

Software Engineering (CS-301)

Smart Street Lighting System

Project Report

Team 17

Aryan Sharma (BT21GCS161)

Jinesh Chhajer (BT21GCS347)

Ishaan Tiwari (BT21GCS362)

Yash Nair (BT21GCS015)

Ananya Vaish (BT21GCS315)

Table of Contents

1. Requirement Analysis
 - a. Functional Requirements
 - b. Non-Functional Requirements
 - c. Design Requirements
2. Project Methodology
3. Function Point Analysis
4. UML Diagrams
 - a. Activity Diagram
 - b. Use Case Diagram
 - c. ER Diagram
5. Database Schema
6. Development
 - a. Coding Standards
 - b. Code Optimisation
 - c. Resusability
 - d. Modularity
7. Testing
 - a. Black Box Testing
 - b. White Box Testing
 - c. Performance Testing
8. References

1. Requirement Analysis

1.1 Functional Requirement

a. User Authentication:

- The system shall provide a user authentication mechanism on the User Login page.
- Users must be able to input their username and password for authentication.
- The system should validate the entered credentials against stored user information.
- system shall support secure user authentication using industry-standard encryption algorithms.

b. Error Handling:

- If the entered username or password is invalid, the system shall display an error message indicating "Invalid username or password."
- The error message should be prominently displayed, alerting the user to the issue.

c. Registration Link:

- A link to the registration page shall be provided, allowing new users to navigate to the registration section easily.

d. Password Visibility Toggle:

- The system shall provide an option to toggle the visibility of the password in the password input field.
- The password visibility toggle button shall allow users to switch between displaying and hiding the password.

e. Password Strength Indicator:

- The system may include a password strength indicator to guide users in creating strong passwords.

f. Logout Notification:

- If a user has successfully logged out, the system shall display a notification indicating "You have been logged out."

g. Navigation Bar:

- The system shall provide a responsive navigation bar that includes links to various sections of the application.
- The navigation bar shall include links for "About Us," "Car," "Emergency," and "Logout" (visible only when authenticated).

h. Day Buttons:

- The system shall display buttons for each day, allowing users to view information related to each day.
- Each button shall trigger a modal that displays relevant information, including date, start time, end time, location, light status, pole ID, and traffic density.

i. Home Link:

- The "Home" link shall redirect users to the home page when clicked.

j. Search Button:

- A "Search" button shall be provided to initiate the car number search.
- It will compare the entered car number with the matrix data and display relevant information if a match is found.

k. Result Display:

- When a car number is found, the result shall be displayed in a readable format.
- Displayed information shall include the car number, current pole number, previous pole numbers (if applicable), and next pole numbers.

1.2 Non – Functional Requirement:

Non-functional requirements for a Smart Street Lighting system are crucial to ensure that the system meets performance, security, and usability standards. Here are some non-functional requirements for a Smart Street Lighting system

a. Performance:

- Response Time: The system should provide a response time within 5 seconds for routine operations
- Scalability: The system should be scalable to accommodate an increase in the number of user's data volume without a significant degradation in performance.

b. Usability:

- User Interface: The system should have an intuitive and user-friendly interface to facilitate easy navigation
- Responsive Design: The app should be accessible on various devices and screen sizes, maintaining usability across different platforms.

c. Reliability

- Availability: The Webapp should be available 24/7, with minimal downtime for maintenance or update.
- Error Handling: it should handle errors gracefully, providing informative error messages and minimizing data loss in case of issues.

d. Scalability:

- Data Storage: Applications should be able to handle a growing volume of data without compromising performance.
- System should handle an increased number of users or additional features without a significant decrease in performance.

e. Security:

- Data Encryption: All sensitive data, including User details, should be encrypted during transmission and storage.
- Communication between the mobile app, web app, and server should be encrypted

f. Compatibility:

- System Compatibility: The system should be compatible with major Operating Systems (Windows, Linux, MacOS) and major web browsers (Google Chrome, Mozilla Firefox, Microsoft Edge, and Brave) to ensure accessibility for users.
- Integration: The system should support integration with other Smart Street Lighting systems and external databases to facilitate data exchange.

g. Backup and Recovery:

- **Data Backup:** The system should perform regular automated backups of all data, and the backup process should be tested periodically.
- **Disaster Recovery:** The system should have a comprehensive disaster recovery plan in place, ensuring minimal data loss and downtime in case of unexpected events.

1.3 Design Requirements

These design requirements outline the architectural design of the application, serving as a blueprint for the coding process. They delineate the roles of each class, define the relationships between them, and establish their connections to the database tables.

a. User Interface Design (UI):

- user interface should be intuitive and easy to navigate.
- Use a consistent and clean design with a well-structured layout.
- Implement a responsive design to support various devices and screen sizes.

b. Architecture Design:

- overall structure of the application, specifying key components and their interactions.
- Establish the relationship between the frontend, backend, and database components.
- Design a modular system with clear separation of concerns for scalability and maintainability.

c. Performance Design:

- Specify a target response time of within 5 seconds for routine operations.
- Design the system to be scalable, accommodating an increase in user data volume without a significant performance impact.

d. User Experience Design (UX):

- User-Friendly Interface: Creating a user-friendly interface with clear and concise design elements, minimizing complexity and promoting a positive user experience.
- Loading Time Optimization: Optimizing loading times for application pages and features, enhancing user experience by minimizing wait times.

e. Security Design:

- Implement data encryption for sensitive information during transmission and storage.
- Enforce encrypted communication between the web app, and server components.

f. Scalability Design:

- Design data storage to handle a growing volume of data without compromising performance.
- Ensure the system can handle an increased number of users or additional features without a significant decrease in performance.

g. Visual Design:

- Use a visually appealing and calming color scheme. Ensure high contrast for readability and consider the use of color to distinguish different medications or actions.

h. Error handling Design

- Identifying the error, and displaying the pop-up message, which help user to knowing his/her mistake, resolve the error.

2. Project Methodology:

2.1 Characteristics of the Project:

i. Basic Requirement / Well understanding project:

- Understanding the client is, what the client wants to convey, and how the client approaches us is essential. Once we understand the client, it becomes easier to identify the methodology. Overall, this leads to the establishment of a well-defined and stable set of requirements from the beginning

ii. Budget Constraints:

- Demands continuous monitoring and a system of checks and balances throughout its duration. The budget constraints highlight the importance of prudent resource management and risk assessment to ensure the successful and cost-effective implementation of the project.

iii. Time constraints

- Effective timeline management to ensure that the project stays within the client's specified timeframe.

2.2 Choosing Methodology:

Selecting the appropriate methodology for a project is vital, as it establishes the framework and approach for project planning, execution, and management.

The chosen methodology significantly influences the project's success, efficiency, and adaptability to changing circumstances.



Why Agile Methodology-

- If the project requirements are well-defined and unlikely to change significantly, the Waterfall methodology might be suitable for its structured and sequential approach.
- If there is a need for frequent adjustments based on user feedback, evolving requirements, or a desire for continuous improvement, the Agile methodology could be more appropriate
- The project, centered around car movement detection and street light control, could benefit from Agile methodologies due to potential ongoing enhancements and adjustments based on user feedback.

Considerations:

- The size and complexity of the project.
- The level of client involvement and feedback required.
- The project's susceptibility to changes in requirements.
- The team's familiarity and experience with a particular methodology.

2.3 Several factors / Reason for choosing Agile Methodology

1) Flexibility to Changing Requirements:

- The project may face evolving requirements or changes during development.
- Agile handles change efficiently, allowing for flexibility in adapting to new features or modifications.

2) Client Collaboration:

- Client feedback is crucial to ensure that the system aligns with their expectations and needs.

3) Effective Risk Management:

- Enabling the identification and mitigation of risks throughout the development process.

4) Iteration Planning:

- Backlog Refinement: Prioritize and refine the user stories in the product backlog based on feedback and changing requirements.
- Sprint Goals: Define specific goals for each sprint, breaking down user stories into smaller, manageable tasks.

