E-PARKING CHALLAN GENERATION SYSTEM

SUBMITTED BY:

NAME : Dhanasree L P

DEPT & SEC : CSE 'A' (IInd year)

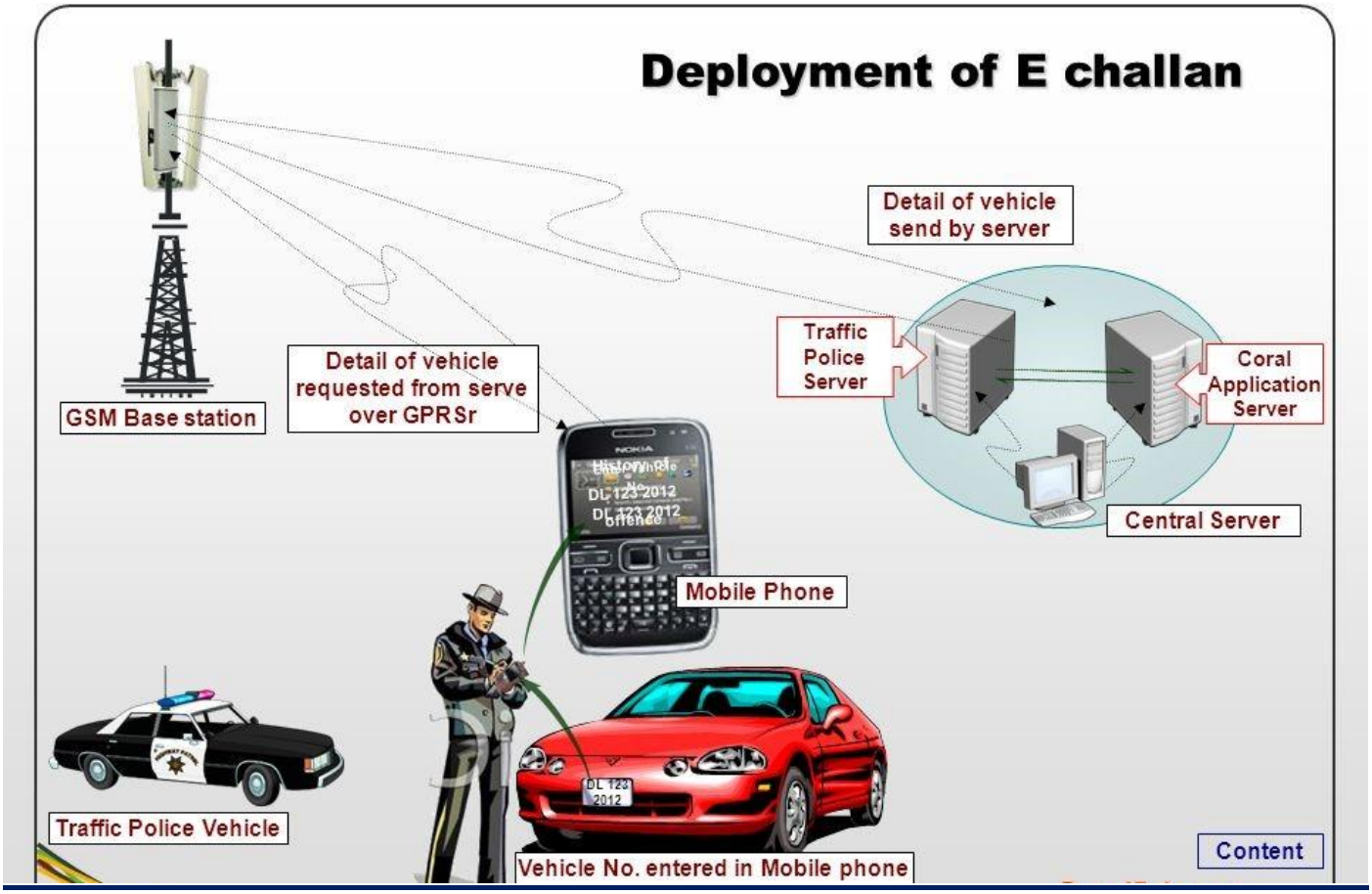
ROLL_NO : 2116220701061

SOFTWARE ENGINEERING CONCEPTS – LAB MANUAL

Table of Contents

OVERVIEW OF THE PROJECT

The e-parking challan generation system revolutionizes the traditional role of police officers in traffic management and enforcement. With this system, officers gain access to advanced tools such as License Plate Recognition (LPR) technology and streamlined digital interfaces. They can swiftly identify parking violations, issue digital challans, and enforce regulations with greater efficiency and accuracy. Real-time access to vehicle registration details and violation records empowers officers to make informed decisions and take timely enforcement actions. The system's notification features enable officers to communicate effectively with vehicle owners, informing them of issued challans and payment deadlines. By embracing digitalization and automation, officers can optimize their duties, minimize paperwork, and focus on proactive measures to improve traffic flow and public safety. Ultimately, the e-parking challan generation system empowers police officers to uphold regulations, enhance urban mobility, and create safer environments for communities to thrive.



SOFTWARE ENGINEERING CONCEPTS – LAB MANUAL

EX_NO: 1 **DATE: 20-02-24**

SOFTWARE REQUIREMENTS SPECIFICATION(SRS)

TABLE OF CONTENTS:

SOFTWARE ENGINEERING CONCEPTS – LAB MANUAL

1.INTRODUCTION

1.1 About project:

- The e-parking challan generation project aims to slove the process of issuing parking fines. This includes capturing vehicle details and issuing the challan electronically. The valuable time of every working professional is important and thus it cannot be interrupted due to a mere traffic congestion. The project enhances the customer experience, reduces paperwork and improves productivity.

1.2 Scope:

- The scope of an e-parking challan generation project would typically involve developing a website to automate the process of issuing challans for parking violation tickets. This would include capturing vehicle information, recording the violation, generating the challan, and sending it to the violator. The system may also include a payment gateway for fine collection and a database for record-keeping.

1.3 Features:

- Efficiency: This project saves time for law enforcement officers and reduces the likelihood of errors in issuing tickets.
- Accuracy: There is less chance of human error in recording vehicle information or violation details, leading to more accurate records and fewer disputes.
- Speed: Faster processing times can lead to more efficient enforcement and improved traffic management.

SOFTWARE ENGINEERING CONCEPTS – LAB MANUAL

2.PROJECT REQUIREMENTS

2.1 General Requirements:

1. User Interface (UI):

- User-friendly interface for both admins and vehicle owners.which makes giving Input for vehicle details, violation information, and payment easier.

