TOLL UI FOR DARB/SALIK SIMULATION

Developing a Toll Plaza Management System Web Service with security vulnerabilities and real-time data access, using REST APIs.

For

TACTICAL CYBERANGE SIMULATIONS PVT. LTD.

Hand-Over Documentation

April 2024

DOCUMENTED BY: AADITYA RENGARAJAN

DEVELOPED BY: INTELLX SOFTWARE CONSULTANCIES

TABLE OF CONTENTS

TABLE OF FIGURES AND ALGORITHMS

1. INTRODUCTION TO THE TOLL PLAZA SYSTEM

The Toll Plaza Management System Web Application epitomizes a sophisticated integration of technology into the critical infrastructure of roadway management, specifically focusing on toll collection and vehicle management processes. This digital framework is engineered to streamline operations at toll plazas, enhance security measures, and provide seamless user experiences for both administrators and cars passing through toll booths. The system harnesses the capabilities of FastAPI for backend services, offering a robust, scalable, and efficient web service architecture.

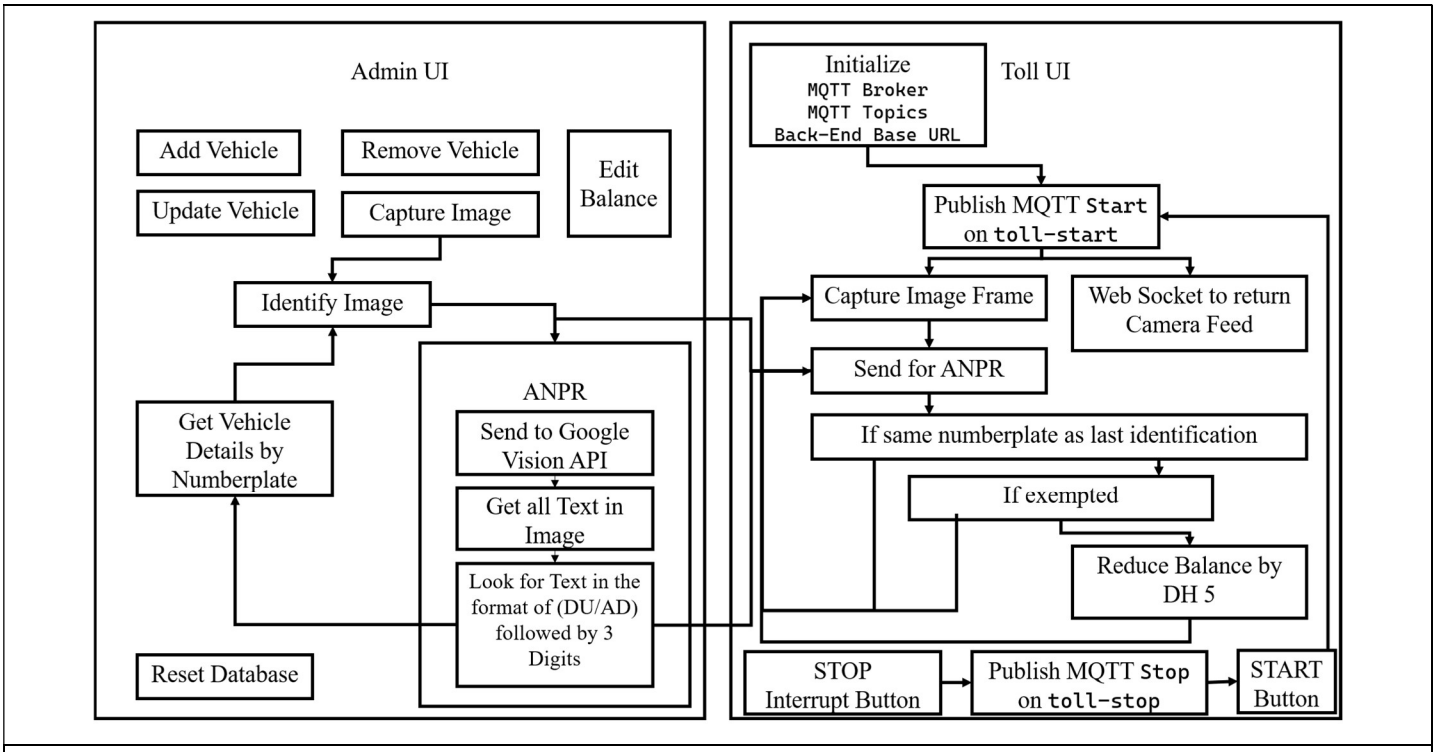

Figure 1: System Architecture

2. SOFTWARE COMPONENTS INVOLVED

2.1. ADMIN-UI

2.1.1 Back-End File Structure

——backend

| | `action.log`: Log of all actions that are done in Admin UI. File can be hijacked from the interface.

| | `app.py`: Main FastAPI Back-End. Required for Toll-UI as well. Hosted on port 8000. Visit /docs to see documentation.

| | `car_database.db`: Main SQLite Database.

| | `Dockerfile`

| | `requirements.txt`: Required Python Modules.

| | `Toll Plaza Test Collection.postman_collection.json`: POSTMAN Sample Requests and Responses for testing.

| |

| ├——modules

| | | `data_odin.py`: Database Management Python Module

| | ├——anpr

| | | | `google_ocr.py`: Google OCR API

| | | | `main.py`: Perform Google OCR on an input image and return numberplate of format (DU/AD) followed by 3 numbers.

| | | | `vision_api.json`: Google Vision API Keys.

2.1.2 Back-End Usage

2.1.2.1 Using Docker

Simply navigate to the parent directory of back-end, and run `docker-compose up --build -d`

2.1.2.2 Installation

Install all the required modules using `pip install -r requirements.txt`. Then, navigate to `backend` directory and run `python main.py`

2.1.3 API Documentation

The Toll Plaza API provides a comprehensive set of functionalities to manage and monitor toll transactions efficiently. It offers endpoints for various operations such as adding cars, retrieving car details, processing toll transactions, and managing administrative tasks.