5) Development:

- Integrate code frequently to identify and resolve issues early in the development process.

6) Monitoring and Maintenance:

- User Analytics: Use analytics tools to monitor user interactions and gather insights for further enhancements.
- Bug Tracking: Implement a system for tracking and addressing bugs promptly

7) Quality Assurance:

- Agile includes practices such as continuous testing and integration, ensuring the delivery of a high-quality product.

Why (Analysis) Not Other Methodology?

- Waterfall is a rigid methodology where changes are challenging to incorporate once the project has started. It doesn't adapt well to evolving requirements.
- While the Spiral Model includes risk analysis, Agile's continuous feedback loops and shorter iterations offer more frequent risk assessment and mitigation.
- RAD might face challenges with late changes, and Agile's adaptability is more suitable for evolving requirements.
- Prototyping may focus heavily on upfront design, whereas Agile allows for ongoing refinement and adjustment based on real feedback.

3. Function Point Analysis

External Input: 13

1. Login: 02
 - a. Username
 - b. Password
2. Registration: 07
 - a. First Name
 - b. Last Name
 - c. Password
 - d. Email ID
 - e. State
 - f. Gender
 - g. Date of Birth
3. Emergency Form: 03
 - a. Location
 - b. Type of Emergency
 - c. Description
4. Search Car Detail: 01
 - a. Car Number

External Outputs: 17

1. Sending Notifications to Users: 01
2. Displaying Car location: 03
 - a. Current Pole no.
 - b. Next 3 Pole No.
 - c. Prev 2 Pole No.

3. Display Emergency History: 03

- a. Location
- b. Type of Emergency
- c. Description

4. Light Status: 07

- a. Date
- b. Start Time
- c. End Time
- d. Location
- e. Light Status
- f. Pole no.
- g. Traffic Density

5. About Us: 03

- a. Name
- b. Email ID
- c. Phone No.

External Inquiries: 27

1. Login: 02

- a. Username
- b. Password

2. Registration Form :07

- a. First Name
- b. Last Name
- c. Password
- d. Email ID
- e. State

- f. Gender
- g. Date of Birth

3. User Registration Record: 07

- a. First Name
- b. Last Name
- c. Password
- d. Email ID
- e. State
- f. Gender
- g. Date of Birth

4. Emergency Form: 03

- a. Location
- b. Type of Emergency
- c. Description

5. Search Car Detail: 01

- a. Car Number

6. Light Status: 07

- a. Date
- b. Start Time
- c. End Time
- d. Location
- e. Light Status
- f. Pole no.
- g. Traffic Density

Internal Logical files: 20

1. User Profile Database: 09

a. Login Detail: 02

- a) Username
- b) password

b. User Registration Record: 07

- a) First Name
- b) Last Name
- c) Password
- d) Email ID
- e) State
- f) Gender
- g) Date of Birth

2. Emergency Form Detail Database: 03

- a) Location
- b) Type of Database
- c) Description

3. Light Status Database: 07

- a) Date
- b) Start Time
- c) End Time
- d) Location
- e) Light Status
- f) Pole no.
- g) Traffic Density

4. Search Car Detail: 01
a) Car Number

External Interface files: 07

1. Login
2. Registration
3. Home Page
4. About us
5. Car Location
6. Emergency Form
7. Logout

Functional Point Analysis:

- Number of External Inputs = 13
- Number of External Output = 17
- Number of External Inquiries = 27
- Number of Internal Logical Files = 20
- Number of External Interface Files = 07

Table 1. Unadjusted Function Point (UAF)

Type of Component	Complexity of Components Multiplier Factor			
	Low	Average	High	Total
External Inputs	3	4	6	
External Outputs	4	5	7	
External Inquiries	3	4	6	
Internal Logical Files	7	10	15	
External Interface Files	5	7	10	
Total Number of Unadjusted Function Points				

We are assuming that all complexity adjustment factors (CAF) and weighting factors are average

Scale

0 – No Influence

1 – Incidental

2 – Moderate

3 – Average

4 – Significant

5 – Essential

Step 1:

Hence, we calculate F:

Taking complexity adjustment Factor (CAF) as **Average**

$$F = 14 * Scale$$

$$F = 14 * 3$$

$$\underline{F = 42}$$

Step 2:

Calculate *Complexity Adjustment Factor (CAF)*

$$CAF = 0.65 + (0.01 * F)$$

$$CAF = 0.65 + (0.01 * 42)$$

$$CAF = 0.65 + 0.42$$

$$\underline{CAF = 1.07}$$

Step 3:

Calculate UFP (*Unadjusted Function Point*):

External Inputs: = 04

External Outputs: = 05

External Inquiries: = 04

Internal Logical files: = 10

External Interface Files: = 07

$$\text{UFP} = (13 * 04) + (17 * 05) + (27 * 04) + (20 * 10) + (07 * 07)$$

$$\text{UFP} = 52 + 68 + 108 + 200 + 49$$

$$\underline{\text{UFP} = 477}$$

Step 4:

Calculating the *functional Point*

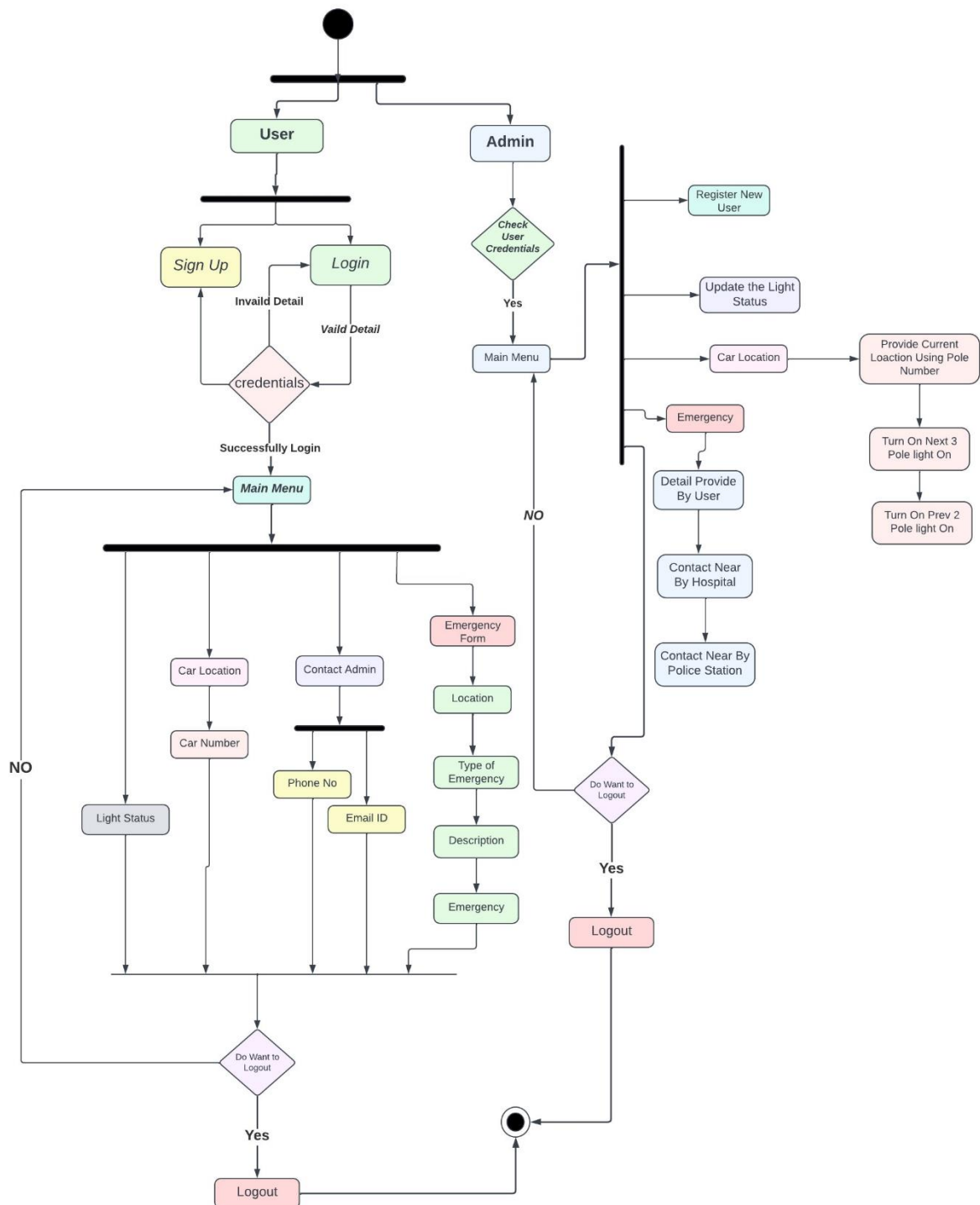
$$\text{FP} = \text{UFP} * \text{CAF}$$

$$\text{FP} = 477 * 1.07$$

$$\underline{\text{FP} = 512.39}$$

4. UML Diagram

4.1 Activity Diagram



Starts with a user either signing up for a new account or logging into their existing account. If the user's credentials are valid, they will be logged in and taken to the main menu. If the user's credentials are invalid, they will be prompted to try again.

Upon successful authentication, the system proceeds to display the main menu.

From the main menu, the user can choose-

- Get light status.
- Get the current location of their car.
- Contact the admin.
- Emergency
- Log out.

If the user has an emergency, they can click on the "Emergency" button. This will open a form where the user can enter their Location, Description of the emergency. The form will also ask the user to specify the type of emergency (medical, etc.). Once the user has submitted the form, their information will be sent to the nearest hospital or police station.

If the user selects the "Get the current location of their car" option, it will ask to enter the car number, then system will use the pole number to determine the current location of the car. The system will then provide the user with the car's location.

If the user selects the "Contact the admin" option, the system will provide the user with the contact information for the admin (Phone No, Email ID).

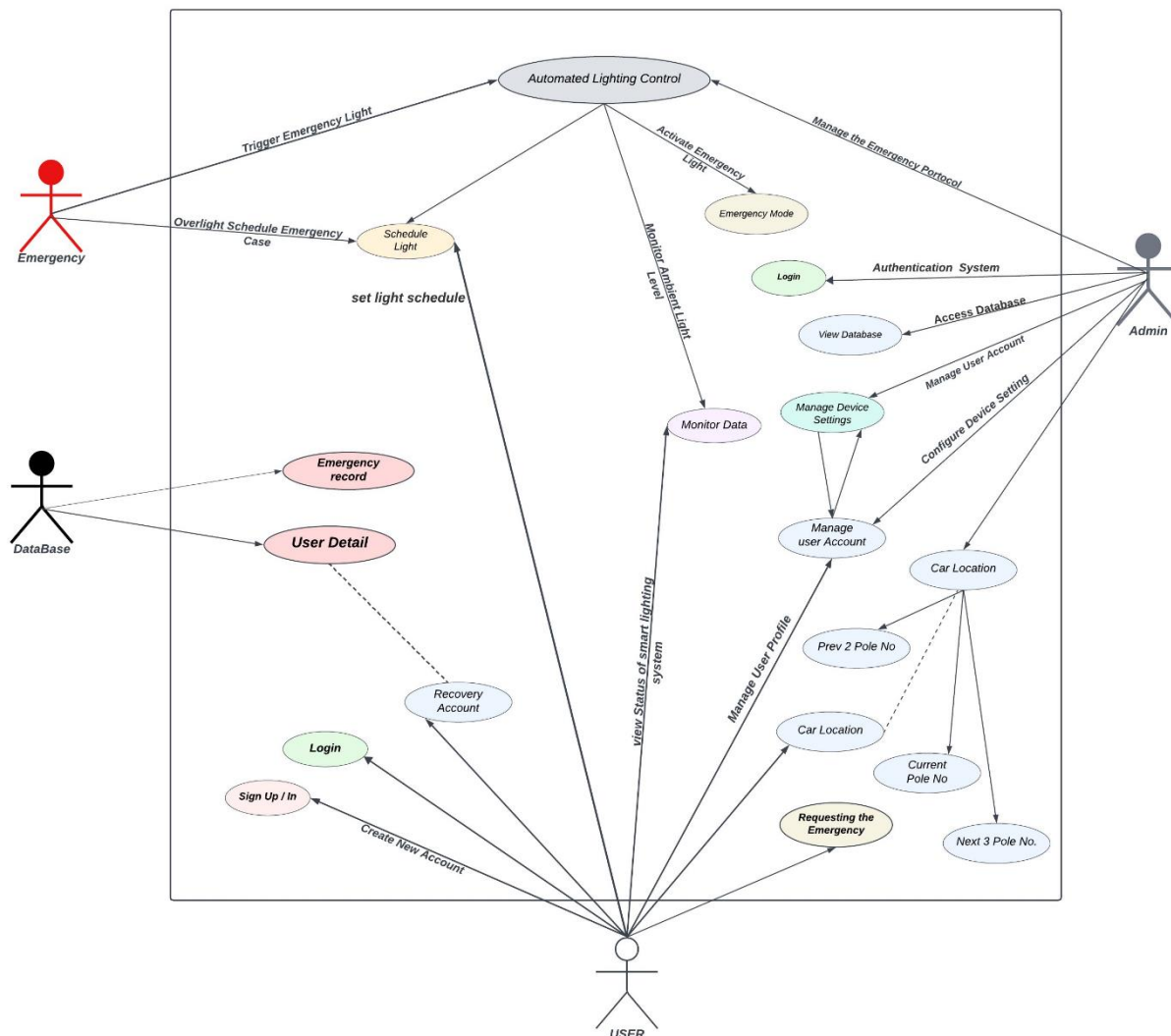
Here is a more detailed explanation of each step in the flowchart:

- Sign Up: The user enters their personal information and creates a password.
- Login: The user enters their email address and password (system authenticates them).
- Check User Credentials: The system checks the user's credentials against its database.

- **Successfully Login:** The user is logged in and taken to the main menu.
- **Main Menu:** The user can view their car location, light status, Car Location, contact admin, Emergency and log out.
- **Light Status:** The user can able to see Traffic Density, start time and End time, along with Date
- **Emergency Form:** The user enters location, and a description of the emergency. The form also asks the user to specify the type of emergency (medical, etc).
- **Contact Admin:** The user can send a message to the admin.
- **Get the current location of their car:** It will ask user to enter the car number, then system will use the pole number to determine the current location of the car. The system will then provide the user with the car's location.
- **Logout:** The user is logged out of their account.

Overall, the activity diagram shows a well-designed process for managing user accounts in a system. The process is efficient and easy to follow, and it allows users to perform all of the necessary tasks related to managing their accounts.

