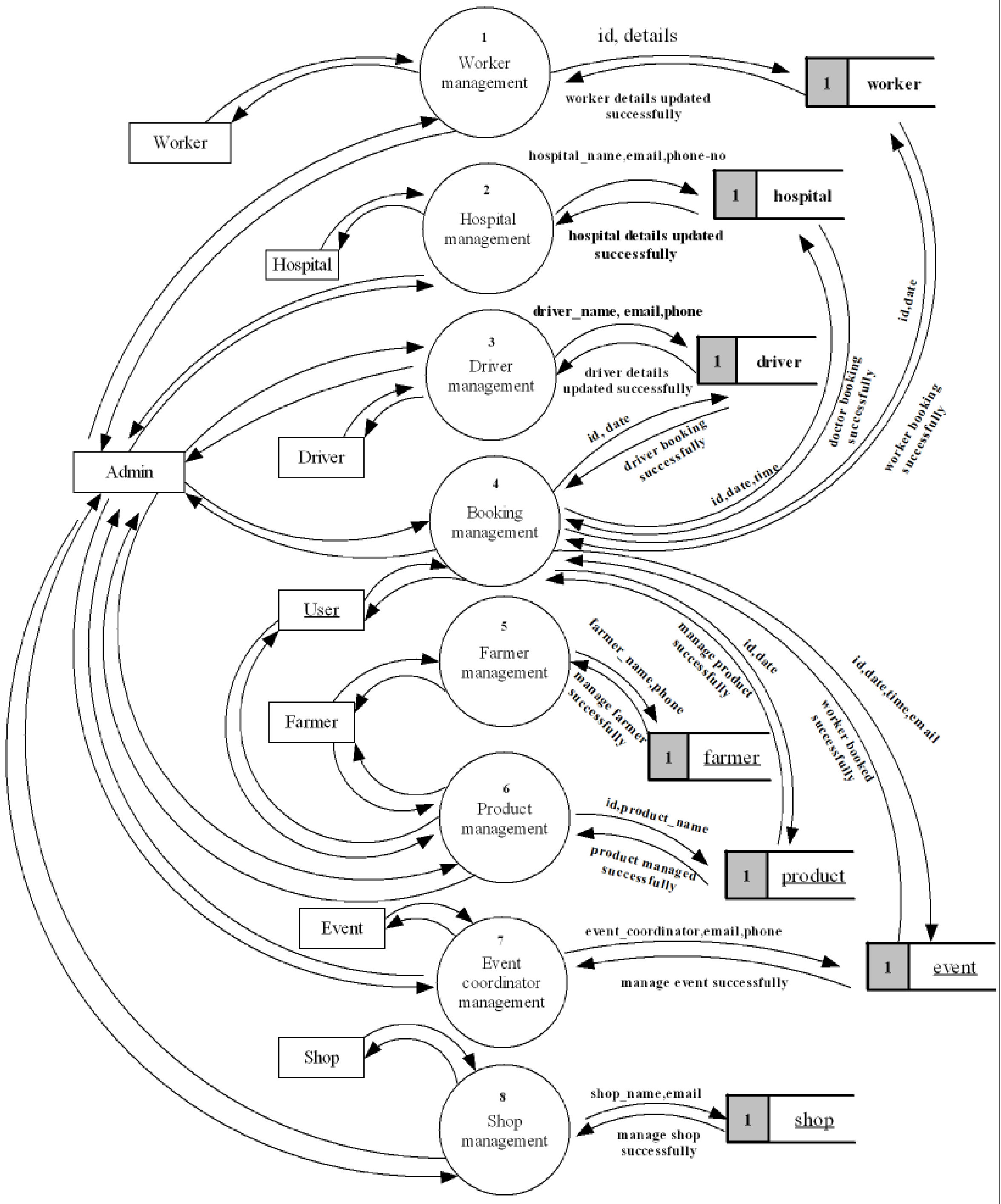
Output Symbol:

The output image is used when a tough copy is produced and the person of the copies cannot be genuinely specified or there are several users of the output. The DFD at the easiest degree is referred to as the Context Analysis Diagram. These are accelerated by using level's, each explaining in technique in detail. Processes are numbered for effortless identification and are commonly labeled in block letters. Each record's flow is labeled for easy understanding.

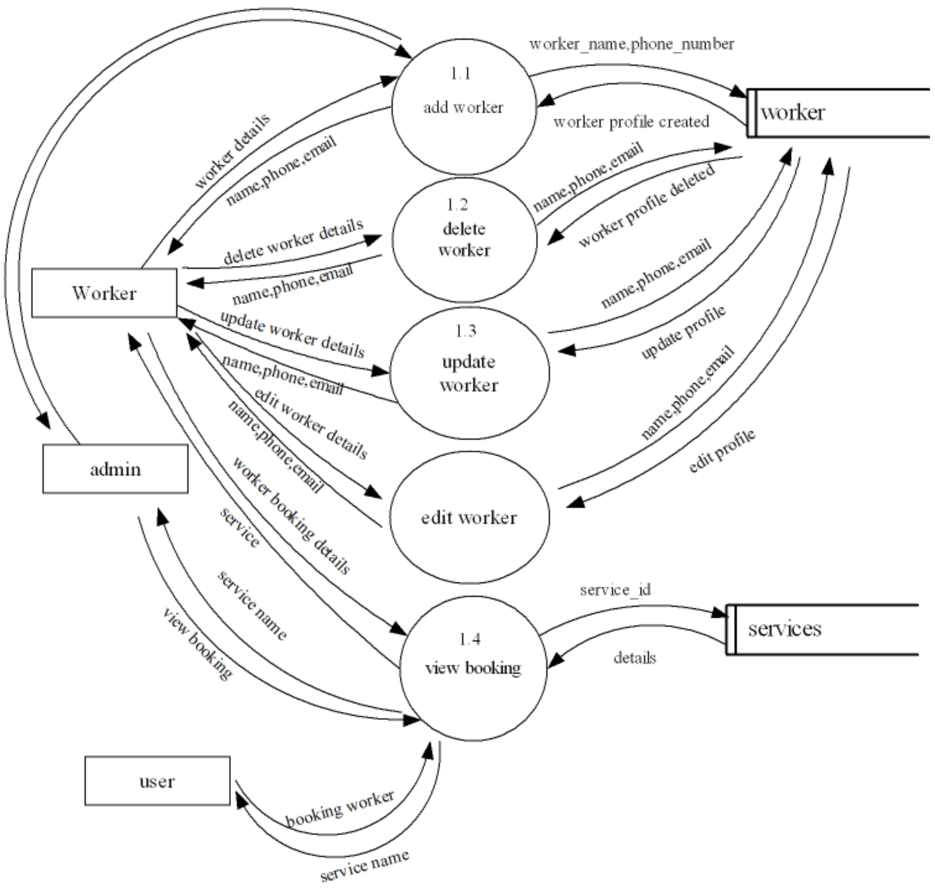
LEVEL 0:


Admin
Events
User
SYNC CITY
Hospital
Shop
Worker
Farmer
Driver
request
response

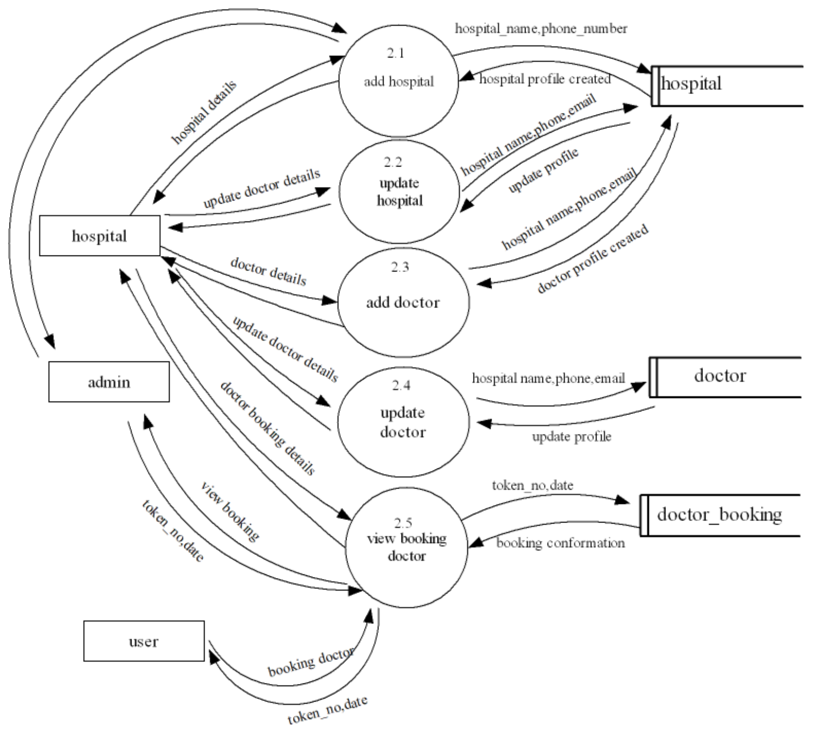
LEVEL 1


1 Worker management
2 Hospital management
3 Driver management
4 Booking management
5 Farmer management
6 Product management
7 Event coordinator management
8 Shop management
Worker
Hospital
Driver
Admin
User
Farmer
Event
Shop
1 worker
1 hospital
1 driver
1 farmer
1 product
1 event
1 shop
id, details
worker details updated successfully
hospital_name,email,phone-no
hospital details updated successfully
driver_name, email,phone
driver details updated successfully
id, date
driver booking successfully
id,date,time
doctor booking successfully
id,date
worker booking successfully
farmer_name,phone
manage farmer successfully
manage product successfully
id,date
worker booked successfully
id,date,time,email
id,product_name
product managed successfully
event_coordinator,email,phone
manage event successfully
shop_name,email
manage shop successfully

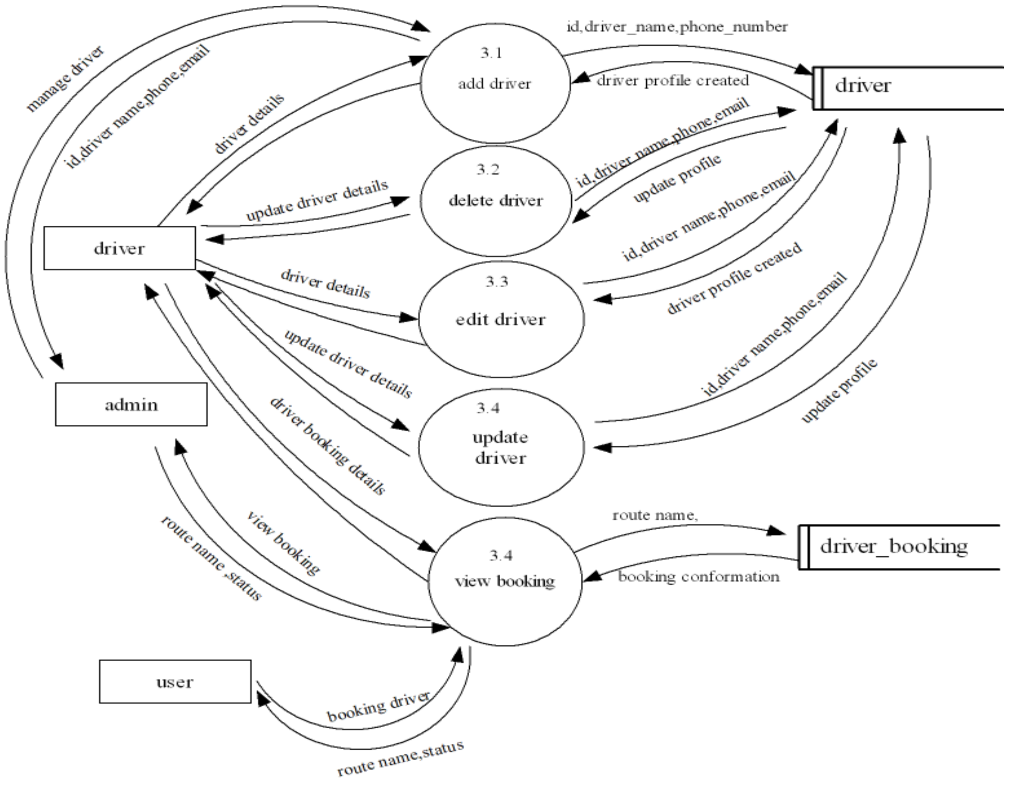
LEVEL 2.1


1.1
add worker
1.2
delete worker
1.3
update worker
edit worker
1.4
view booking
Worker
admin
user
worker
services
worker details
name,phone,email
worker_name,phone_number
worker profile created
delete worker details
name,phone,email
worker profile deleted
update worker details
name,phone,email
update profile
edit worker details
edit profile
worker booking details
service
service name
view booking
service_id
details
booking worker
service name

LEVEL 2.2



LEVEL 2.3



5.6 INTERFACE DESIGN

Interface design is the process of creating a user interface that facilitates and eases user interaction (UI). User interfaces (UI) are created for technologies like computers, software, and mobile devices in order to improve the user experience. Maximizing the efficiency of user interaction in attaining user goals is the user interface's main objective. By creating a user interface that is effective, the user may complete the task at hand without drawing attention to themselves. The use of typography and visual design can make it easier or harder for a user to use the functions of an interface. This might have an impact on specific interactions the user makes.
An effective user interface is :

Concise: In addition to being clear, we should keep things straightforward. When we can describe a feature in one sentence as opposed to three, we must. By making things simpler, this helps Our users save significant time. It takes time and work to maintain clarity and concision at the same time, but the benefits are worthwhile.

Responsive: Fast is a description of responsiveness. The user interface includes some kind of comment. The user interface should engage in conversation with them to inform them of what is happening.

Familiar: Something is familiar when it resembles another object we have seen before. When we are familiar with something, we understand how it functions and what to anticipate. Find familiar elements for our users and incorporate them into our user interface.

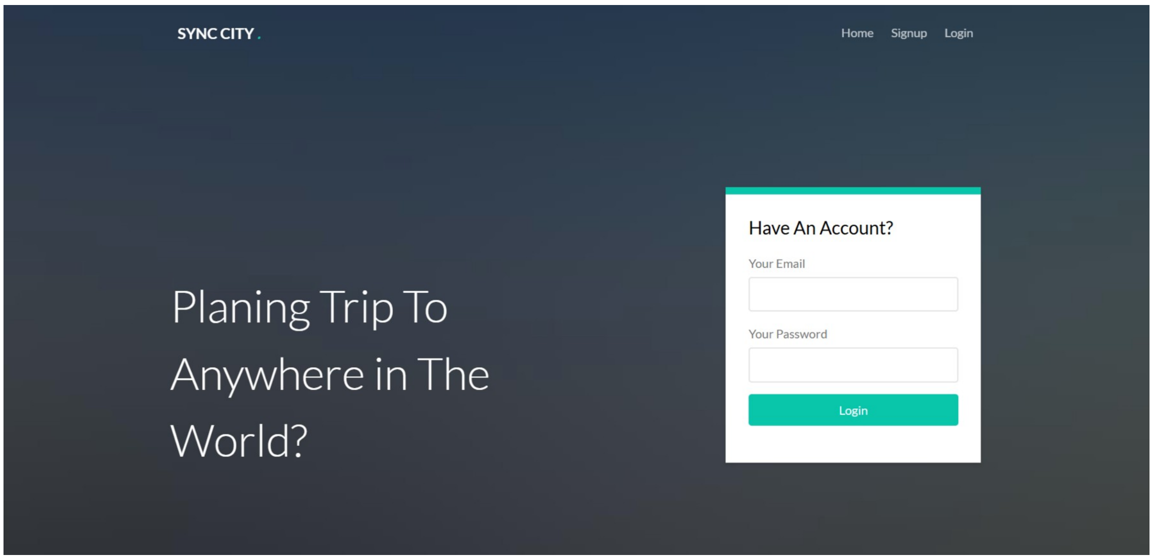
Attractive: In the sense that it makes using the interface enjoyable, an interface is attractive.