2. Database:

- helps record violations and corresponding penalties.

3. License Plate Recognition (LPR):

- Implementing LPR technology cameras for vehicle identification.

4. Challan Generation:

- Generation of parking challans with violation details and penalties.

5. Payment:

- Online payment gateway for vehicle owners to pay fines.

6. Notification System:

- Email or SMS notifications for vehicle owners with challan details.

7. Responsiveness:

- Ensure the system works on various devices.

8. Legal Compliance:

- Align the system with local parking regulations and privacy laws.

9. Maintenance Plan:

- Establish a plan for regular updates, bug fixes, and system maintenance.

2.2References:

https://www.researchgate.net/publication/351315376_E_Challan_System_for_Non-Parking_Areas

We have referred to the "E-Challan System for Non-Parking Areas" published on ResearchGate for insights and methodologies relevant to the development of our project.

3.FUNCTIONAL REQUIREMENTS

3.1 Registration:

- **User Registration:** Users should be able to register on the platform using their personal details.
- **Vehicle Registration:** Users should be able to register their vehicles on the platfrom along with vehicle details.

3.2 Challan Generation:

- The system must be able to generate a challan automatically when a violation of parking rules takes place.

3.3 Payment:

- **Payment Processing:** Users should pay the challan online through the platform using various payment methods credit/debit cards, upi, net banking, etc.)
- **Notification:** Users should receive notifications about the challan, payment status and any updates.

3.4 Enforcement:

- **Enforcement Interface:** Officers should have access to an interface to input violations manually and view violation details.

4.Non-functional Requirements

4.1 Performance:

- The system can generate parking challans quickly and easily.
- Scalability: The system will be able to handle a large volume of parking challans as the number of users increases.
- Reliable: The system can ensure that parking challans are generated accurately.

4.2 Usage:

- The system will be user-friendly to access the challans easily.
- Compliance: The system will comply with relevant laws and regulations related to parking enforcement and data privacy.

4.3 Availability:

- The system will be available 24/7 to generate parking challans for users.

4.4 Cost:

- The website will be cost-effective to develop and maintain.

EX_NO: 2 **DATE: 01-03-24**

AGILE – SCRUM METHODOLOGY

Product Backlog:

❖ **Sprint 1 Backlog**

1) **User Authentication**
   - **User Story:** As a user, I want to securely register and log in to the e-parking system so that I can access my account.
   - **Tasks:**
     - Implement user registration form with email and password.
     - Develop email verification feature.
     - Implement login functionality.
     - Create password recovery feature.

2) **Vehicle Registration**
   - **User Story:** As a user, I want to register my vehicle so that it can be recognized by the system.
   - **Tasks:**
     - Create a vehicle registration form.
     - Validate vehicle details to prevent duplicates.
     - Allow users to edit and delete vehicle details.

3) **Basic UI Setup**
   - **User Story:** As a user, I want a user-friendly interface so that I can easily navigate the system.
   - **Tasks:**
     - Set up the basic layout and navigation.
     - Design login, registration, and dashboard screens.

❖ **Sprint 2 Backlog**

4) **Challan Generation**
   - **User Story:** As a parking officer, I want to generate a challan for a vehicle violating parking rules so that the violation is recorded and a fine is issued.

- **Tasks:**
  - Create a form to generate challans with violation details.
  - Store challan data in the database.

5) Challan Notification

- **User Story:** As a vehicle owner, I want to receive notifications when a challan is issued so that I am aware of any fines.
- **Tasks:**
  - Implement email notification for issued challans.
  - Include a link to view the challan details online.

6) User Profile

- **User Story:** As a user, I want to view and manage my profile and vehicle information so that my details are up-to-date.
- **Tasks:**
  - Create a profile page.
  - Display user and vehicle information.
  - Allow users to update their profile details.

❖ **Sprint 3 Backlog**

7) Role-Based Access Control

- **User Story:** As an administrator, I want to manage user roles and permissions so that only authorized personnel can perform certain actions.
- **Tasks:**
  - Implement role-based access control.
  - Assign roles (e.g., admin, parking officer, user) to users.
  - Set permissions for each role.

8) Payment Integration

- **User Story:** As a vehicle owner, I want to pay my challan online so that I can conveniently settle fines.
- **Tasks:**
  - Integrate a payment gateway.
  - Implement payment processing for challans.
  - Generate receipts for successful payments.

9) Testing and Bug Fixes

- **User Story:** As a developer, I want to ensure the system is reliable and bug-free so that users have a smooth experience.
- **Tasks:**
  - Conduct unit and integration testing.
  - Perform user acceptance testing.
  - Fix any identified bugs.

❖ **Sprint 4 Backlog (Optional)**

10) Admin Dashboard

- **User Story:** As an admin, I want a dashboard to monitor system activity and manage users so that I can efficiently oversee the system.
- **Tasks:**
  - Develop an admin dashboard.
  - Display user statistics and system logs.
  - Implement user management features.

11) Localization

- **User Story:** As a user, I want to use the system in my preferred language so that I can understand all features and instructions.
- **Tasks:**
  - Implement multi-language support.
  - Allow users to select their preferred language.
  - Translate key UI elements and notifications.

12) Performance Optimization

- **User Story:** As a user, I want the system to be fast and responsive so that I have a smooth experience using it.
- **Tasks:**
  - Optimize database queries.
  - Improve load times for key pages.
  - Conduct performance testing.

Non-Functional Requirements Backlog

1) **High Availability**
   - **User Story:** As a user, I want the system to be available 99.9% of the time so that I can access it whenever needed.
   - **Tasks:**
     - Implement failover mechanisms.
     - Set up monitoring and alerting.

2) **Performance**
   - **User Story:** As a parking officer, I want the system to generate a challan within 2 seconds so that I can efficiently issue multiple challans.
   - **Tasks:**
     - Optimize backend processes.
     - Ensure fast database transactions.

3) **Usability**
   - **User Story:** As a user, I want a clean and intuitive interface so that I can easily navigate the system.
   - **Tasks:**
     - Conduct usability testing.
     - Refine UI based on user feedback.

4) **Security**
   - **User Story:** As a user, I want my personal and vehicle information to be encrypted so that my data is protected.
   - **Tasks:**
     - Implement data encryption.
     - Set up secure authentication mechanisms.

5) **Compliance**
   - **User Story:** As a regulatory authority, I need the system to comply with local data protection laws so that users' privacy is ensured.
   - **Tasks:**
     - Conduct regular compliance audits.
     - Implement necessary legal and regulatory measures.