User Authentication and Access Control:

The API ensures secure access through user authentication, implemented using OAuth2 token-based authentication. Administrators can obtain access tokens by logging in, which grants them access to protected endpoints. This mechanism ensures that only authorized users can perform administrative tasks within the system, enhancing security and preventing unauthorized access.

Car Management:

Administrators can add, retrieve, update, and delete car details using dedicated endpoints. They can add new cars to the system, update existing car information such as owner name, car model, and color, and delete cars when necessary. Additionally, administrators can retrieve detailed information about specific cars, including their balance, stolen status, and exemption status.

Transaction Processing:

The API facilitates toll transaction processing for vehicles passing through toll booths. Administrators can record toll transactions, update car balances, and handle exemptions seamlessly. Transactions are logged, providing a comprehensive transaction history for auditing and monitoring purposes.

Stolen and Exemption Status Management:

Administrators can flag cars as stolen or exempted from toll charges using dedicated endpoints. This functionality helps in managing toll operations effectively by preventing toll charges for exempted vehicles and alerting authorities about stolen vehicles.

ANPR Image Processing:

The API includes functionality for processing Automatic Number Plate Recognition (ANPR) images captured at toll booths. It extracts license plate numbers from images, enabling automated identification of vehicles passing through toll booths. This feature enhances efficiency and accuracy in toll collection processes.

Database Management:

Administrators can perform database management tasks such as resetting the database and deleting user accounts using dedicated endpoints. These operations ensure data integrity, system maintenance, and user management within the toll plaza system.

The Toll Plaza API includes purposeful SQL injection (SQLi) vulnerabilities and file hijacking vulnerabilities to facilitate learning and practice in secure coding and penetration testing labs. These intentional vulnerabilities are designed to simulate real-world scenarios where security flaws may exist, allowing users to understand the risks associated with such vulnerabilities and learn how to mitigate them effectively.