4.2 Use Case Diagram

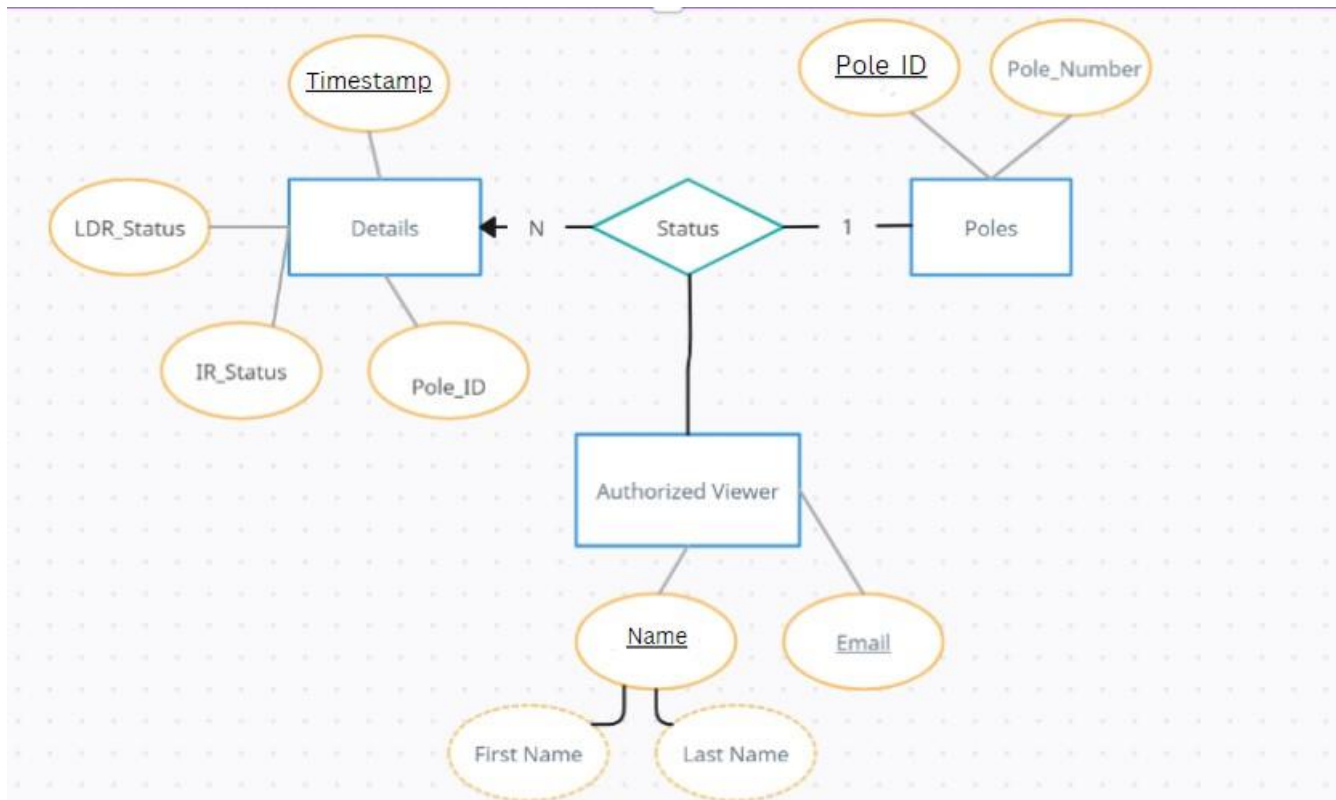


It shows that the user can interact with the system to set light schedules, view data, manage device settings, and view the status of the system. The User can create new user accounts, manage user accounts. The emergency actor can trigger emergency lighting in the event of an emergency. It shows the interactions between the user and the system, as well as the system functions.

Actors in the system are:

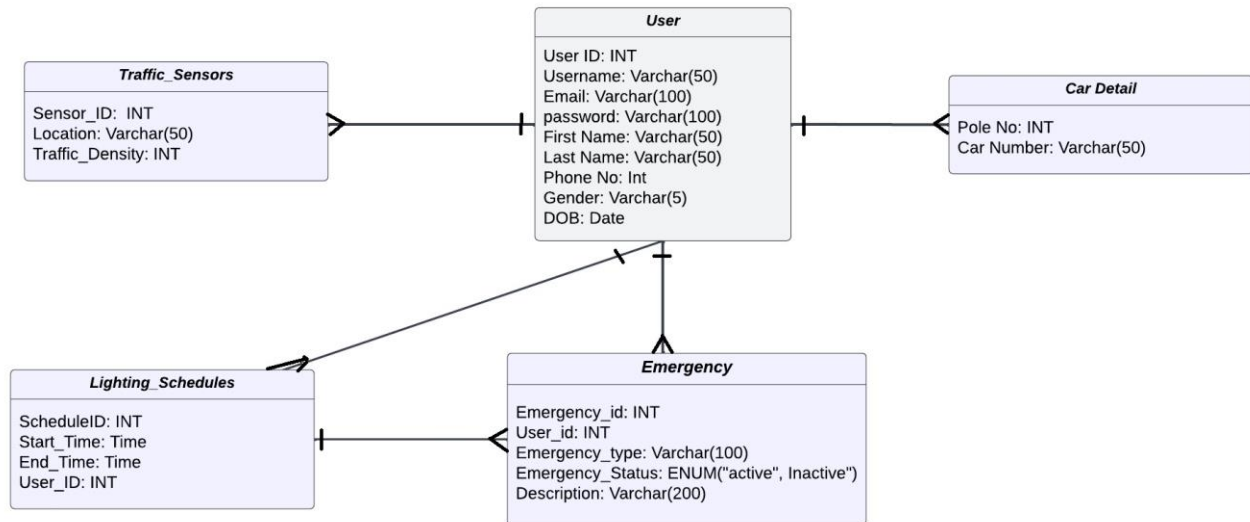
- User Actor
- Admin Actor
- Emergency Mode Control Actor
- Database Manager Actor

4.3 ER Diagram



It checks the LDR status and IR status. If the LDR status is "On" and the IR status is "Off", the system then displays the pole number and the name and email of the authorized viewer system then checks the LDR status and IR status. If the LDR status is "On" and the IR status is "Off", the system then displays the pole number and the name and email of the authorized viewer.

5.Database Design Schema



Users Table:

Purpose: Manages information about users interacting with the system.

Importance: Essential for user authentication, access control, and associating users with specific lights they control.

Lighting_Schedules Table:

Purpose: Stores data related to individual street and highway lights.

Importance: Critical for controlling and monitoring the status, brightness of lights. Also associates each light with a specific user.

Emergency Table:

purpose: Stores information about the emergency events that have been triggered by the devices

Importance: Facilitates quick identification and classification of emergency types. Enables logging and monitoring of emergency events for rapid response.

Traffic_Sensors Table:

Purpose: Records data from sensors monitoring traffic density.

Importance: Enables the system to dynamically respond to variations in traffic flow, optimizing lighting for safety and energy efficiency.

Car Detail Table:

Purpose: Record the car number to retrieve the current location of the car.

Importance: In case of a stolen vehicle, recording car numbers allows law enforcement to track the vehicle's movement and locate it quickly

A) User Table:

User-related information and authentication data

Column Name	Data Type	Description
<i>user_id</i>	<i>INT</i>	<i>Unique identifier for the user (primary key)</i>
<i>username</i>	<i>VARCHAR (50)</i>	<i>Username for the user</i>
<i>email</i>	<i>VARCHAR (100)</i>	<i>Email address for the user</i>
<i>password</i>	<i>VARCHAR (255)</i>	<i>Password for the user</i>
<i>First_Name</i>	<i>VARCHAR (50)</i>	<i>User First Name</i>
<i>Last_Name</i>	<i>VARCHAR (50)</i>	<i>Last Name of User</i>
<i>Gender</i>	<i>VARCHAR (5)</i>	<i>Male, Female, NA</i>
<i>DOB</i>	<i>DATE</i>	<i>User Date Of Birth</i>

B) *Lighting_Schedule Table:*

Column Name	Data Type	Description
<i>schedule_id</i>	<i>INT</i>	<i>Unique identifier for the lighting schedule (primary key)</i>
<i>user_id</i>	<i>INT</i>	<i>Foreign key to the User table</i>
<i>start_time</i>	<i>TIME</i>	<i>Time when the lighting schedule starts</i>
<i>end_time</i>	<i>TIME</i>	<i>Time when the lighting schedule ends</i>

C) *Emergency_Event table:*

Column Name	Data Type	Description
<i>emergency_id</i>	<i>INT</i>	<i>Unique identifier for the emergency event (primary key)</i>
<i>user_id</i>	<i>INT</i>	<i>Foreign key to the user table, referencing the device triggered by the emergency</i>
<i>emergency_type</i>	<i>Varchar (50)</i>	<i>Type of emergency (fire, etc.)</i>
<i>emergency_status</i>	<i>Varchar (50)</i>	<i>Status of the emergency (active or inactive)</i>
<i>Description</i>	<i>Varchar (50)</i>	<i>Description</i>