SOFTWARE ENGINEERING CONCEPTS – LAB MANUAL

EX_NO: 3 **DATE: 12-03-24**

USER STORIES

USER STORY - 1: As a Traffic Officer, **I want to** issue a parking challan to a vehicle parked in a no-parking zone **so that**, I can enforce parking regulations and maintain traffic order.

ACCEPTANCE CRITERIA:

- The system allows the officer to input vehicle details.
- The system captures the location and time of the violation.
- The challan is generated and saved in the system.

USER STORY - 2: As a Vehicle Owner, **I want to** view my past parking challans **so that,** I can keep track of my parking violations and payments.

ACCEPTANCE CRITERIA:

- The system provides a secure login for vehicle owners.
- Owners can view a list of all issued challans associated with their vehicles.
- Each challan displays details such as date, time, location, violation type and fine amount.
- Owners can filter challans by date range and violation type.

USER STORY - 3: As a Vehicle Owner **I want to** pay my parking challan online **so that** I can settle my fines conveniently and avoid additional penalties.

ACCEPTANCE CRITERIA:

- The system provides multiple payment options.
- Owners can select a particular challan and proceed for payment.
- The system confirms the payment and updates the challan status to “Paid”.
- A receipt is generated and sent to the owner’s registered email or phone number.

USER STORY - 4: As an Admin, **I want to** manage user and vehicle information **so that** I can ensure the accuracy of challan generation and user notifications.

ACCEPTANCE CRITERIA:

- The system should allow admin to add, update, or delete user and vehicle information.
- Each vehicle should be associated with the correct user.
- The system should validate vehicle registration details against a central database.

USER STORY - 5: As a Vehicle Owner, **I want to** receive notifications when a challan is issued to my vehicle **so that** I am aware of the violation and can take timely action.

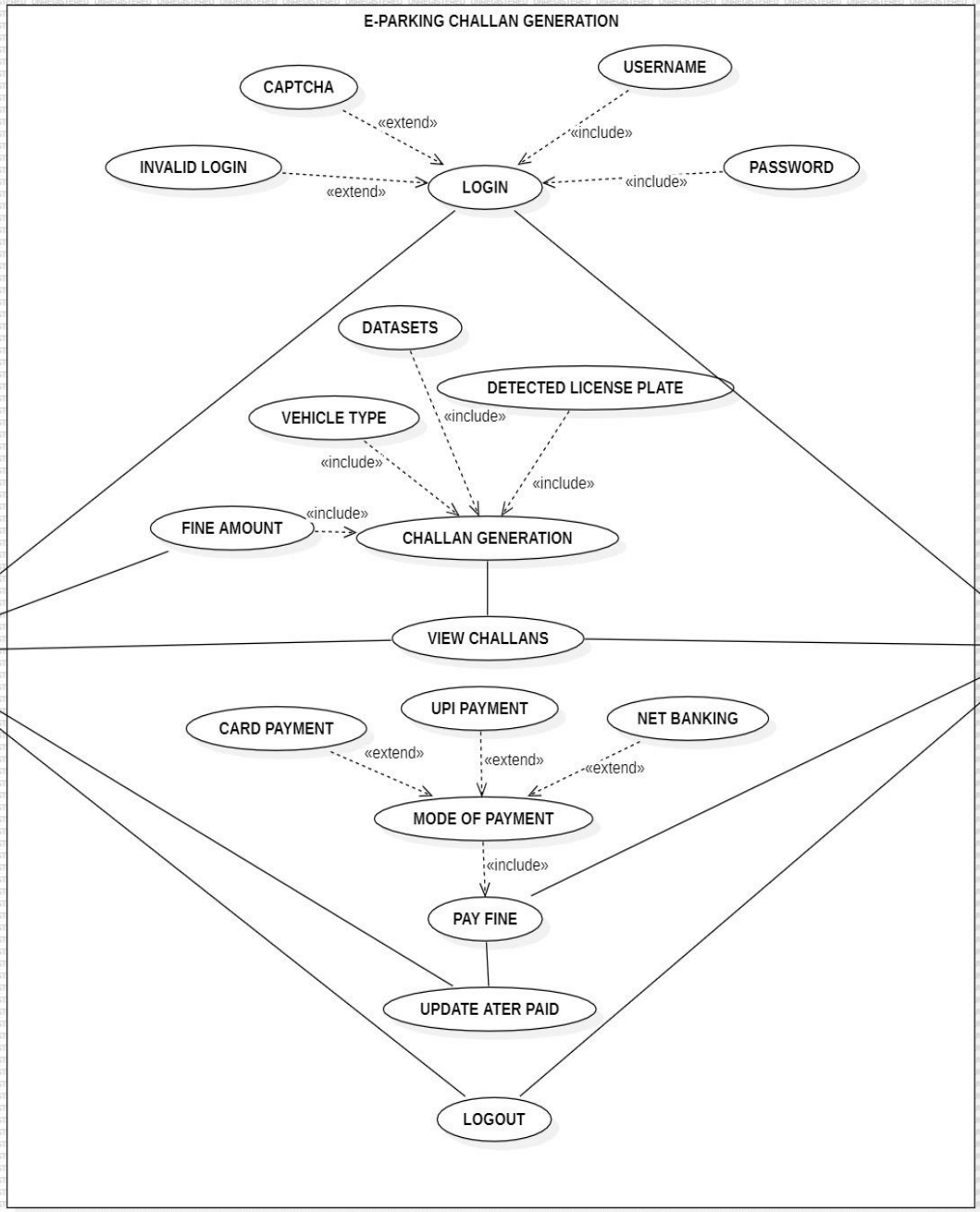
ACCEPTANCE CRITERIA:

- The system sends notifications via SMS and email immediately after a challan is issued.
- Notifications include challan details such as violation type, fine amount, and payment due date.
- Owners can access the detailed challan information through a link provided in the notification.

SOFTWARE ENGINEERING CONCEPTS – LAB MANUAL

EX_NO: 4 **DATE: 19-03-24**

USECASE DIAGRAM



❖ **Actors**

- **Admin**: This actor is responsible for managing the system and user data.
- **Violator:** This actor represents the person who has received a parking ticket.

❖ **Use Cases**

- **Login:** This use case allows a user to login to the system using a username and password. It includes an extension for handling invalid login attempts and another extension for capturing a CAPTCHA code.
- **View Challans:** This use case allows the violator to view a list of parking tickets they have received.
- **Pay Fine:** This use case allows the violator to pay their parking ticket using various methods including UPI, card payment, and net banking. This use case also includes an extension for updating the system after a fine is paid.
- **Logout:** This use case allows a user to log out of the system.