SQL Injection Vulnerabilities:

The API intentionally incorporates SQL injection vulnerabilities in certain endpoints to demonstrate the risks associated with improper input validation and insufficient SQL query parameterization. By exploiting these vulnerabilities, users can learn about the potential consequences of SQLi attacks, such as unauthorized data access, data manipulation, and database compromise. Additionally, users can practice crafting and executing SQL injection payloads to exploit these vulnerabilities and gain unauthorized access to sensitive data within the system.

File Hijacking Vulnerabilities:

In addition to SQL injection vulnerabilities, the API also includes purposeful file hijacking vulnerabilities that demonstrate the risks associated with insecure file handling and directory traversal attacks. These vulnerabilities allow users to upload malicious files or manipulate file paths to access restricted files or directories on the server. By exploiting these vulnerabilities, users can learn about the potential impact of file hijacking attacks, such as unauthorized access to sensitive files, arbitrary code execution, and server compromise. Additionally, users can practice implementing secure file upload and handling mechanisms to prevent file hijacking attacks and mitigate the associated risks effectively.

Educational Purpose:

The inclusion of purposeful vulnerabilities in the Toll Plaza API serves an educational purpose by providing users with hands-on learning opportunities in secure coding practices, vulnerability assessment, and penetration testing techniques. By interacting with the API and identifying and exploiting these vulnerabilities, users can gain valuable insights into common security flaws and learn how to address them proactively in real-world applications. Furthermore, users can experiment with different attack scenarios, defensive strategies, and security best practices to enhance their skills and knowledge in cybersecurity.

Let us explore the functionality of each endpoint below:

POST /token

This endpoint facilitates user login to obtain an access token. Users can authenticate themselves by providing valid credentials. Upon successful authentication, the system generates and returns an access token, which the user can then use to access protected resources within the API.

POST /admin/add_car/

This endpoint enables administrators to add new cars to the toll plaza system. Administrators can provide details such as the plate number, vehicle type, owner information, and any relevant metadata associated with the car. Once added, the car becomes registered in the system and is eligible for toll transactions.

GET /admin/car_details/{plate_number}

With this endpoint, administrators can retrieve detailed information about a specific car registered in the toll plaza system. By providing the plate number as a parameter, administrators can access data such as the vehicle type, owner details, registration date, and any other pertinent information stored for the car.

GET /admin/balance/{plate_number}

This endpoint allows administrators to check the balance associated with a particular car. By providing the plate number of the car as a parameter, administrators can retrieve the current balance associated with that vehicle's toll account. This information is crucial for monitoring toll payments and ensuring sufficient funds for future transactions.

POST /admin/transaction/{toll_booth}

Administrators can utilize this endpoint to add toll transactions for vehicles passing through a specific toll booth. By providing the toll booth identifier and relevant transaction details, such as the plate number, timestamp, and transaction amount, administrators can record toll payments and update the car's balance accordingly.

PUT /admin/stolen/{plate_number}

This endpoint allows administrators to flag a car as stolen within the toll plaza system. By providing the plate number of the stolen vehicle, administrators can mark it as stolen, triggering appropriate actions such as alerting authorities and preventing further toll transactions associated with the car.

PUT /admin/exempted/{plate_number}

Administrators can utilize this endpoint to exempt a specific car from toll charges. By providing the plate number of the exempted vehicle, administrators can mark it as exempted, allowing it to pass through toll booths without incurring charges for a specified period or under specific conditions.

PUT /admin/update_car/{plate_number}

This endpoint enables administrators to update the details of a registered car within the toll plaza system. Administrators can modify information such as the vehicle type, owner details, or any other metadata associated with the car by providing the plate number and the updated information.