Clarity: Designing an intuitive interface is the most challenging part of the process. User interface design is all about communicating meaning and functionality so that people can interact with our systems. People will get confused and frustrated if they don't understand how their application works or where to find it on our website.

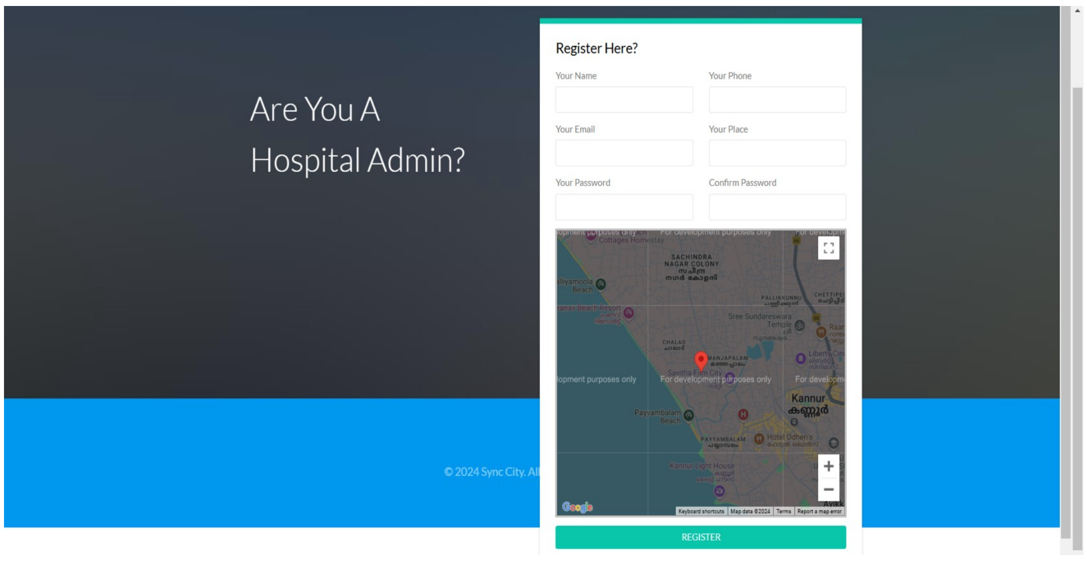
Consistent: Consistent user interfaces enable users to establish usage habits, learn the varied buttons, tabs, icons, and other interface element's appearances, recognize them, and understand what they do in various settings.

Interface screenshot:

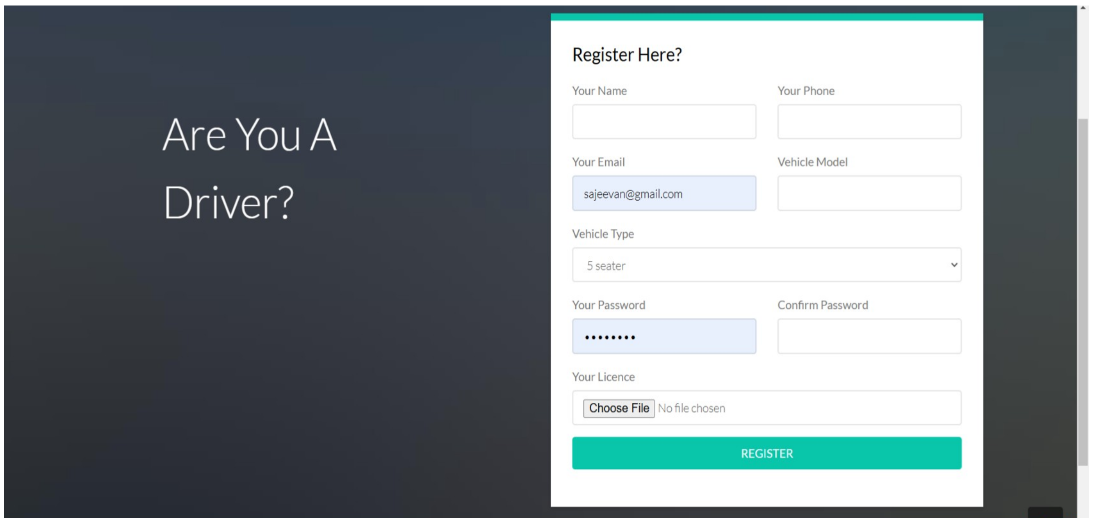
LOGIN PAGE



HOSPITAL REGISTRATION

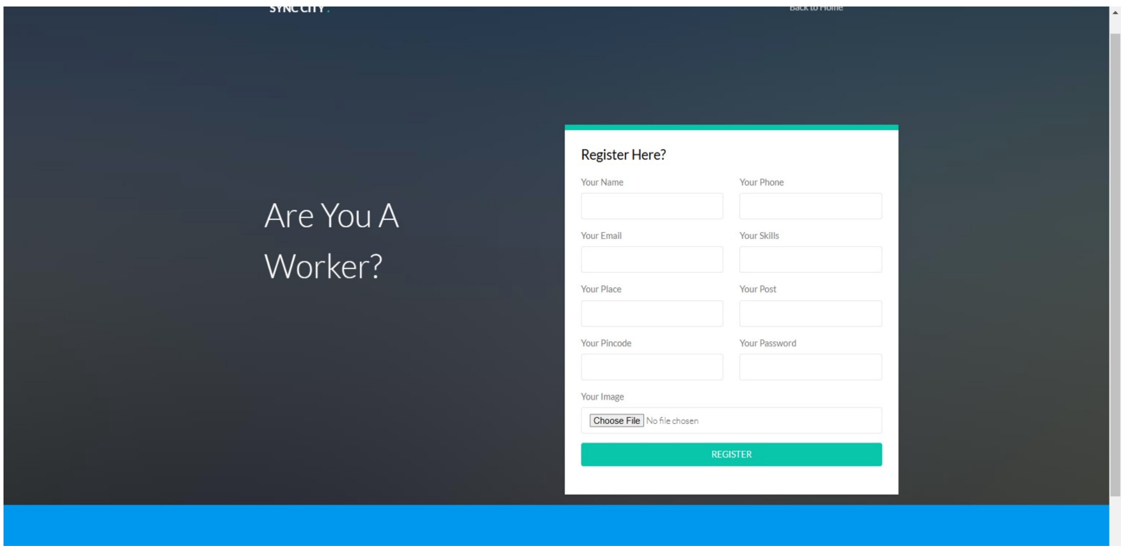


DRIVER REGISTRATION



Your Email: sajeevan@gmail.com
Vehicle Type: 5 seater
Your Password: ••••••••

WORKER REGISTRATION



6.CODING

```
# Create your views here.
def log(request):
    return render(request,'login.html')

def login_post(request):
    username=request.POST['name']
    password=request.POST['pass']
    data=login.objects.filter(username=username,password=password)
    if data.exists():
        data=data[0]
        request.session['lid']=data.id
        if data.usertype == "admin":
            return HttpResponse("<script>alert('login
successfully');window.location='/admin_home'</script>")
        elif data.usertype == "shop":
            return HttpResponse("<script>alert('login
successfully');window.location='/shop_home'</script>")
        elif data.usertype == "driver":
            return HttpResponse("<script>alert('login
successfully');window.location='/driver_home'</script>")
        elif data.usertype == "hospital":
            return HttpResponse("<script>alert('login
successfully');window.location='/hospital_home'</script>")

        else:
            return HttpResponse("<script>alert('doesnot exist try
again');window.location='/'</script>")

    return HttpResponse("<script>alert('user is not found,please try
again');window.location='/'</script>")



def admin_home(request):
    return render(request,'admin/admin home.html')



def add_service(request):
    return render(request,'admin/add service.html')

def add_service_post(request):
    ser=request.POST['textfield']
    print(ser)

    obj=service()
    obj.service_name=ser
    obj.save()
    return HttpResponse("added")

def Approve_hospital(request):
    res=hospital.objects.filter(LOGIN__usertype = 'pending')
    return render(request,'admin/Approve hospital.html', {'data':res})

def approved_hospital(request,id):
    login.objects.filter(id=id).update(usertype='hospital')
    return HttpResponse("approved")
def rejected_hospital(request,id):
    login.objects.filter(id=id).update(usertype='hospital')
    return HttpResponse("rejected")
```

```
def send_reply(request,id):
    return render(request,'admin/send reply.html',{"id":id})


def send_reply_post(request,id):
    rep=request.POST['textarea']
    d=datetime.datetime.now()
    complaint.objects.filter(id=id).update(reply=rep,reply_date=d)

    return HttpResponse("replied")


def view_approved_hospital(request):
    data = hospital.objects.filter(LOGIN__usertype = 'hospital')
    return render(request,'admin/view approved hospital.html',
{'data':data})


def view_booking(request):
    data=booking_doctor.objects.all()
    return render(request,'admin/view booking.html',{'data':data})


def view_doctor(request):
    data=doctor.objects.all()
    return render(request,'admin/view doctor.html',{'data':data})


def view_driver_and_verify(request):
    data=driver.objects.all()
    return render(request,'admin/view driver and verify.html',
{'data':data})
def approved_driver(request,id):
    login.objects.filter(id=id).update(usertype='driver')
    return HttpResponse("approved")
def rejected_driver(request,id):
    login.objects.filter(id=id).update(usertype='driver')
    return HttpResponse("rejected")

def view_driver_booking(request):
    data=driver_booking.objects.all()
    return render(request,'admin/view driver booking.html',{'data':data})

def view_fake_report(request):
    data=complaint.objects.all()
    return render(request,'admin/view fake report.html',{"data":data})



def view_previous_booking(request):
    data=booking_doctor.objects.all()
    return render(request,'admin/view previous booking.html',
{"data":data})



def view_registered_worker(request):
    data=worker.objects.all()
    return render(request,'admin/view registered worker.html',
{"data":data})
def approved_worker(request,id):
    login.objects.filter(id=id).update(usertype='worker')
    return HttpResponse("approved")
def rejected_worker(request,id):
    login.objects.filter(id=id).update(usertype='worker')
    return HttpResponse("rejected")



def view_service(request):
    data=service.objects.all()
    return render(request,'admin/view service.html',{"data":data})
```

USER

```
def and_register(request):
    hn = request.POST['name']
    e = request.POST['email']
    ph = request.POST['phone']
    pl = request.POST['place']
    po = request.POST['post']
    pi = request.POST['pin']
    pa = request.POST['passwd']

    v = login.objects.filter(username=e)
    if v.exists():
        return JsonResponse({'status':"no"})
    else:
        obj = login()
        obj.username = e
        obj.password = pa
        obj.usertype = 'user'
        obj.save()

        qry=user()
        qry.user_name=hn
        qry.phoneno=ph
        qry.email=e
        qry.place=pl
        qry.post=po
        qry.pincode=pi
        qry.LOGIN=obj
        qry.save()

        return JsonResponse({'status':"ok"})

def and_login(request):
    username=request.POST['name']
    password=request.POST['pass']
    data=login.objects.filter(username=username,password=password)
    if data.exists():
        data=data[0]
        if data.usertype == "user":
            return JsonResponse({'status':'ok', 'lid':data.id})
        else:
            return JsonResponse({'status':'no'})
    else:
        return JsonResponse({'status': 'invalid'})

def and_profile(request):
    lid=request.POST['lid']
    res=user.objects.get(LOGIN_id=lid)
    return JsonResponse({'status':'ok', "name":res.user_name,
"email":res.email, "phone":res.phoneno,
                         "place":res.place, "post":res.post, "pin":res.pincode})

def and_update_profile(request):
    hn = request.POST['name']
    ph = request.POST['phone']
    pl = request.POST['place']
    po = request.POST['post']
    pi = request.POST['pin']
    lid = request.POST['lid']

    qry=user.objects.get(LOGIN_id=lid)
    qry.user_name=hn
    qry.phoneno=ph
    qry.place=pl
    qry.post=po
    qry.pincode=pi
    qry.save()

    return JsonResponse({'status':"ok"})

def and_view_hosp(request):
    res=hospital.objects.filter(LOGIN__usertype="hospital")
    ar=[]
    for i in res:
        ar.append({
            "id":i.id, "hname":i.hospital_name, "phone":i.phoneno,
"email":i.email,
            "place":i.place, "lati":i.latitude, "logi":i.longitude
        })
    return JsonResponse({'status': 'ok', 'data':ar})

def and_view_doctors(request):
    hid=request.POST['hid']
    res=doctor.objects.filter(HOSPITAL_id=hid)
    ar = []
    for i in res:
        ar.append({
            "id": i.id, "docname": i.doctor_name, "specialized": i.specialized,
"phone": i.phoneno
        })
    return JsonResponse({'status': 'ok', 'data': ar})

def and_view_schedule(request):
    did=request.POST['did']
    dt=datetime.datetime.now().date()
    res=schedule.objects.filter(DOCTOR_id=did, scheduled_date__gt=dt)
    ar = []
    for i in res:
        ar.append({
            "id": i.id, "sch_date": i.scheduled_date, "start_time":
i.starting_time, "end_time": i.closing_time,
            "tokens":i.total_token
        })
    return JsonResponse({'status': 'ok', 'data': ar})

def and_book_schedule(request):
    lid=request.POST['lid']
    sid=request.POST['sid']
    res=booking_doctor.objects.filter(USER__LOGIN_id=lid,
SCHEDULE_id=sid)
    if res.exists():
        return JsonResponse({'status':"already"})
    else:
        res2=booking_doctor.objects.filter(SCHEDULE_id=sid)
        if res2.exists():
            res2=res2[0]
            tkn=res2.token_no
            new_tkn=int(tkn)+1
            obj=booking_doctor()
            obj.USER=user.objects.get(LOGIN_id=lid)
            obj.SCHEDULE_id=sid
            obj.status="pending"
            obj.date=datetime.datetime.now().date()
            obj.time=datetime.datetime.now().strftime("%H:%M")
            obj.token_no=new_tkn
            obj.save()
        else:
            obj = booking_doctor()
            obj.USER = user.objects.get(LOGIN_id=lid)
            obj.SCHEDULE_id = sid
            obj.status = "pending"
            obj.date = datetime.datetime.now().date()
            obj.time = datetime.datetime.now().strftime("%H:%M")
            obj.token_no = 1
            obj.save()
        return JsonResponse({'status': "ok"})



def and_view_bookings(request):
    lid=request.POST['lid']
    res=booking_doctor.objects.filter(USER__LOGIN_id=lid)
    ar = []
    for i in res:
        ar.append({
            "id": i.id, "booked_on": i.date + " " + i.time, "sch_date":
i.SCHEDULE.scheduled_date,
           "start_time": i.SCHEDULE.starting_time, "end_time":
i.SCHEDULE.closing_time,
           "token_no": i.token_no,
'docname':i.SCHEDULE.DOCTOR.doctor_name,
"spec":i.SCHEDULE.DOCTOR.specialized
        })
    return JsonResponse({'status': 'ok', 'data': ar})

def and_delete_bookings(request):
    bid=request.POST['bkid']
    booking_doctor.objects.filter(id=bid).delete()
    return JsonResponse({'status': 'ok'})
```