❖ **Relationships**

- **Include:** This relationship signifies that the use case it points to is included as a mandatory step in the use case it originates from. For instance, the "View Challans" use case likely includes an "Login" use case, meaning a user must log in before viewing their parking tickets.
- **Extend:** This relationship signifies that the use case it points to adds optional functionality to the use case it originates from. For instance, the "Login" use case can be extended by a "Capture CAPTCHA" use case, which would only be required if the login attempt fails initial requirements.

EX_NO: 5 **DATE: 29-03-24**

NON-FUNCTIONAL REQUIREMENTS (NFR)

1. Performance and Scalability:

- **Response Time:** The system must generate and deliver a challan within 2 seconds of detecting a parking violation.
- **Scalability:** The system should handle up to 10,000 concurrent users without performance degradation, accommodating peak times and future growth.

2. Security:

- **Data Protection:** All personal and vehicular data must be encrypted using industry-standard encryption techniques both at rest and in transit.
- **Access Control:** The system must implement role-based access control(RBAC) to ensure only authorized personnel can access sensitive information and perform administrative functions.

3. Availability and Reliability:

- **Uptime:** The system should have an uptime of 99.9%, ensuring minimal downtime.
- **Backup and Recovery:** Regular backups must be taken every 12 hours, and a disaster recover plan should be in place to restore services within 1hour of any major failure.

4. Usability:

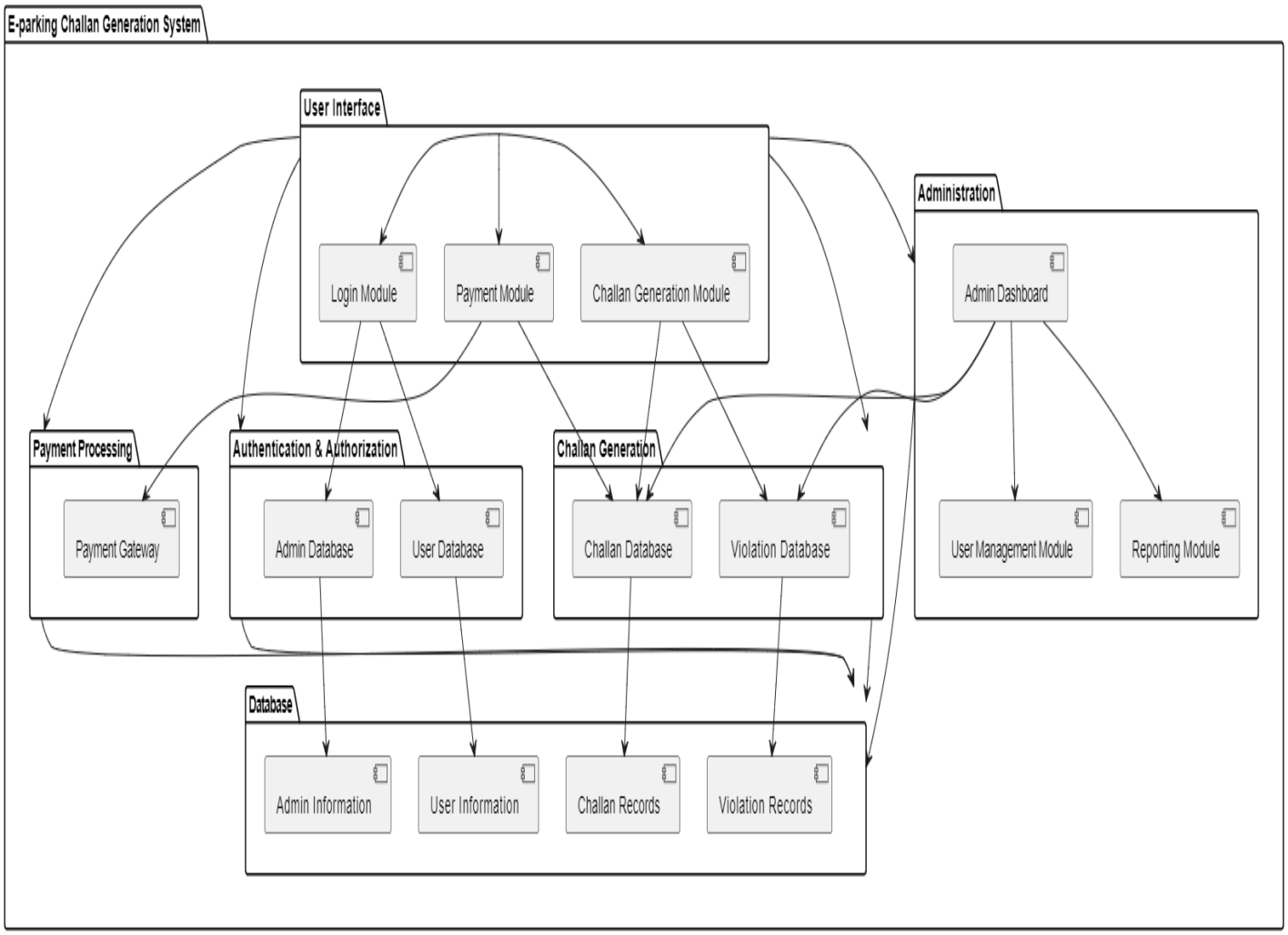
- **User Interface:** The system interface must be intuitive and user-friendly, allowing users to navigate and perform tasks with minimal training.
- **Accessibility:** The system should comply with accessibility standards such as WCAG 2.1, ensuring it is usable by people with disabilities.

5. Maintainability:

- **Code Quality:** The system should follow coding standards and best practices to ensure code readability and maintainability.
- **Documentation:** Comprehensive documentation should be provided for all system components, including user manuals, API documentation, and developer guides to facilitate easy maintenance and upgrades.

EX_NO: 6 **DATE: 09-04-24**

COMPONENTS-BASED OVERALL PROJECT ARCHITECTURE



❖ **User Interface (UI)**

- **Description:** This package contains modules that interact directly with the users of the system.
- **Components:**
  - **Login Module**: Manages user authentication and authorization.
  - **Challan Generation Module:** Allows users to generate challans based on parking violations.
  - **Payment Module:** Facilitates payment of the generated challans.

❖ **Authentication & Authorization**

- **Description:** This package handles the login process and ensures that users and admins are properly authenticated and authorized.
- **Components:**
  - **User Database:** Stores user credentials and information.
  - **Admin Database:** Stores administrator credentials and information.
- Interaction: The Login Module within the UI interacts with both User and Admin Databases to verify credentials.