DELETE /admin/delete_car/{plate_number}

Administrators can use this endpoint to delete a car from the toll plaza system. By providing the plate number of the car to be deleted, administrators can remove all associated data and transactions from the system, effectively deregistering the vehicle.

GET /admin/car_history/{plate_number}

This endpoint allows administrators to retrieve the transaction history of a specific car within the toll plaza system. By providing the plate number of the car as a parameter, administrators can access a chronological list of all toll transactions associated with that vehicle, including timestamps and transaction details.

DELETE /admin/reset_db

Administrators can use this endpoint to reset the toll plaza system's database. This action deletes all data stored within the system, including car registrations, transaction records, and user information. It effectively restores the system to its initial state, ready for fresh registrations and transactions.

DELETE /admin/delete_user/{username}

This endpoint enables administrators to delete a user account from the toll plaza system. By providing the username of the user to be deleted, administrators can remove their account along with any associated data or permissions within the system.

POST /admin/anpr/

This endpoint processes Automatic Number Plate Recognition (ANPR) images captured at toll booths. Administrators can submit ANPR images containing vehicle plate numbers, and the system will extract the plate number information for further processing and transaction recording.

POST /lookup

This endpoint facilitates car lookup functionality within the toll plaza system. Users can query the system by providing relevant parameters such as plate number or owner details to retrieve information about registered cars within the system.

2.1.4 Front-End File Structure

```
-——frontend
    |   Dockerfile
    |   nginx.conf: NGINX Configuration File
    |   package-lock.json
    |   package.json
    |
    ├——public: Compiled ReactJS Application for hosting
    |       favicon.ico
    |       index.html
    |       logo192.png
    |       logo512.png
    |       manifest.json
    |       robots.txt
    |
    └——src
        |   App.css
        |   App.jsx: Main Application
        |   App.test.js
        |   index.css
        |   index.js
        |   logo.svg
        |   reportWebVitals.js
        |   setupTests.js
        |
        ├——components
        |       AddCarDetails.jsx: Add Car Functionality
        |       CarDetails.jsx: View car details by ANPR Functionality
        |       Details.jsx: View all car details functionality
        |       History.jsx: View history of a car functionality
```

```
|       LogIn.jsx: Log-In functionality
|       LogOut.jsx: Log-Out Functionality
|       Navbar.jsx
|       SignUp.jsx: (Deprecated) Sign-Up Functionality
|       UpdateCarDetails.jsx: Update a Car's Details Functionality
|
├──services
|       api.jsx: API Calls with back-end
|
├──styles
|       datatables.min.css
|       styles.css
|
└──utils
        auth.jsx: Authenticate and manage session
        timer.jsx: Delay() function
```

2.1.5 Front-End Usage

2.1.5.1 Using Docker

Simply navigate to the parent directory of back-end, and run `docker-compose up --build -d`

2.1.5.2 Installation

Install all the required modules using `npm i`. Then, navigate to `frontend` directory and run `npm run start`

2.1.6 Front-End Code Documentation

The React.js front-end for the Toll Plaza System provides a user-friendly interface designed primarily for laptops and larger screens of similar ratios. While the interface is not fully responsive, it offers administrators a seamless experience for managing toll plaza operations. The theme of the front-end features a dark purple-blue gradient with a hint of glassmorphism, enhancing the visual appeal and providing a modern look and feel to the application.

Ease of Use for Administrators:

The front-end interface makes life easy for administrators by offering intuitive navigation and straightforward access to essential functionalities. Administrators can log in securely, view car details, update car information, add new cars to the system, and access transaction history seamlessly. The navbar provides easy access to different sections of the application, enhancing usability and efficiency in managing toll plaza operations.

React Hooks and Components:

The front-end utilizes React hooks and components to manage state, handle side effects, and render user interfaces dynamically. The useEffect hook is used to fetch car details from the API and ensure that users are authenticated properly. Each route in the application is associated with a specific component, allowing for modularization and reusability of code. Components such as LogIn, Details, CarDetails, UpdateCarDetails, AddCarDetails, History, Navbar, and LogOut encapsulate specific functionalities and contribute to the overall structure and functionality of the application.