D) *Traffic_Sensors Table:*

Column Name	Data Type	Description
<i>sensor_id</i>	<i>INT</i>	<i>Unique identifier for the traffic sensor (primary key)</i>
<i>location</i>	<i>VARCHAR (50)</i>	<i>Location of the traffic sensor</i>
<i>traffic_density</i>	<i>INT</i>	<i>Measured traffic density</i>

E) Car Detail Table:

<i>Column Name</i>	<i>Data Type</i>	<i>Description</i>
<i>Car Number</i>	<i>VARCHAR (50)</i>	<i>Unique identifier for the Car Detail (primary key)</i>
<i>Pole Number</i>	<i>INT</i>	<i>Location of the Pole Number</i>

IN Terminal:

Step 1:

```
File Edit View Terminal Tabs Help
[oracle@demo ~]$ lsnrctl start

LSNRCTL for Linux: Version 11.2.0.1.0 - Production on 17-NOV-2023 22:08:37

Copyright (c) 1991, 2009, Oracle. All rights reserved.

TNS-01106: Listener using listener name LISTENER has already been started
```

Step 2:

```
[oracle@demo ~]$ sqlplus "/as sysdba"

SQL*Plus: Release 11.2.0.1.0 Production on Fri Nov 17 22:08:52 2023

Copyright (c) 1982, 2009, Oracle. All rights reserved.

Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production
With the Partitioning, Oracle Label Security, OLAP, Data Mining,
Oracle Database Vault and Real Application Testing options
```

Step 3:

```
SQL> create user SW identified by SW1;
```

User created.

Step 4:

```
SQL> grant connect, resource, dba to SW;
```

Grant succeeded.

After this open sql_developer and establish a connection

New / Select Database Connection

Connection Name	Connection Details
ETL3	ETL3@//localhost:1521/niit
MDAMAC	MDAMAC@//localhost:1521/niit
MPS	MPS@//localhost:1521/niit
MPS-2	SCD2@//localhost:1521/niit
MPS-3	SCD3@//localhost:1521/niit
niit_ecommerce	ecommerce@//localhost:1521/niit
niit_ETL1	ETL1@//localhost:1521/niit
niit_ETL2	ETL2@//localhost:1521/niit
niit_ETL3	ETL3@//localhost:1521/niit
niit_zilean	zilean@//localhost:1521/niit
SLGTS	SW@//localhost:1521/niit

Connection Name: SLGTS
Username: SW
Password: ***
☒ Save Password

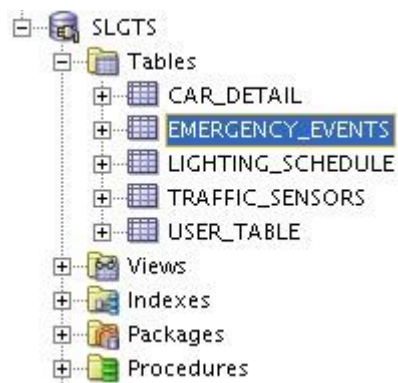
Oracle
Role: default
Connection Type: Basic
☐ OS Authentication
☐ Kerberos Authentication
☐ Proxy Connection

Hostname: localhost
Port: 1521
☒ SID: niit
☐ Service name

Status: Success

Help Save Clear Test Connect Cancel

ALL THE TABLE IN AN INSTANCE ARE LISTED BELOW



User Table:

Column Name	Data Type	Nullable	Data Default	COLUMN ID	Primary Key	COMMENTS
USER_ID	NUMBER	No	(null)	1	1 (null)	
USERNAME	VARCHAR2(50 BYTE)	No	(null)	2	(null) (null)	
EMAIL_ID	VARCHAR2(255 BYTE)	No	(null)	3	(null) (null)	
PASSWORD	VARCHAR2(255 BYTE)	No	(null)	4	(null) (null)	
FIRST_NAME	VARCHAR2(50 BYTE)	No	(null)	5	(null) (null)	
LAST_NAME	VARCHAR2(50 BYTE)	No	(null)	6	(null) (null)	
GENDER	VARCHAR2(5 BYTE)	No	(null)	7	(null) (null)	
DOB	DATE	No	(null)	8	(null) (null)	

USER_ID	USERNAME	EMAIL_ID	PASSWORD	FIRST_NAME	LAST_NAME	GENDER	DOB
1	1 Zilean	aryan@ma...	2580@Aryan	Aryan	Sharma	M	19-JAN-04
2	2 Raja	raja.p@ma...	alohomora	Raja	Pandey	M	14-AUG-04

Emergency_Events Table:

Column Name	Data Type	Nullable	Data Default	COLUMN ID	Primary Key	COMMENTS
EMERGENCY_ID	NUMBER	No	(null)	1	1 (null)	
USER_ID	NUMBER	No	(null)	2	(null) (null)	
EMERGENCY_TYPE	VARCHAR2(50 BYTE)	No	(null)	3	(null) (null)	
EMERGENCY_ST...	VARCHAR2(50 BYTE)	No	(null)	4	(null) (null)	
DESCRIPTION	VARCHAR2(50 BYTE)	No	(null)	5	(null) (null)	

EMERGENCY_ID	USER_ID	EMERGENCY_TYPE	EMERGENCY_STATUS	DESCRIPTION
1	1	5 Car accident	InActive	OverSpeed
2	2	6 Car Accident	Active	Hit and Run

Traffic_Sensors Table:

Column Name	Data Type	Nullable	Data Default	COLUMN ID	Primary Key	COMMENTS
SENSOR_ID	NUMBER	No	(null)	1	1 (null)	
LOCATION	VARCHAR2(50 BYTE)	No	(null)	2	(null) (null)	
TRAFFIC_DENSITY	VARCHAR2(20 BYTE)	No	(null)	3	(null) (null)	

SENSOR_ID	LOCATION	TRAFFIC_DENSITY
1	201 Neemrana	Low
2	202 Thane	High

Lighting_Schedule Table

Column Name	Data Type	Nullable	Data Default	COLUMN ID	Primary Key	COMMENTS
SCHEDULE_ID	NUMBER	No	(null)	1	1 (null)	
USER_ID	NUMBER	No	(null)	2	(null) (null)	
START_TIME	VARCHAR2(20 BYTE)	No	(null)	3	(null) (null)	
END_TIME	VARCHAR2(20 BYTE)	No	(null)	4	(null) (null)	

SCHEDULE_ID	USER_ID	START_TIME	END_TIME
1	160	12:30pm	05:30pm
2	161	27:30pm	8:45pm

Car Detail Table:

Column Name	Data Type	Nullable	Data Default	COLUMN ID	Primary Key	COMMENTS
CAR NUMBER	VARCHAR2(50 BYTE)	No	(null)	1	1 (null)	
POLE NUMBER	NUMBER	No	(null)	2	(null) (null)	