❖ **Challan Generation**

- **Description:** This package is responsible for creating and managing challans based on parking violations.
- **Components:**
  - **Violation Database:** Stores records of parking violations.
  - **Challan Database:** Stores details of the generated challans.
- **Interaction:** The Challan Generation Module within the UI accesses both the Violation and Challan Databases to create new challans.

❖ **Payment Processing**

- **Description:** This package deals with the financial transactions related to challan payments.
- **Components:**
  - **Payment Gateway:** External service for processing payments.

- **Challan Database:** Accessed to update the status of paid challans.

- **Interaction**: The Payment Module within the UI communicates with the Payment Gateway for transaction processing and updates the Challan Database.

❖ **Administration**

- **Description:** This package provides administrative functionalities, allowing administrators to manage violations, challans, users, and generate reports.
- **Components:**
  - **Admin Dashboard:** Central interface for administrators to manage the system.
  - **User Management Module:** Manages user accounts and roles.
  - **Reporting Module:** Generates various reports related to violations and payments.
  - **Violation Database:** Accessed for managing and reviewing violation records.
  - **Challan Database:** Accessed for reviewing and updating challan records.
- **Interaction:** The Admin Dashboard interacts with various databases and modules to perform administrative tasks.

❖ **Database**

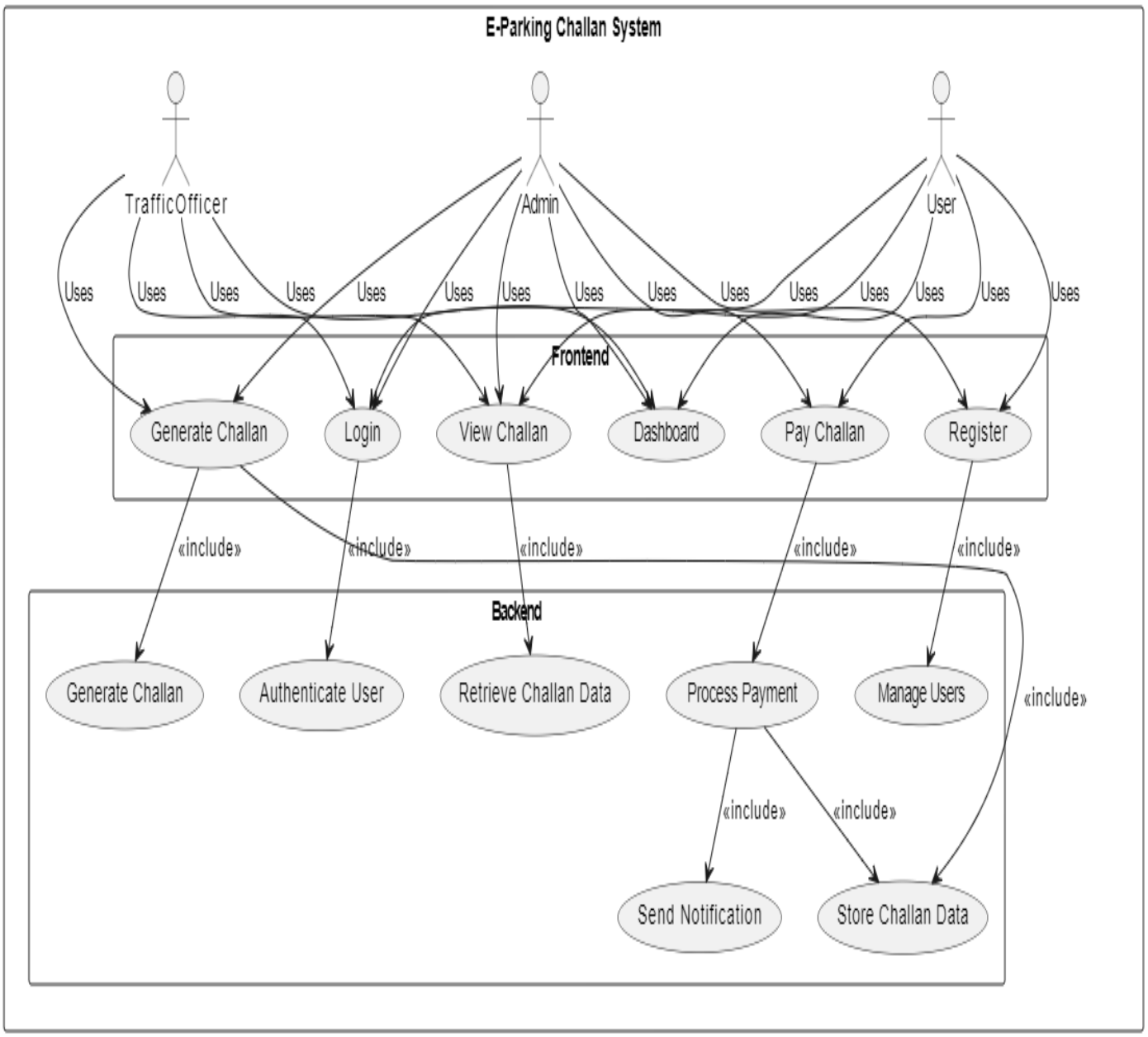
- **Description:** This package encompasses all the databases used by the system, containing essential data for its operation.
- **Components:**
  - **User Database:** Stores user-related information.
  - **Admin Database:** Stores administrator-related information.
  - **Violation Database:** Contains records of parking violations.
  - **Challan Database:** Contains records of issued challans.
- **Interaction:** All other packages (Authentication & Authorization, Challan Generation, Payment Processing, and Administration) interact with the Database package to store and retrieve necessary data.

❖ **Inter-Package Interactions**

- **User Interface:** Acts as the primary access point, interfacing with all core packages: Authentication & Authorization, Challan Generation, Payment Processing, and Administration.
- **Authentication & Authorization:** Interfaces with the Database to verify user/admin credentials.
- **Challan Generation:** Interfaces with the Database to generate and store challans.
- **Payment Processing:** Interfaces with the Payment Gateway for transactions and updates the Challan Database accordingly.
- **Administration:** Interfaces with multiple databases and modules for comprehensive system management.

EX_NO: 7 **DATE: 19-04-24**

BUSINESS ARCHITECTURE DIAGRAM



❖ **Actors:**

- **Admin:** The administrator who manages users, views all challans, and oversees the system.
- **End User:** The user who interacts with the system to view and pay challans.
- **Traffic Officer:** The officer who generates challans for parking violations.

❖ **Frontend Use Cases:**

- **Login:** For users to log into the system.
- **Register:** For new users to register into the system.
- **Dashboard:** The main interface for users after logging in.
- **Generate Challan:** For officers to generate new challans.
- **View Challan:** For users to view their challans.
- **Pay Challan:** For users to pay their challans.