2.1.7 Endpoints Overview:

LogIn (/login):

Purpose: Allows administrators to log in securely.
Functionality: Renders the LogIn component, where administrators can enter their credentials and authenticate themselves.
Implementation: Uses the LogIn component to handle user authentication, making a request to the backend API to verify credentials and obtain access tokens.

Details (/details):

Purpose: Displays general details and summary information.
Functionality: Renders the Details component, which provides an overview of the toll plaza system, including transaction statistics, user activity, and system status.
Implementation: Retrieves relevant data from the backend API and displays it in the Details component, offering administrators insights into the overall performance of the system.

CarDetails (/cardetails):

Purpose: Displays detailed information about individual cars.
Functionality: Renders the CarDetails component, allowing administrators to view specific details about registered cars, including owner name, model, color, balance, and status.
Implementation: Fetches car details from the backend API and presents them in a user-friendly format within the CarDetails component, facilitating efficient management of car information.

History (/history/:plateNumber):

Purpose: Provides a history of transactions and activities related to a specific car.
Functionality: Renders the History component, which displays a chronological list of transactions, events, and activities associated with a particular car identified by its plate number.
Implementation: Retrieves car history data from the backend API based on the provided plate number and presents it in the History component for administrators to review and analyze.

UpdateCarDetails (/update/:plateNumber):

Purpose: Allows administrators to update details of a specific car.
Functionality: Renders the UpdateCarDetails component, where administrators can modify information such as owner name, model, color, and other relevant details for a car identified by its plate number.
Implementation: Handles user input and updates car details by making requests to the backend API, ensuring that changes are reflected accurately in the system.

AddCarDetails (/add_car):

Purpose: Enables administrators to add new cars to the system.
Functionality: Renders the AddCarDetails component, providing a form for administrators to input details of a new car, including plate number, owner name, model, color, and other relevant information.
Implementation: Validates user input and submits new car details to the backend API, ensuring that the new car is registered correctly in the system.

LogOut (/logout):

Purpose: Allows administrators to log out securely and terminate their session.
Functionality: Renders the LogOut component, where administrators can initiate the logout process to end their current session and clear authentication tokens.
Implementation: Handles logout functionality by removing authentication tokens from local storage and redirecting the user to the login page, ensuring a secure logout process.

2.2: TOLL-UI

2.2.1 Back-End File Structure

`-——backend`

`|  Dockerfile`

`|  lookup.html:` Numberplate lookup HTML for End-Users

`|  README.txt`

`|  requirements.txt:` Python Module Requirements

`|  ui_backend.py:` Live ANPR Back-End to communicate with Admin-UI and provide live camera-connected detection

2.2.2 Back-End Usage

Install all the required modules using `pip install -r requirements.txt`. Then, navigate to `backend` directory and run `python ui_backend.py`

The Toll-UI is dockerized, however, does not support being run inside a docker container as the user will have to spend a long time configuring the docker container to connect to the hardware camera of the system, which is a tedious and time-taking procedure.

2.2.3 API Documentation

The Toll Plaza System API serves as the backbone for managing toll plaza operations efficiently. It leverages FastAPI, a modern web framework for building APIs with Python, to handle incoming requests and WebSocket connections seamlessly. The API integrates various functionalities, including license plate detection, balance reduction, and interaction with the admin UI backend.

2.2.3.1 Workflow Overview:

Initialization and Authentication: The API initializes essential components and establishes connections to external services, including MQTT brokers and the admin UI backend.
It generates an access token by authenticating with the admin UI backend using predefined credentials (self-identifying as either DARB or SALIK simulation plaza).

WebSocket Camera Feed: The API provides a WebSocket endpoint ("/ws/camera") for streaming live camera feed data. It continuously captures frames from the camera and processes them for license plate detection and balance reduction.