-

7.SYSTEM TESTING

Software testing is an important systematic method that requires significant effort from the developer. Proper strategies are needed to test the software successfully. The developer wants to follow a systematic and effective way to test the software. So the testing activity contains the framework and set of activities, which are very important for the success of the project. These activities include planning, designing test cases, execution of program test cases, outcome interpretation, and lastly, collection and management of data. Testing maintains the quality of the software during software development. The important function of testing is to find errors. Effective software testing can lead to the delivery of good-quality software. We can deliver that product to the customer without any trouble. The customer satisfies all requirements, with less cost, and more reliable and accurate results. In the case of ineffective testing, it cannot deliver a good software product.
Software testing is a crucial activity in the case of software development. There are some conditions in the test plan and also these conditions that should be tested. Each module is to be tested, and the modules will also be integrated. A document is produced for different test case specifications. that contains different test cases along with the desired output. At the time of unit testing, appropriate test cases are executed and compared with the desired output. The test report and the error report are the actual output of the testing phase. For each test case, the result of executing the code also exists, and that must be specified in the test report. The error report contains errors that were found during testing and some actions that were taken to remove these errors.

Testing Objectives

Testing is mainly conducted to find errors and check if the software products satisfy the requirements specified in the SRS. The main objectives are

1. Develop a test case that is capable of finding the undiscovered errors in the software.
2. Test for the removal of the types of bugs

3. Test verification

7.1 TYPES OF TESTING

Various types of testing should be conducted before the system is ready for implementation and that should be documented properly. Various types of testing are

- Unit Testing
- Integration Testing
- Validation Testing

Unit Testing

Unit testing is a crucial phase in the development of the Sync City platform, ensuring the reliability and correctness of individual components. Each module, such as hospital management, tradesman services, and event ticket booking, is tested in isolation to verify its functionality. For instance, tests are performed to check if the token-based appointment booking system correctly schedules, updates, and cancels appointments. Similarly, the tradesman marketplace is tested to ensure accurate display of service listings, booking functionality, and ratings submission.

Django's built-in testing framework is used to write and execute test cases for the back-end. These tests validate views, models, and APIs, ensuring the application logic and data interactions function as expected. Front-end unit tests, performed using tools like Selenium, ensure that the user interface behaves correctly, such as form submissions and navigation workflows. Mock data is utilized during testing to avoid affecting the actual database and to simulate edge cases. Regular unit testing throughout development helps identify bugs early, reducing the risk of issues in later stages. The comprehensive approach ensures that all core functionalities operate seamlessly, providing a robust and reliable system for end-users.

Integration Testing

Integration testing for the Sync City platform ensures that different modules and components work together seamlessly as a unified system. The hospital management system is tested with the appointment booking API to verify smooth interaction between the front-end and back-end. Similarly, the tradesman marketplace is integrated with the payment gateway to ensure secure and successful transactions. Event ticket booking is tested with the user authentication system to validate that only registered users can book tickets.

APIs connecting various features, such as user registration, booking, and notifications, are tested to confirm data flow and consistency across modules. Edge cases, like handling incorrect user input or interrupted transactions, are simulated to ensure system resilience. Tools like Postman and Django Rest Framework's test utilities are used to test API endpoints and their integration with the database. Regular integration testing helps identify and resolve issues arising from interactions between modules, ensuring that the system performs cohesively and delivers a smooth user experience

Validation Testing

Validation testing ensures that the Sync City platform meets its requirements and functions as intended. The hospital management module is validated to confirm that users can successfully book, modify, or cancel doctor appointments using the token-based system. The tradesman marketplace is tested to ensure that service listings are displayed accurately and bookings are processed correctly. Event ticket booking is validated to ensure users can select events, view ticket availability, and complete the purchase process without errors.

User input fields, such as registration forms, login credentials, and booking details, are tested to verify proper validation for formats, lengths, and required fields. The system is tested for compatibility across different devices and browsers to ensure consistent performance. Validation testing also includes checking for adherence to user requirements, ensuring the platform delivers the expected functionalities and user experience. This comprehensive testing phase guarantees that the system aligns with stakeholder expectations and operates reliably under real-world conditions.

Black box testing

A software testing method known as "black-box testing" looks only at the functionality of an activity without peeping inside its core components or mechanisms. likewise called functional testing.

The reason for the system's name is because, in the tester's eyes, the software program resembles a black box that one cannot see inside. This system looks for crimes in the following sequences:

- Wrong or absent functionalities
- Interface crimes
- Data structure errors, unauthorised external access to databases
- Behaviour or performance infractions
- Errors in initialization and termination

White box testing

Sometimes referred to as glass box, clear box, open box, and structural testing a method of testing software in which the test data is chosen using complete knowledge of how the test object operates inside. White box testing, as contrast to black box testing, examines workers using specialised understanding of programming law. It is guaranteed during this test that:

1. Each autonomous corridor inside a module has been used at least once before.
2. Examine both the truthful and incorrect sides of every logical opinion.
3. Execute all circles at their boundaries.

In BlackBox testing, we don't bother about the software's internal knowledge; instead, we merely focus on the inputs and activities of the system.

7.2 TEST CASES

It is a document that contains a precondition, desired results, postconditions, and a set of data. It is developed for a particular set of scenarios to verify that it satisfies all the requirements. For test execution, a test case is considered a starting point. The application has a definite outcome by applying the set of input values. The system reaches some endpoint and knows the postcondition of execution.

7.3 VALIDATION AND CHECKS

VALIDATION AND CHECKS FOR SYNC CITY

Admin

Sl No	Process	Input	Expected Output	Obtained Output	Remarks
1	Login	Email, password	Login Successful / Error in Login	Login Successful	Success
2	User Management	User ID, status	User status updated in database.	User status updated successfully.	Success
3	Offer Approval/Reject	Offer details, status	Offer status updated in database.	Offer status updated successfully.	Success
4	Data Review	Record ID, data	Correct data fields displayed for review.	Data displayed correctly for review.	Success

User

Sl No	Process	Input	Expected Output	Obtained Output	Remarks
1	Registration	Name, email, password	Registration Successful / Error	Registration Successful	Success
2	Login	Email, password	Login Successful / Error in Login	Login Successful	Success
3	Booking Services	Service ID, date, time	Booking Successful / Slot not available	Booking Successful	Success
4	Payment	Card details, payment details	Payment Completed / Error in Payment	Payment Completed Successfully	Success

Driver

Sl No	Process	Input	Expected Output	Obtained Output	Remarks
1	Registration	Name, email, password, vehicle info	Registration Successful / Error	Registration Successful	Success
2	Login	Email, password	Login Successful / Error in Login	Login Successful	Success
3	Booking Notification	Booking ID	Booking details displayed correctly.	Booking displayed correctly.	Success

Hospital

Sl No	Process	Input	Expected Output	Obtained Output	Remarks
1	Login	Email, password	Login Successful / Error in Login	Login Successful	Success
2	Doctor Registration	Name, specialization, availability, ID	Doctor registered successfully in the database.	Doctor added successfully	Success
3	Appointment Booking	Patient ID, doctor ID, date, time	Appointment confirmed / Slot unavailable	Appointment confirmed	Success
4	Appointment Review	Appointment ID	Appointment details displayed for review.	Appointment details retrieved correctly	Success

Worker

Sl No	Process	Input	Expected Output	Obtained Output	Remarks
1	Registration	Name, email, password, skills	Registration Successful / Error	Registration Successful	Success
2	Login	Email, password	Login Successful / Error in Login	Login Successful	Success
3	Service Listing	Service details, availability, rate	Service listed successfully in the database.	Service added successfully	Success
4	Service Booking	Service ID, date, time, user ID	Booking Successful / Slot unavailable	Booking confirmed	Success

8.IMPLEMENTATION AND DEPLOYMENT

Implementation refers to the process of putting the newly developed system into practical use, ensuring it operates as intended while meeting user requirements. In the case of Sync City, implementation is critical for the success of its integrated smart city platform, as it involves managing various interconnected modules such as hospital management, tradesman services, and event ticket booking. This phase demands meticulous planning, coordination, and preparation to ensure the platform is user-friendly and stable during its initial operation.