❖ **Backend Use Cases:**

- **Authenticate User:** Handles user authentication.
- **Manage Users:** Manages user registration and data.
- **Generate Challan:** The backend logic for generating a challan.
- **Store Challan Data:** Stores challan details in the database.
- **Retrieve Challan Data:** Retrieves challan details from the database.
- **Process Payment:** Processes the payment of challans.
- **Send Notification:** Sends notifications about challan status and payment.

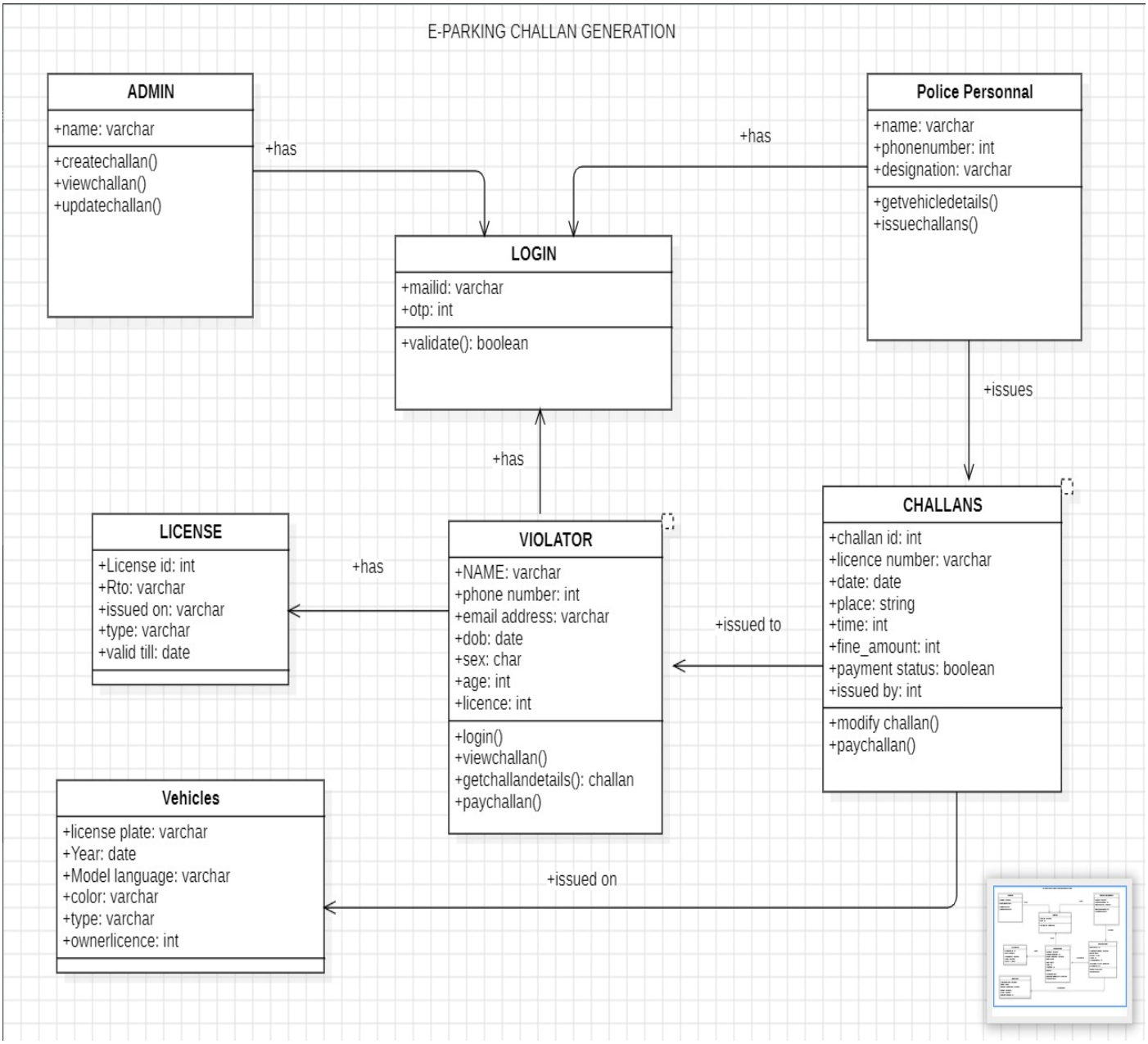
❖ **Relationships**

- Admin, EndUser, and Officer interact with various frontend use cases.
- Each frontend use case has a corresponding backend functionality.
- Include relationships (<<include>>) are used to show that certain actions (like generating a challan or processing a payment) involve multiple backend processes.

EX_NO: 8 **DATE: 30-04-24**

CLASS DIAGRAM

A class diagram is a type of UML(Unified Modeling Language) diagram that represents the structure and behavior of a system or application by illustrating the classes, attributes, methods, and relationships between objects. Class Diagrams are widely used in software development to visualize the static design of a system.



The above class diagram's breakdown into the classes and their relationships is given below,

❖ **Login:**

- This class likely represents the login functionality for uses of the system. It has the attributes like mailid and otp for logging in.
- It has a method validate(), which checks the entered credentials against stored ones to grant access to the system.

❖ **Admin:**

- This class represents the system administrator and has an attribute named name.
- It has methods for managing the system,such as createchallan() to create parking tickets, viewchallan() to view existing tickets, and updatechallan() to udate challan details.

❖ **Police Personnel:**

- This class represents police personnel who can issue parking tickets. It has attributes like name, phonenumber, designation.
- It has a method named issuechallans() to create parking tickets, likely similar to the createchallan() method in the Admin class.
- It also has a method named getvehicledetails(), likely to retrieve details of a vehicle to be included in a parking ticket.

❖ **License:**

- This class represents a vehicle license and has attributes such as LicenseID, licencenumber, Rto, issued on, valid till, etc., storing information about the vehicle's license.

❖ **Violator:**

- This class likely represents the person who violated parking rules and has attributes like name, date, phonenumber, emailaddress, dob, sex, etc., storing personal information about the violator.

❖ **Challan:**

- This class represents a parking ticket, likely containing attributes like challan id, licence, likely a reference to the License class, place, time, date, and fine_amount.
- It also has methods like paychallan(), likely to record payment of the ticket, modify challan(), likely to update details of the ticket, and getchallandetails(), likely to retrieve details of a specific ticket.

❖ **Vehicles:**

- This class represents the vehicle that violated parking rules and has attributes like licenseplate, Year, Model, color, and type.
- It also has an attribute ownerlicence, likely a reference to the License class.