License Plate Detection and Balance Reduction: Upon receiving a frame, the API asynchronously detects license plates using computer vision techniques.
If a license plate is detected, the corresponding car details are retrieved from the admin UI backend. The API then reduces the balance associated with the detected license plate by a predefined amount, ensuring seamless toll collection.

HTTP Endpoints for Additional Functionality: The API offers several HTTP endpoints to retrieve car details, perform MQTT control actions, and serve static files for car lookup.

2.2.3.2 MQTT Support and Functionality:

MQTT, or Message Queuing Telemetry Transport, stands as a pinnacle in the realm of lightweight, efficient messaging protocols. MQTT was designed with a vision to facilitate communication between devices with minimal bandwidth and power consumption. Over the years, MQTT has gained widespread adoption across various industries, becoming a cornerstone in the Internet of Things (IoT) ecosystem and beyond.

At its core, MQTT operates on a publish-subscribe messaging model, enabling seamless communication between publishers and subscribers. This model decouples the sending and receiving of messages, allowing for asynchronous communication and scalability. MQTT is characterized by its lightweight nature, making it ideal for resource-constrained devices and low-bandwidth networks. The protocol utilizes a simple header structure and operates over TCP/IP or other transport protocols, ensuring efficient message delivery across diverse environments.

The MQTT Publisher in Toll-UI is a fundamental component responsible for disseminating essential information to a centralized Node-RED instance. This instance serves as the nerve center, orchestrating the functionalities and operations of the toll plaza. MQTT acts as the conduit through which data flows, enabling rapid transmission of critical messages and updates across the system.

One of the primary functionalities facilitated by MQTT is the initiation and control of toll plaza operations. Through the publication of messages to designated topics such as "toll-start" and "toll-stop," the system can be remotely activated or deactivated, allowing for flexible management of toll collection activities. This capability ensures that toll operations can be seamlessly controlled and monitored, contributing to enhanced operational efficiency and adaptability.