Once the platform's development and testing are complete, the deployment phase begins. This involves making the platform accessible to its intended users, such as residents, businesses, and administrators, while ensuring seamless functionality. Detailed user guides and support resources are provided to familiarize users with the features, such as token-based doctor booking, service advertisements, and city offers. The system's robustness is evaluated through live testing and real-world use cases to identify and address any remaining issues promptly.

The implementation plan includes the following steps: testing all functional modules to ensure reliability, identifying and resolving any bugs or errors in the system, and making necessary adjustments based on real-time user feedback. This comprehensive approach ensures that Sync City achieves its goal of providing an efficient, integrated, and accessible platform for city residents and visitors alike.

8.1 INSTALLATION

Install Python and Django

Python is a powerful, high-level programming language widely used for web development. Django is a Python-based web framework that simplifies the creation of robust and scalable web applications.

- Install Python from the official website: Python.org.
- Install Django using pip

Install Django REST Framework

The Django REST Framework (DRF) is a powerful and flexible toolkit for building APIs in Django.

- Install DRF using pip

Install SQL Database Connector

Sync City uses SQL for backend data storage. Install the necessary connector for the SQL database. For example, for SQLite, it is included by default; for PostgreSQL:

Install Frontend Tools

To enhance the user interface of Sync City:

- Install React.js: A powerful JavaScript library for creating user interfaces.

9. CONCLUSION

The antique buying and selling platform provides an efficient solution for connecting buyers and sellers in the niche market of antiques. By incorporating distinct roles for buyers, sellers, and admins, the platform ensures a streamlined and transparent workflow. Admin verification plays a pivotal role in maintaining the authenticity and integrity of listed items, fostering trust among users. Buyers enjoy a user-friendly interface for browsing and purchasing unique antiques, while sellers benefit from intuitive tools for listing and managing their items. The use of the MERN stack ensures that the system is both scalable and responsive, catering to the needs of modern web applications.

The project addresses the core functionalities of a marketplace and sets the foundation for future enhancements, such as advanced search and feedback mechanisms. With its emphasis on security and simplicity, the platform successfully bridges the gap between technology and the traditional antique trade. The troubleshooting and support features further enhance user satisfaction, making the system reliable and accessible. Overall, this project demonstrates the effective integration of technology into a niche marketplace, creating a trustworthy and efficient environment for antique enthusiasts. It lays the groundwork for future innovations to make the platform even more robust and feature-rich.

10. SUGGESIONS AND FUTURE WORK

Sync City has made great progress in providing a comprehensive platform for city services. However, there are some areas where improvements can be made. One suggestion is to add **multi-language support**, so users from different regions can easily use the platform. Another improvement could be the integration of **real-time data** from services like public transportation, which would allow users to track buses, taxis, or events in real time. The **user interface (UI)** could also be made more user-friendly and customizable, allowing users to prioritize the services they use most. It would be helpful to add **social media integration** so users can share their experiences or promotions with others. As the platform grows, **data security** should be strengthened by implementing better encryption methods to protect user information. Future work could also include adding **AI-powered customer support** to assist users more effectively. Finally, Sync City could explore partnerships with other smart city initiatives to offer more services, like environmental monitoring or smart energy management. These improvements will make Sync City even more useful for residents and businesses

11. BIBLIOGRAPHY

1) Django Software Foundation. (n.d.). *Django: The Web Framework for Perfectionists with Deadlines.* Retrieved from https://www.djangoproject.com

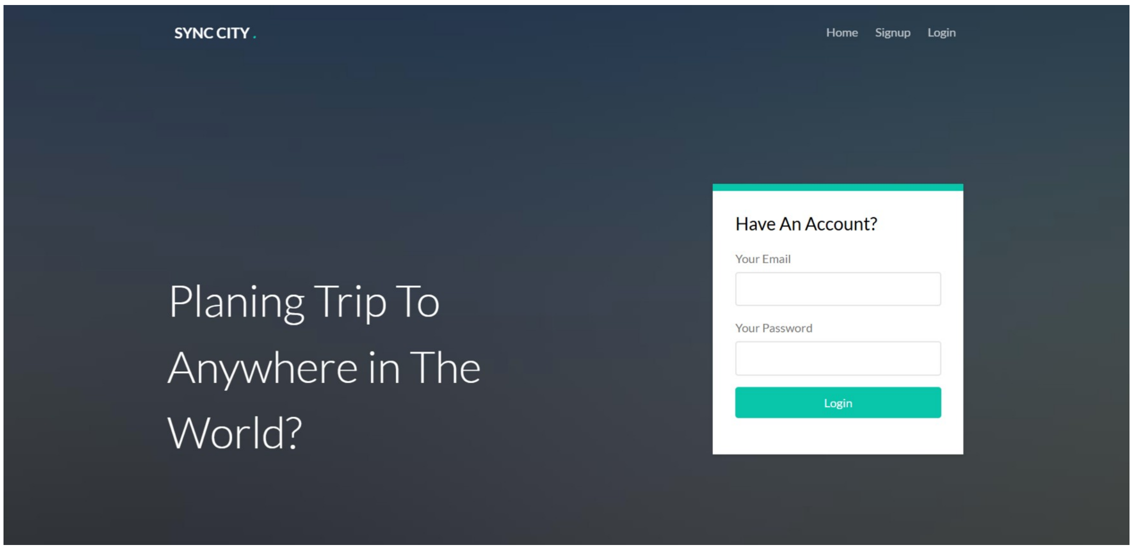
2) Agile Alliance. (n.d.). *What is Agile?* Retrieved from https://www.agilealliance.org

3) Mozilla Developer Network (MDN). (n.d.). *HTML, CSS, and JavaScript Resources*. Retrieved from https://developer.mozilla.org

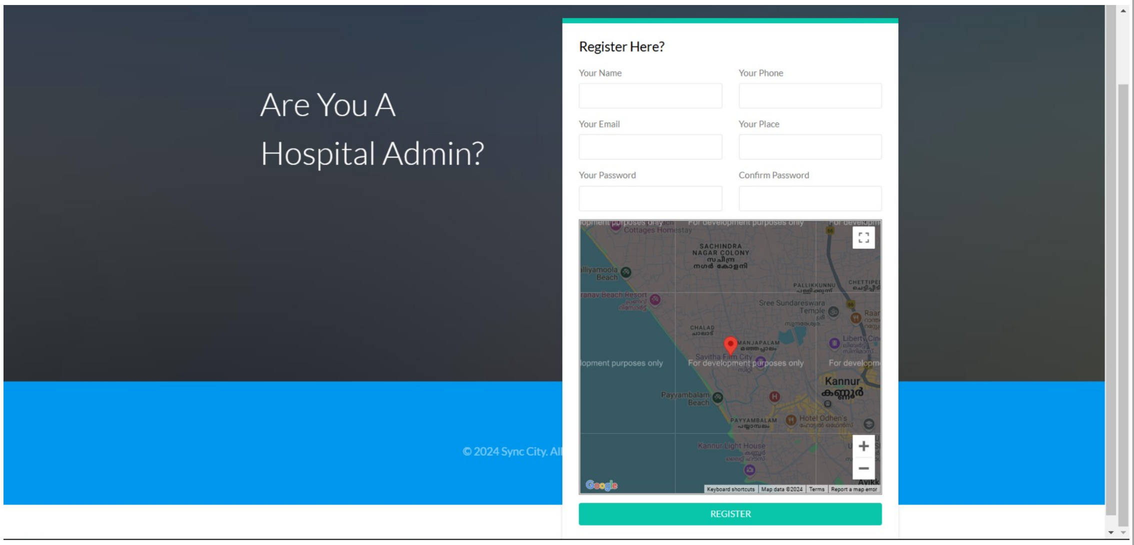
4) Postgresql.org. (n.d.). *PostgreSQL: The World's Most Advanced Open Source Database*. Retrieved from https://www.postgresql.org