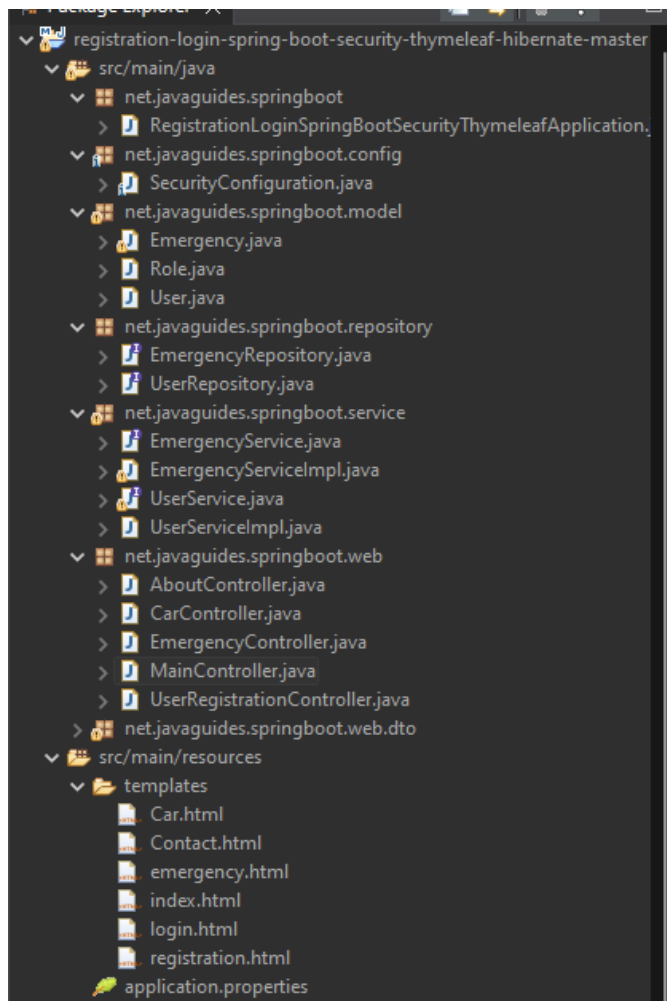
CAR NUMBER	POLE NUMBER
1 MH05EN3312	16
2 HR26DQ551	7

6.Development:

To implement a module of our program following the general coding standards of Java, maintaining modularity, and ensuring the module's reusability, the first step was to review all files of our application. This involved analyzing the relevant coding standards and identifying where they were being followed in our program.

6.1) Coding Standards:

The subsequent coding standards and conventions are crucial to ensure that your code is readable, maintainable, and consistent. While different organizations and teams may have their own coding standards, the following represent some widely accepted Java coding standards and best practices.



The project is structured with a "Layer First" folder structure in Eclipse, wherein data model, data repository/provider, and main pages/UI are separated.

Maintaining coding standards enhances code readability and maintainability, ensuring a consistent approach across the entire project.

2. Naming Conventions:

Class names (User): Follows the CamelCase

Variable names: Follows the CamelCase convention

Package name (net.javaguides.springboot.model): Follows the reverse domain naming convention.

```
package net.javaguides.springboot.model;
```

```
public User(String firstName, String lastName, String email, String password, Collection<Role> roles) {  
    super();  
    this.firstName = firstName;  
    this.lastName = lastName;  
    this.email = email;  
    this.password = password;  
    this.roles = roles;  
}
```

3. Comments:

Commenting is essential to elucidate the roles of classes, methods, and critical code segments. Ensure a uniform commenting style, such as Javadoc for documenting classes and methods, while steering clear of superfluous or repetitive comments.

```
// Primary key for the User entity
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private Long id;

// First name of the user
@Column(name = "first_name")
private String firstName;

// Last name of the user
@Column(name = "last_name")
private String lastName;

private String email; // Unique email address for the user

private String password; // Password for user authentication

// Many-to-Many relationship between User and Role entities
@ManyToMany(fetch = FetchType.EAGER, cascade = CascadeType.ALL)
@JoinTable(
    name = "users_roles", // Name of the association table
    joinColumns = @JoinColumn(
        name = "user_id", referencedColumnName = "id"), // Column mapping for User entity
    inverseJoinColumns = @JoinColumn(
        name = "role_id", referencedColumnName = "id")) // Column mapping for Role entity

private Collection<Role> roles;

// Default constructor required by JPA
public User() {
```

4.Import:

Arrange import statements systematically and eliminate any unused imports. Specific imports instead of wildcard imports to clearly indicate the classes being utilized.

5.Keeping Length of The Functions Small:

```
@Override
public User save(UserRegistrationDto registrationDto) {
    User user = new User(registrationDto.getFirstName(),
        registrationDto.getLastName(), registrationDto.getEmail(),
        passwordEncoder.encode(registrationDto.getPassword()), Arrays.asList(new Role("ROLE_USER")));

    return userRepository.save(user);
}
```

6.Code Reusability:

Encapsulate and modularize code to enhance reusability and maintainability.

7. Brace Placement:

The brace placement is consistent and follows the Java standard, placing opening braces on the same line as the statement. Place closing braces on their own line.

8. Whitespace:

Enhance code readability, Add spaces around operators and after commas.

```
// Method to save a new user based on registration data
@Override
public User save(UserRegistrationDto registrationDto) {
    // Creating a new User instance with encoded password and default role
    User user = new User(
        registrationDto.getFirstName(),
        registrationDto.getLastName(),
        registrationDto.getEmail(),
        passwordEncoder.encode(registrationDto.getPassword()),
        Arrays.asList(new Role("ROLE_USER"))
    );

    // Saving the user in the repository
    return userRepository.save(user);
}

// Method to load user details for authentication
@Override
public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
    // Retrieving user from the repository based on the provided username (email)
    User user = userRepository.findByEmail(username);
    if (user == null) {
        // Throwing an exception if the user is not found
        throw new UsernameNotFoundException("Invalid username or password.");
    }
    // Creating UserDetails using Spring Security's User class
    return new org.springframework.security.core.userdetails.User(
        user.getEmail(),
        user.getPassword(),
        mapRolesToAuthorities(user.getRoles())
    );
}
```

6.2) Code Optimisation

1) Constructor Injection:

Used constructor injection for the UserRepository and BCryptPasswordEncoder. Constructor injection is considered a best practice as it ensures that dependencies are injected at the time of object creation, promoting better testability and reducing coupling.

```
// Constructor injection of the UserRepository
// Injecting UserRepository dependency using constructor
public UserServiceImpl(UserRepository userRepository, BCryptPasswordEncoder passwordEncoder) {
    this.userRepository = userRepository;
    this.passwordEncoder = passwordEncoder;
}
```

2) Consistent Comments:

Improved and added comments for better readability. Comments should focus on explaining the why and how rather than what the code is doing

```
// Method to save a new user based on registration data
// Saves a new user using the provided registration data
@Override
public User save(UserRegistrationDto registrationDto) {
    User user = new User(
        registrationDto.getFirstName(),
        registrationDto.getLastName(),
        registrationDto.getEmail(),
        passwordEncoder.encode(registrationDto.getPassword()),
        Arrays.asList(new Role("ROLE_USER"))
    );

    return userRepository.save(user);
}
```

3) Error Message Enhancement:

error message enhancement is to provide more specific information about the nature of the error, making it easier for developers to identify and troubleshoot issues.

```
@Override
public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
    // Attempt to find a user in the repository based on the provided email (username)
    User user = userRepository.findByEmail(username);

    // If no user is found, throw a UsernameNotFoundException with a specific error message
    if (user == null) {
        throw new UsernameNotFoundException("User with email '" + username + "' not found.");
    }

    // Create and return a UserDetails object for the authenticated user
    return new org.springframework.security.core.userdetails.User(
        user.getEmail(),
        user.getPassword(),
        mapRolesToAuthorities(user.getRoles())
    );
}
```

6.3) Reusability:

The application upholds the principle of reusable code by ensuring that the code is generalized and not hard-coded. This approach allows the code to be easily utilized in other projects or applications.

UserService interface defines a set of methods related to user management, such as saving a user (save method) and retrieving a user by email (getUserByEmail method).