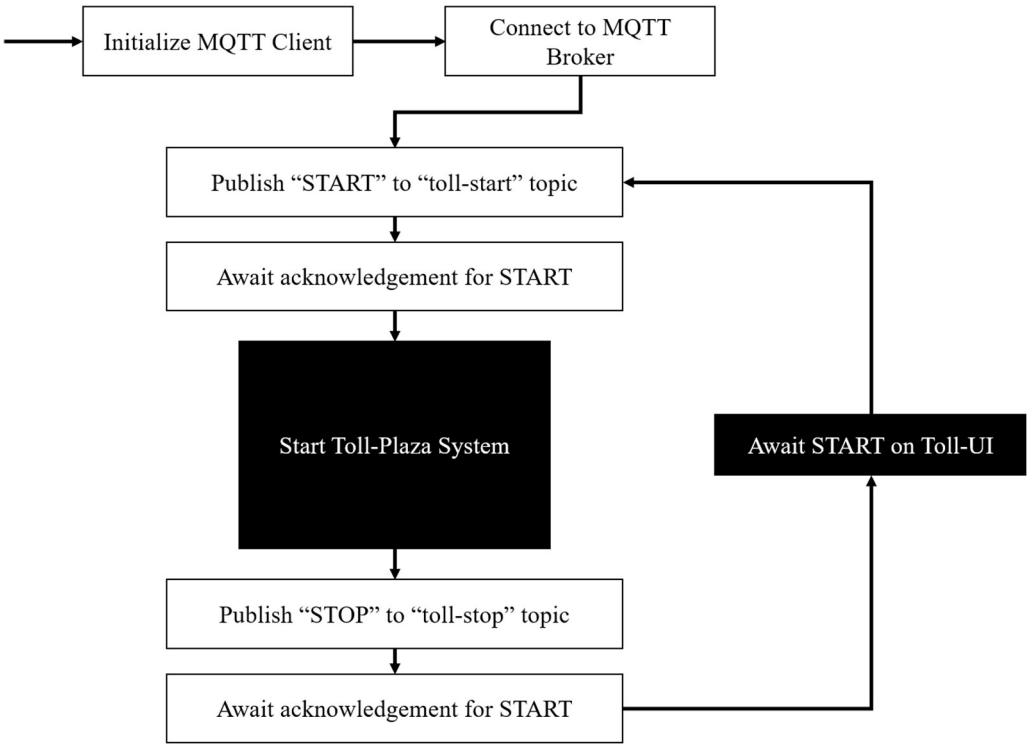

Figure 2: Application of MQTT on Toll-Plaza

```
FUNCTION start_operations(client):
    start()
    publish_message(client, "START", "toll-start")

FUNCTION stop_operations(client):
    stop()
    publish_message(client, "STOP", "toll-stop")

FUNCTION publish_message(client, message, topic):
    PUBLISH message to topic using MQTT client
    PRINT "Published message:", message, "to topic:", topic

MAIN:
    client = INITIALIZE MQTT client

    TRY:
        CONNECT to MQTT broker ("test.mosquitto.org", port: 1883)
        start_operations(client)
    EXCEPT Exception AS e:
        PRINT "MQTT Publishing Failed:", e
```

Algorithm 1: MQTT Pseudocode

2.2.3.2 Endpoint Details:

GET /api/car/details: Returns details of the last detected car. Implements caching to avoid frequent backend requests for car details.

GET /car-lookup: Serves a static HTML file for performing car lookup operations. Allows users to search for car details using a web interface.

GET /api/stop-mqtt: Stops the MQTT communication by publishing a "STOP" message to the specified MQTT topic. Ensures proper shutdown of MQTT communication when required.

GET /api/start-mqtt: Initiates MQTT communication by publishing a "START" message to the specified MQTT topic. Restarts MQTT communication after a previous stop action.

2.2.4 Front-End File Structure

```
-——frontend
|   Dockerfile
|   nginx.conf: NGINX Configuration for Deployment
|   package-lock.json
|   package.json
|   README.md
├——public: Compiled folder for production deployment
|       favicon.ico
|       index.html
|       logo192.png
|       logo512.png
|       manifest.json
|       robots.txt
└——src
        App.css
        App.jsx: Source Code with all functionalities
        App.test.js
        index.css
        index.js
        logo.svg
        reportWebVitals.js
        setupTests.js
```

2.2.5 Front-End Usage

Install all the required modules using `npm i`. Then, navigate to `frontend` directory and run `npm run start`

The Toll-UI is dockerized, however, does not support being run inside a docker container as the user will have to spend a long time configuring the docker container to connect to the hardware camera of the system, which is a tedious and time-taking procedure.

2.2.6 Front-End Documentation

React Components and State Management:

At the heart of the system lies React.js, leveraging its component-based architecture and state management capabilities to ensure a dynamic and responsive user experience. The App component serves as the central hub, orchestrating various sub-components and handling state changes.

The system harnesses the power of React hooks, particularly useState and useEffect, to manage component state and side effects efficiently. useState facilitates the creation of state variables such as carDetails, cameraFeed, and loading, enabling seamless interaction and data flow within the application.

Real-Time Data Retrieval and Display:

A pivotal aspect of the system is its ability to fetch real-time data from backend APIs and WebSocket connections, providing administrators with up-to-date information on vehicle details and camera feeds. The fetchCarDetails function, triggered at regular intervals, retrieves car details from the backend API, ensuring timely updates and accurate monitoring.

The WebSocket connection established for the camera feed enables the system to receive live frames directly from surveillance cameras, enhancing situational awareness and enabling swift responses to any anomalies detected.

User Interface and Interaction:

The user interface (UI) is thoughtfully crafted to prioritize usability and efficiency, catering to the needs of administrators tasked with overseeing toll plaza operations. The UI design encompasses intuitive navigation, sleek visual elements, and seamless interaction patterns, fostering a productive working environment for users.