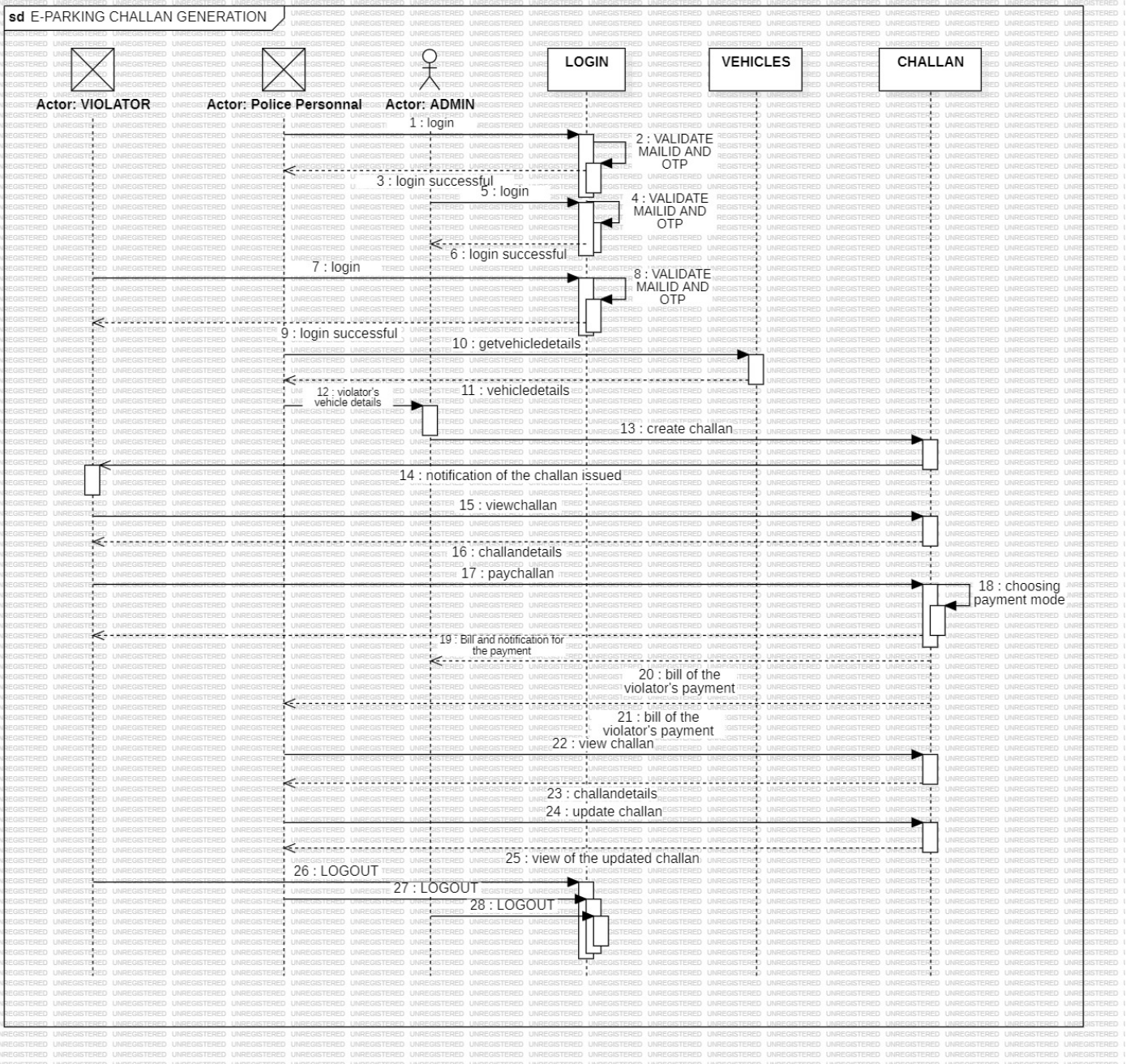
The relationships between the classes show how they interact with each other. For instance, the Challan class has a reference to the License class, indicating that a parking ticket is associated with a particular vehicle license. Similarly, the Challan class also has a reference to the Violator class, indicating that a parking ticket is issued to a specific person.

SOFTWARE ENGINEERING CONCEPTS – LAB MANUAL

EX_NO: 9 **DATE: 10-05-24**

SEQUENCE DIAGRAM

A sequence diagram is a type of UML (Unified Modeling Language) diagram that represents how objects interact in a particular sequence to achieve a specific behavior or functionality within a system. It is considered a dynamic diagram because it focuses on the flow of messages and interactions over time, rather than the static structure of the system.



❖ Actor**: Violator**
- This represents the person who violated parking rules.

❖ Actor: **Police Personnal**
- This represents a police officer who will issue the parking ticket.

❖ Actor: **ADMIN**
- This represents an administrator who can view and manage parking tickets.