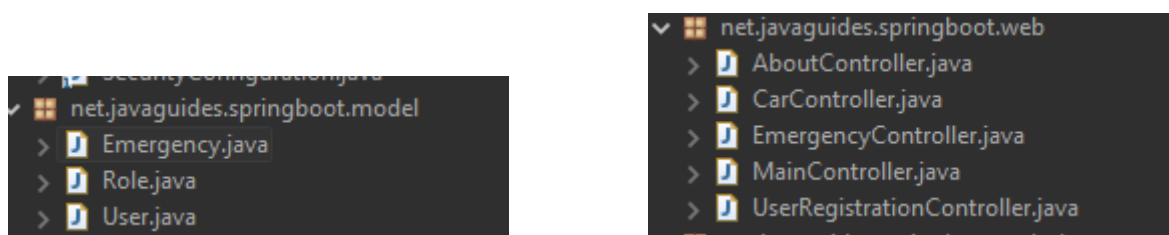
```
@Service
public interface UserService extends UserDetailsService {
    User save(UserRegistrationDto registrationDto);

    User getUserByEmail(String email);
}
```

6.4) Modularity

To implement modularity, each module should possess its own function, with each class containing only relevant and related modules. Our application fully adheres to this modularity principle, as modules are created for every major action in the software and are appropriately separated within their respective classes.

Broke down the system into smaller, manageable modules, showcasing design decisions that contribute to optimal and maintainable code.



7. Testing

7.1 BlackBox Testing –

Black-box testing evaluates the functionality of a software application without requiring knowledge of its internal code, structure, or implementation details.

Key Characteristics of BlackBox Testing:

1) No Knowledge of Internal Code:

- Testers performing black box testing do not have access to the source code or knowledge of the internal workings of the system being tested.
- Testing process is conducted from an external perspective, simulating how end-users interact with the software.

2) Focus on Functionalities:

- Testers evaluate whether the software performs as expected based on the given requirements and specifications without delving into the internal logic or implementation details.

3) User's Perspective:

- Goal is to ensure that the system behaves correctly and meets user expectations, regardless of the underlying code structure.
- Testers approach black box testing from the perspective of end-users or external entities interacting with the software.

4) Test Cases Based on Requirement:

- Test cases for black box testing are derived from the software's requirements and specifications.
- Testers design test scenarios that cover various functionalities to verify that the system meets the specified criteria and performs as intended.

5) Test Cases Cover Different Scenarios:

- Black box testing involves creating test cases that cover a range of scenarios, including normal operation, boundary conditions, error conditions, and other relevant use cases.

BlackBox Testing Techniques:

1) Functional Testing:

- Ensure that the software functions according to the specified requirements.
- Evaluate the software's features and capabilities to verify that it performs as expected

2) Non-Functional Testing:

- Assess the non-functional aspects of the system, such as performance, reliability, usability, and scalability.

3) Regression Testing:

- Ensure that new changes or updates do not negatively impact existing functionalities

4) System Testing:

- Validate the entire system as a whole to ensure all components work together seamlessly.

5) Acceptance Testing:

- Confirm that the software meets user and business requirements and is ready for deployment.

6) Compatibility Testing:

- Ensure that the software functions correctly across different environments, devices, and configurations

Functional Testing:

Case 1: Register a New Account

Input:

- First Name: Aman
- Last Name: Sharma
- Email ID: aman.21@gmail.com
- Password: 123@Aman

Registration

First Name

Last Name

Email

Password

[Register](#) [Already registered? Login here](#)

Expected Output:
Successfully Registration

You've successfully registered to our awesome app!

Case 1.1: Register a New Account (Wrong Email ID)

Input:

- First Name: Soham
- Last Name: patil
- Email ID: sohamp.in
- Password: 123@Patil

Expected Output:
Invalid Email ID

Email

Please enter a valid Gmail address.

Case 1.2: Register a New Account (Wrong Password)

Input:

- First Name: Soham
- Last Name: patil
- Email ID: sohamp@gmail.in
- Password: 123@atil

Expected Output:

Invalid Password

Password

Please enter a valid password. It must be at least 6 characters long and contain at least 1 number, 1 lowercase alphabet, 1 uppercase alphabet, and 1 special character (@, #, \$, %, &).

Case 1.3: Register a New Account (Invalid Email ID and Password)

Input:

- First Name: Soham
- Last Name: patil
- Email ID: sohamp.in
- Password: 123@atil

Expected Output:

Invalid Detail

Email

Please enter a valid Gmail address.

Password

Please enter a valid password. It must be at least 6 characters long and contain at least 1 number, 1 lowercase alphabet, 1 uppercase alphabet, and 1 special character (@, #, \$, %, &).

Case 1.4: Register a New Account (Email Id Already)

Input:

- First Name: Soham
- Last Name: Patil
- Email ID: sohamp@gmail.com
- Password: 123@Soham

Expected Output:

Already Email ID Exists

Email address already exists

Case 1.5: Register a New Account (Complete Detail Not Fill)

Input:

- First Name: Soham
- Last Name: Patil
- Email ID: null
- Password: 123@Soham

Expected Output:

Please fill complete Detail

Email

Password



Please fill out this field.



Case 2.1: Login Detail (Invalid Login Detail)

Input:

- Username: Soham@gmail.com
- Password: 123@Soham

Expected Output:

Account Not Found

Invalid username or password.

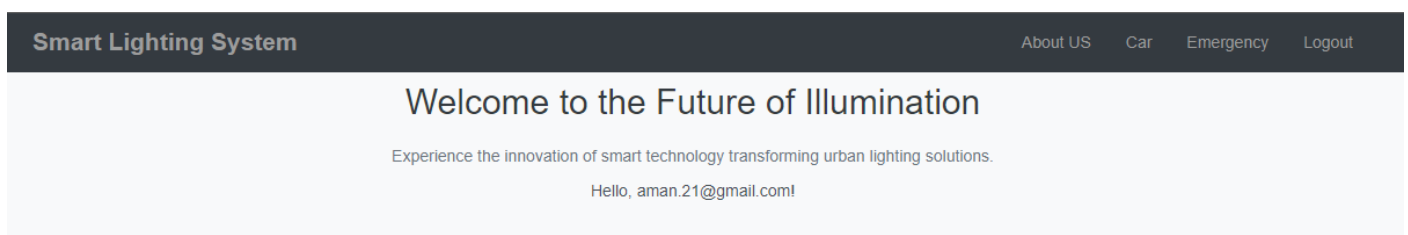
Case 2.2: Login Detail (Successfully Login)

Input:

- Username: aman.21@gmail.com
- Password: 123@Aman

Expected Output:

Successfully Login



Case 3.1: Light Status: (Day 1)

Input:

Clicking on the Day 1 button.

Expected Output:

Light Detail such as Start Time, End Time, Date,
Traffic Density or Any kind of emergency

Day 1 Information

Date: 15-Nov-2021

Start Time: 12:30 pm

End Time: 12:50 pm

Location: Neemrana

Light Status: On

Pole Id: 10

Traffic Density: Low

Emergency Id: 01

Emergency Type: NA

Emergency Status: Inactive

Close

Case 3.2: Light Status: (Day 4)

Input:

Clicking on the Day 4 button.

Expected Output:

Light Detail such as Start Time, End Time, Date,
Traffic Density or Any kind of emergency

Day 4 Information

Date: 16-Nov-2021
Start Time: 10:30 pm
End Time: 11:50 pm
Location: Neemrana
Light Status: ON
Pole Id: 10
Traffic Density: Medium

Emergency Id: 04
Emergency Type: Car Accident
Emergency Status: Active

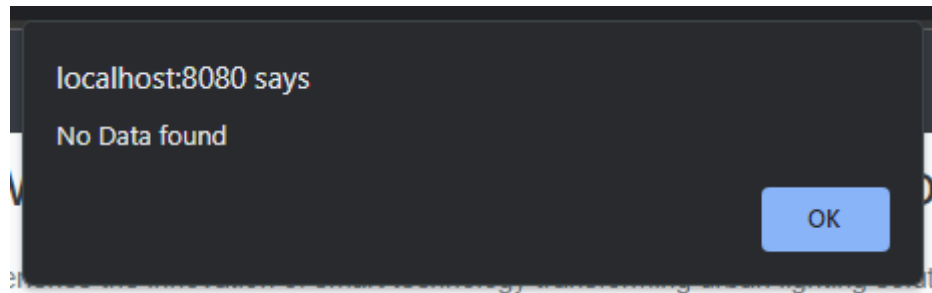
Close

Case 3.3: Light Status: (Day 5/6)

Input:

Clicking on the Day 5/6 button.