12.APPENDICES

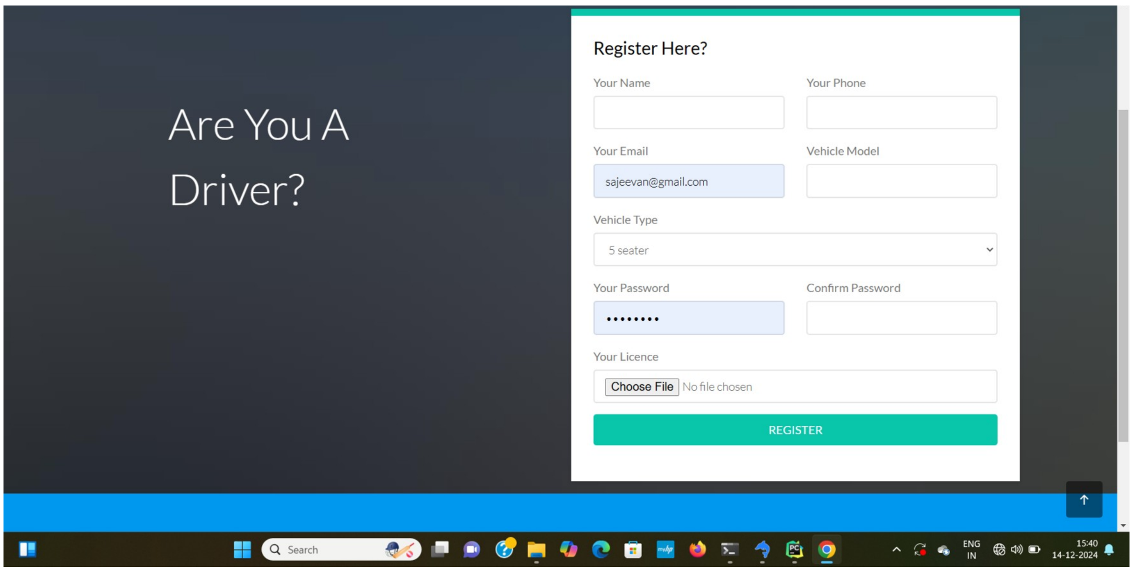
homepage - user



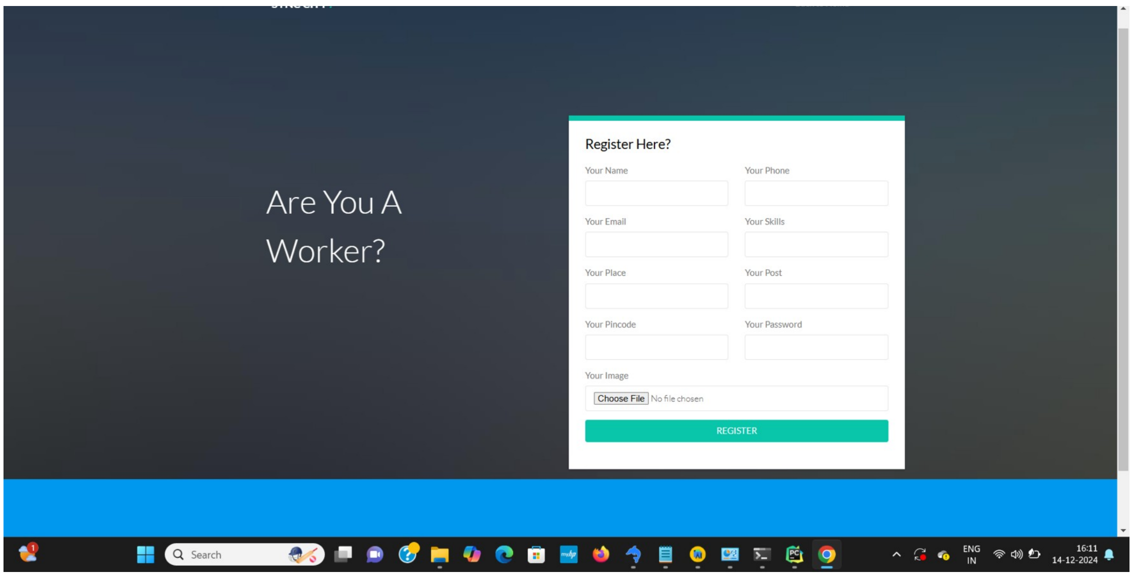
register - hospital



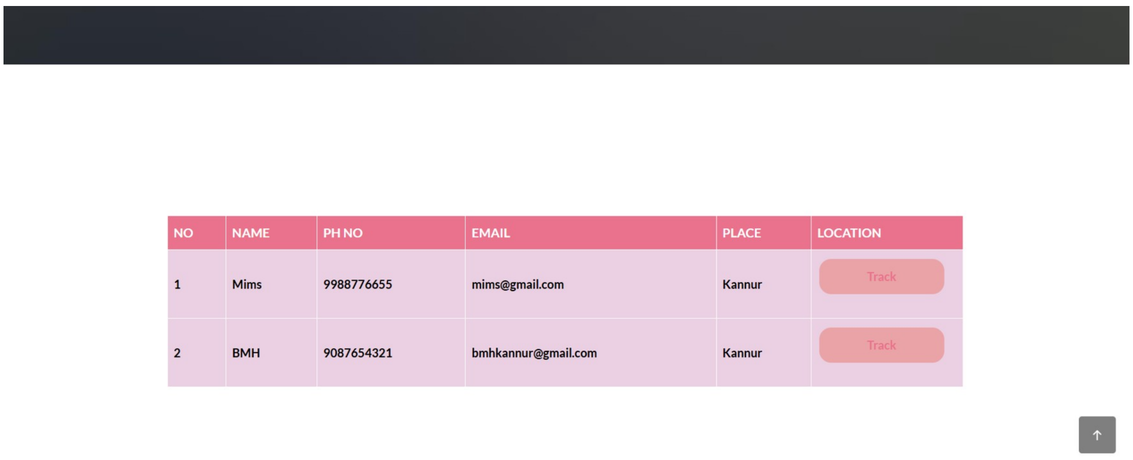
register - driver



register – worker

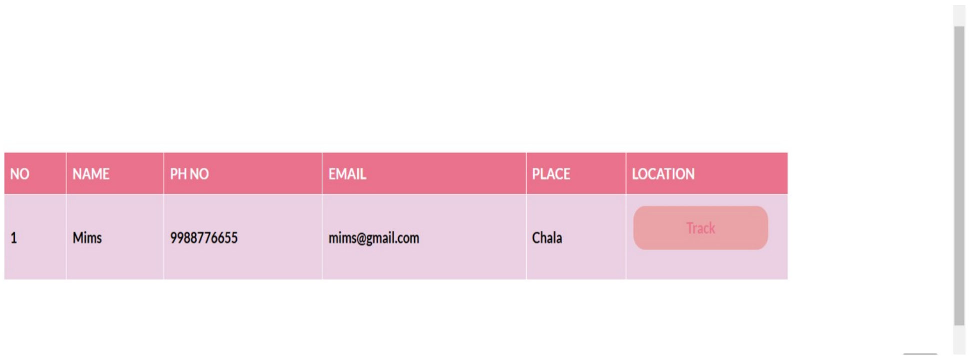


view - hospital



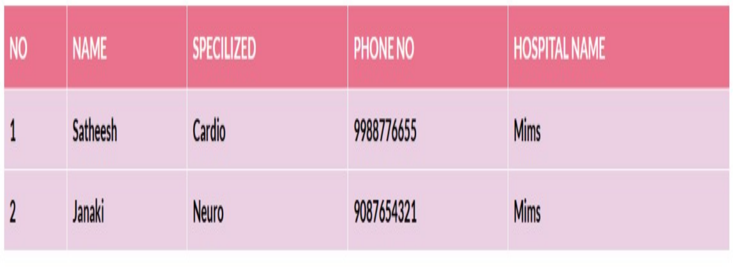
NO	NAME	PH NO	EMAIL	PLACE	LOCATION
1	Mims	9988776655	mims@gmail.com	Kannur	Track
2	BMH	9087654321	bmhkannur@gmail.com	Kannur	Track

approved - hospital



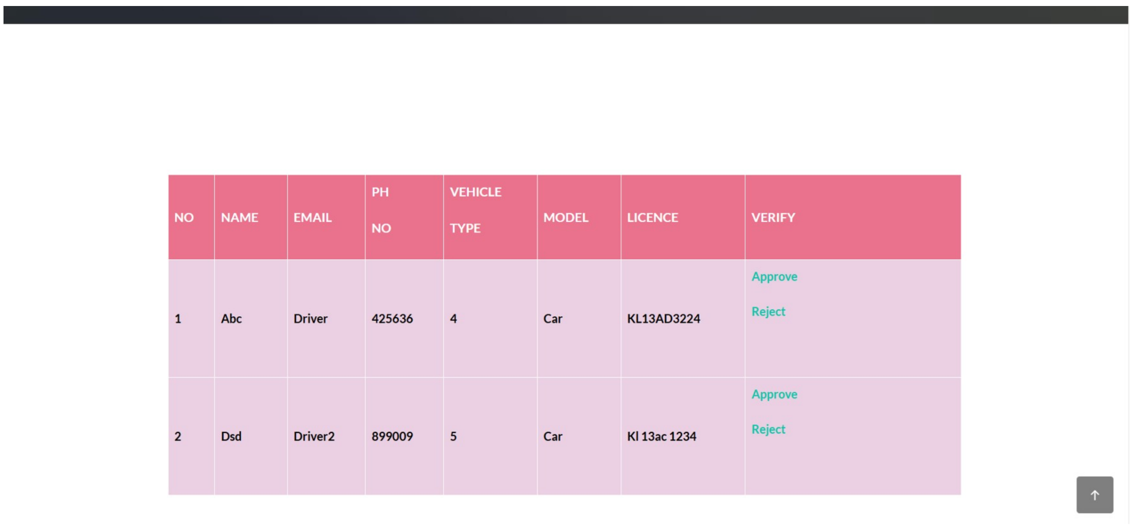
NO	NAME	PH NO	EMAIL	PLACE	LOCATION
1	Mims	9988776655	mims@gmail.com	Chala	Track

views - doctor



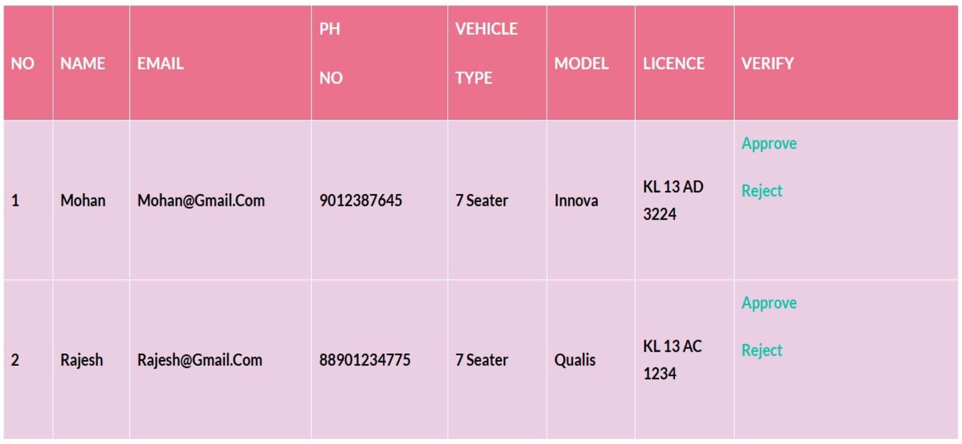
NO	NAME	SPECILIZED	PHONE NO	HOSPITAL NAME
1	Satheesh	Cardio	9988776655	Mims