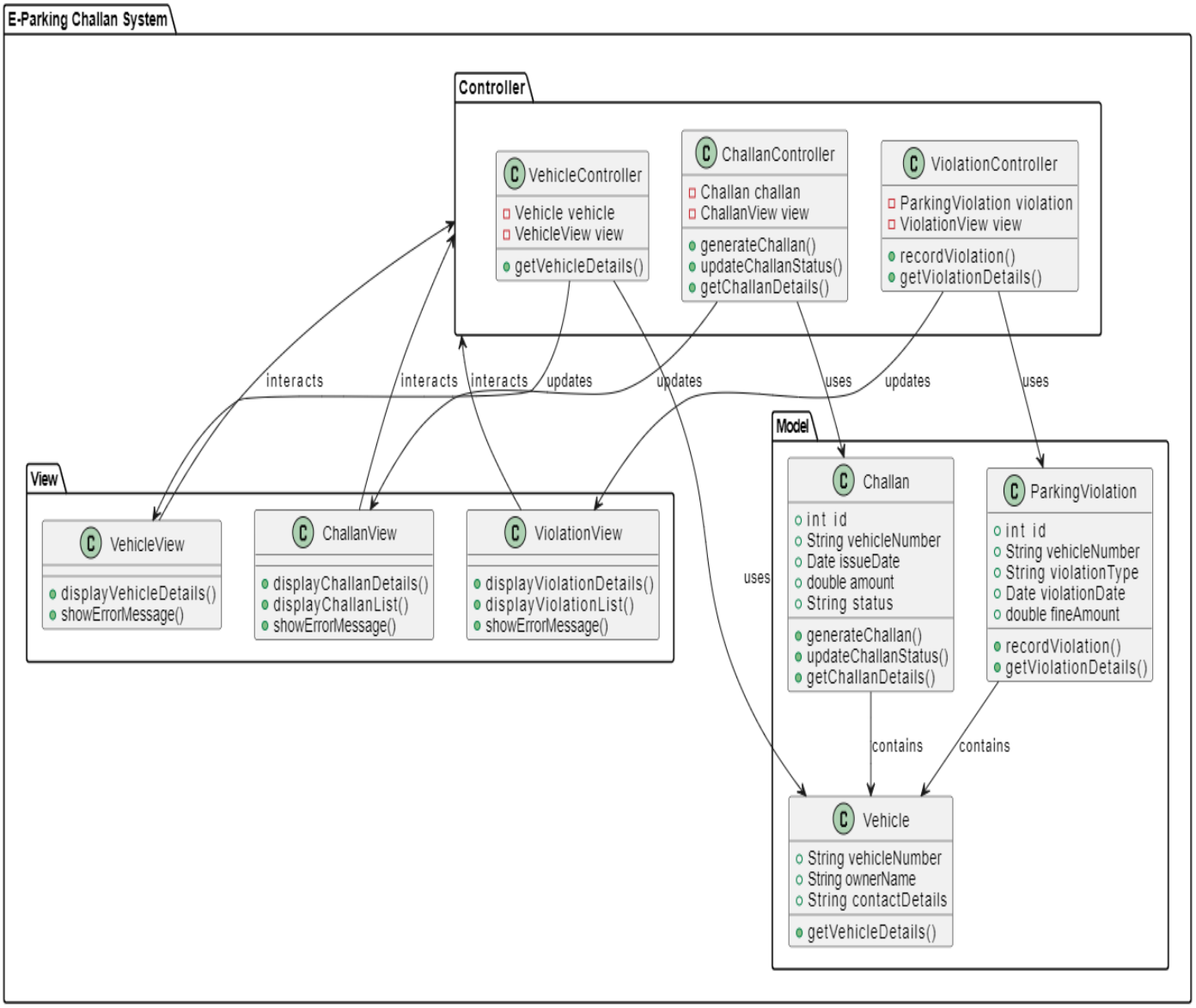
❖ The sequence of interactions is as follows:
- **Login:** The police officer first logs into the system **(1)**.
- **Validate Login:** The system validates the officer's credentials **(2)**.
- **Login Successful/Failed:** The system grants access **(3)** upon successful validation, or denies access if the credentials are invalid.
- **If Login Successful:**
  - **Validate Violator:** The officer enters the violator's email address and OTP **(4).**
  - **Validate Email and OTP:** The system validates the violator's credentials **(5)**.
  - **Login Successful/Failed:** Similar to step 3, the system grants access **(6)** if the credentials are valid, or denies access if not.
- **If Login Successful**:
  - **Login:** The violator logs in to the system, though it is unclear why this step is included here since the officer is issuing the ticket **(7)**.
  - **Validate Email and OTP:** The system validates the violator's credentials again **(8)**, though it seems redundant since this was already done in step 5.
- **If Login Successful:**
  - **Login Successful:** The system confirms successful login **(9)**.
  - **Get Vehicle Details:** The officer retrieves the vehicle details associated with the violation **(10)**.
  - **Vehicle Details**: The system provides the vehicle details to the officer **(11)**.

- **Create Challan:** The officer creates a parking ticket for the violator **(13)**.
- **Notification of Challan Issued:** The system sends a notification to the violator about the issued ticket **(14)**.

- **End of Sequence**
- **View Challan:** The violator can view the details of the parking ticket **(15)**.
- **Challan Details:** The system displays the challan details to the violator **(16)**.
- **Optional:**
  - **Update Challan:** The admin can update the details of the parking ticket **(17)**.
  - **Updated Challan View:** The system displays the updated ticket details **(18)**.
  - **View Challan:** The violator can again view the challan details **(19)**.
  - **Challan Details:** The system displays the challan details to the violator **(20)**.
  - **Pay Challan:** The violator can choose a payment method to pay the parking ticket **(21)**.
  - **Choosing Payment Mode:** The system facilitates the selection of a payment mode **(22)**.
- **Payment Successful:**
  - Bill and Notification for the Payment: The system generates a bill and sends a notification regarding the successful payment **(23)**.
  - Bill of the Violator's Payment: The violator receives the bill **(24)**.
- **End Sequence**
- **Logout**: The violator logs out of the system **(26)**.
- **Logout**: The officer logs out of the system **(27)**.
- **Logout**: The admin logs out of the system **(28)**.

EX_NO: 10 **DATE: 17-05-24**

ARCHITECTURAL PATTERN (MVC)



❖ **Model:**

- **Challan**
  - Represents a parking citation. It contains attributes like id, vehicleNumber, issueDate, amount, and status.
  - It likely has methods for CRUD (Create, Read, Update, Delete) operations on challan data.
- **Vehicle**
  - Represents a vehicle. It contains attributes like id, vehicleNumber, and ownerDetails (which likely has nested attributes for owner name, contact details, etc.).
  - It likely has methods for CRUD operations on vehicle data.
- **ParkingViolation**
  - Represents a parking violation. It contains attributes like id, violationType, violationDate, and fineAmount.
  - It likely has methods for CRUD operations on parking violation data.

❖ **View:**

- **ChallanView**
  - Responsible for displaying challan details. It likely has methods to display a single challan (displayChallanDetails()) or a list of challans (displayChallanList()).
- **VehicleView**
  - Responsible for displaying vehicle details. It likely has a method to display vehicle details (displayVehicleDetails()).
- **ViolationView**
  - Responsible for displaying parking violation details. It likely has a method to display violation details (displayViolationDetails() or displayViolationList()).

❖ **Controller:**

- **VehicleController**
  - Handles user interactions related to vehicles. It likely has methods to

    - getVehicleDetails(): retrieve details of a particular vehicle.
- **ChallanController**
    - Handles user interactions related to challans. It likely has methods to:
        - generateChallan(): create a new parking citation.
        - updateChallanStatus(): update the status of a parking citation (e.g., paid, unpaid).
        - getChallanDetails(): retrieve details of a particular parking citation.
- **ViolationController**
    - Handles user interactions related to parking violations. It likely has methods to:
        - recordViolation(): record a new parking violation.
        - getViolationDetails(): retrieve details of a particular parking violation.

Interactions

The controllers interact with the models to retrieve and update data, and then instruct the views to display the data to the user. For instance, when a user wants to view the details of a parking citation, the ChallanController would likely call the getChallanDetails() method of the Challan class to retrieve the data, and then instruct the ChallanView to display the details.