Expected Output:
No Detail Found



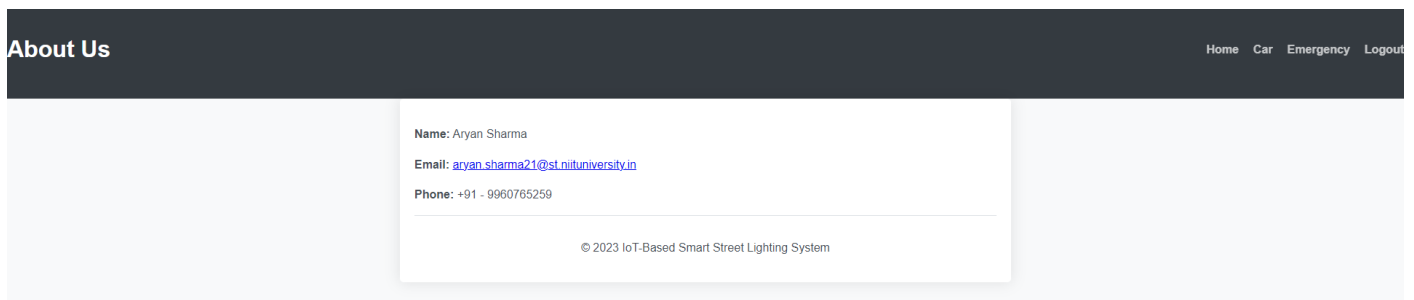
Case 4: About Us:

Input:
Clicking on the About Us Tab

Expected Output:

Admin Team Contact Detail

- Admin Name
- Phone No
- Email Id



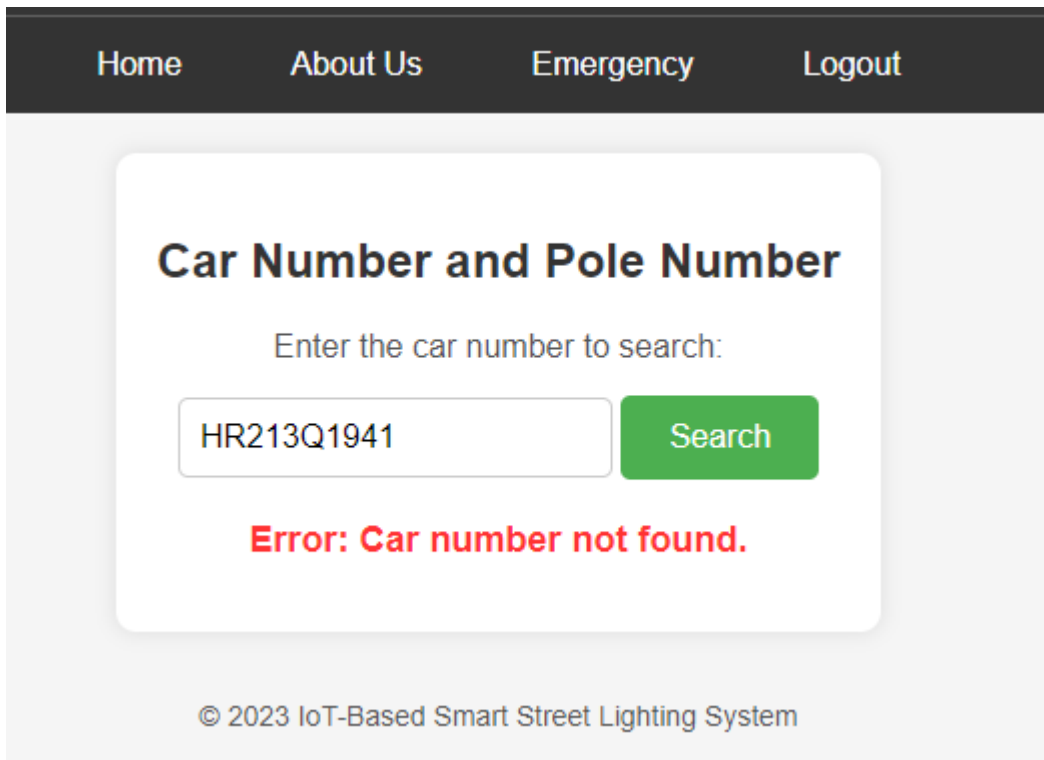
Case 4.1: Car Detail (Detail Not Found)

Input:

- User Car Number (Car license plate): HR213Q1941

Expected Output:

Detail Not Found



The screenshot displays a web application interface for searching car details. At the top, there is a dark navigation bar with links for 'Home', 'About Us', 'Emergency', and 'Logout'. The main content area features a white card titled 'Car Number and Pole Number'. Inside the card, there is a prompt 'Enter the car number to search:' followed by a text input field containing 'HR213Q1941' and a green 'Search' button. Below the input field, a red error message states 'Error: Car number not found.' At the bottom of the page, a copyright notice reads '© 2023 IoT-Based Smart Street Lighting System'.

Case 4.2: Car Detail (Detail Found)

Input:

- User Car Number (Car license plate): MH12DE1433

Expected Output:

Detail Found

[Home](#)[About Us](#)[Emergency](#)[Logout](#)

Car Number and Pole Number

Enter the car number to search:

Car Number: MH12DE1433

Current Pole Number: 6

Previous Pole Numbers: 5, 4

Next Pole Numbers: 7, 8, 9

© 2023 IoT-Based Smart Street Lighting System

Case 5: Emergency Form

Input:

- Description:
emergency car accident
1 - Adult Death on side
2 - child injured
- Emergency Type:
Car Accident
- Location:
Neemrana

Expected Output:

Detail Save and contact nearby Hospital and Police Station

Emergency submitted successfully!

Case 6: Logout Account

Input:

Clicking on the Logout Tab

Expected Output:

Successfully Logout Account

You have been logged out.

7.2 White Box Testing –

In white-box testing, the tester has knowledge of the internal code and logic of the software being tested. This knowledge is used to design test cases that ensure all internal components and code paths are exercised.

For White-box testing, our strategy was to implement it during the development phase of the project to save time and ensure the proper working and execution of the code.

developer performs manual testing to ensure that every required line of code is executed for a task, all conditions are handled, and a function or method produces the desired output.

Key aspects of white box testing include:

1) Code-Level Testing

- White box testing is conducted at the code level, focusing on individual functions, statements, and paths within the software

2) Understanding Internal Logic

- Testers need knowledge of the internal workings of the software, including the source code, algorithms, and data structures. This knowledge is essential for designing effective test cases.

7.3 Performance Testing

Due to a shortage of hardware resources, we were unable to employ proper means for conducting performance testing strategies such as load testing. Consequently, we made the best use of the available resources, and the results are further elaborated upon.

Additionally, the code for the application is well-optimized, good coding practices and exhibiting low execution times.

The responsivity and speed of the application are influenced by the network speeds of the user and the latency of the local server, factors that were beyond our control during the testing process.

8. References

1. Agile Process 101: Understanding the Benefits of Using Agile Methodology
<https://www.nvisia.com/insights/agile-methodology>
2. Study of Various Process Model in Software Development
https://www.researchgate.net/publication/260632268_Comparative_Study_of_Various_Process_Model_in_Software_Development
3. Function Point Analysis
<https://www.semanticscholar.org/paper/Function-Point-Analysis-for-Software-Maintenance-Hira-Boehm/eb2b4eeaccf52725c51efae5648fc08cc0946a7#citing-papers>
4. Black Box and White Box Testing
<https://www.spiceworks.com/tech/devops/articles/black-box-vs-white-box-testing/>
5. Black Box Testing
<https://www.browserstack.com/guide/black-box-testing#:~:text=White%20box%20testing%20aims%20to,from%20an%20end%2Duser%20perspective.>