2	Janaki	Neuro	9087654321	Mims

view – driver



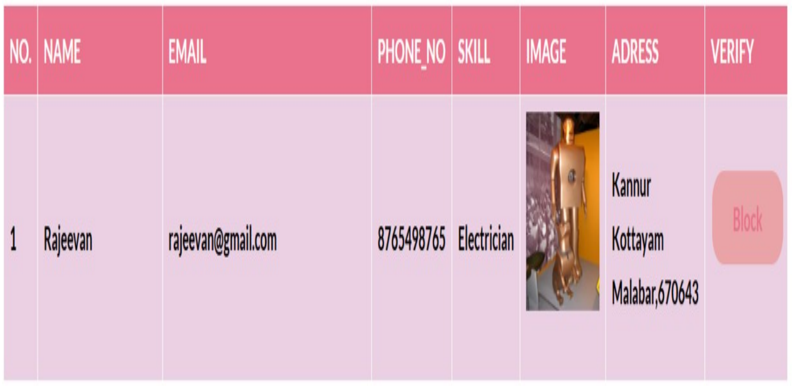
NO	NAME	EMAIL	PH NO	VEHICLE TYPE	MODEL	LICENCE	VERIFY
1	Abc	Driver	425636	4	Car	KL13AD3224	Approve Reject
2	Dsd	Driver2	899009	5	Car	Kl 13ac 1234	Approve Reject

verify - driver



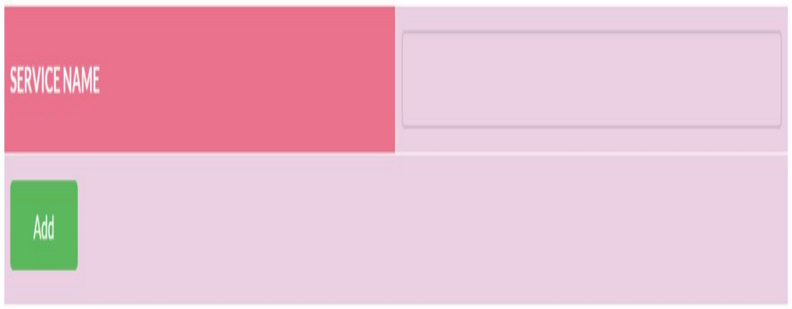
NO	NAME	EMAIL	PH NO	VEHICLE TYPE	MODEL	LICENCE	VERIFY
1	Mohan	Mohan@Gmail.Com	9012387645	7 Seater	Innova	KL 13 AD 3224	Approve Reject
2	Rajesh	Rajesh@Gmail.Com	88901234775	7 Seater	Qualis	KL 13 AC 1234	Approve Reject

approved - worker



NO.	NAME	EMAIL	PHONE_NO	SKILL	IMAGE	ADRESS	VERIFY
1	Rajeevan	rajeevan@gmail.com	8765498765	Electrician		Kannur Kottayam Malabar,670643	Block

add – services


SERVICE NAME
Add