Administrators are greeted with a loading screen during system initialization, providing feedback on the loading process and ensuring a smooth transition to the main interface upon completion. The UI dynamically adapts to changes in data availability, seamlessly transitioning between loading states and content display to maintain a seamless user experience.

Operational Controls and System Management:

In addition to surveillance capabilities, the system offers robust operational controls, empowering administrators to manage system state and perform critical actions with ease. The handleStart and handleStop functions enable administrators to initiate or halt system operations, facilitating quick responses to operational requirements and ensuring smooth system functionality.

2.2.7 `lookup.html` – Public Facing Vulnerability

2.2.7.1 UX Principles and UI Design Styles and Principles:

User Experience (UX) principles and User Interface (UI) design styles are crucial elements in creating effective and engaging web applications. In the provided code for a "Car Details Lookup" form, several UX and UI design principles can be observed:

- **Simplicity**: The design follows a minimalist approach with a clean layout and straightforward form fields, enhancing usability by reducing cognitive load.
- **Consistency:** The use of consistent fonts, colors, and spacing throughout the interface creates a cohesive visual experience, making it easier for users to navigate.
- **Accessibility:** The form includes appropriate labels and input placeholders, ensuring accessibility for all users, including those with disabilities who may rely on screen readers.
- **Feedback**: Instant feedback is provided to users upon form submission, with loading indicators and error messages displayed as appropriate, enhancing user confidence and clarity.
- **Responsive Design:** The design is responsive, adapting seamlessly to different screen sizes and devices, thus catering to a diverse user base.
- **Intuitiveness:** The form layout and labeling are intuitive, with clear instructions prompting users to enter the required information, reducing the likelihood of errors.
- **Engagement:** While the design is functional, it may benefit from additional elements such as icons or visual cues to enhance user engagement and aesthetic appeal.

2.2.7.2 Exploitable Form and SQL Injection (SQLi) Vulnerability:

Despite its functional design, the form in the provided code is susceptible to SQL Injection (SQLi) attacks, posing a significant security risk. SQL Injection occurs when malicious SQL code is inserted into input fields, exploiting vulnerabilities in the application's backend database.

In this case, the form accepts user input for the "Plate Number" field without proper validation or sanitization, allowing attackers to manipulate the input and execute arbitrary SQL commands. Below are five sample SQL Injection payloads that could be injected into the form's input field:

```
DU916' OR '1'='1'--
```

```
DU916'; DROP TABLE users;--
```

```
DU916'; SELECT * FROM information_schema.tables;--
```

```
DU916'; INSERT INTO admins (username, password) VALUES ("HACKER", "HACKED");--
```

Each of these payloads aims to manipulate the SQL query executed by the backend server, potentially leading to unauthorized data access, data manipulation, or even database deletion.

To mitigate SQL Injection vulnerabilities, the application should implement input validation and parameterized queries to sanitize user input effectively. Additionally, employing security measures such as web application firewalls (WAFs) and regularly updating software patches can help protect against SQL Injection attacks and ensure the security of user data.

As seen above, attackers can exploit SQLi vulnerabilities by manipulating the input fields to inject malicious SQL code. In this scenario where the web application allows users to search for car details using a plate number. Since the application fails to properly validate or sanitize user input, attackers can insert SQL code into the plate number field to modify the intended SQL query.

Attackers can execute more advanced SQL Injection payloads to perform various malicious actions, such as extracting sensitive data, modifying database records, or even executing system commands. For instance, an attacker could inject a payload like '; DROP TABLE users;-- to delete the entire users table from the database, resulting in data loss and potential service disruption.

While the concept of exploiting SQLi vulnerabilities through scanning images of vulnerable number plates might seem plausible, it presents a challenge as the back-end is built to sanitize the input. The Automated Number Plate Recognition (ANPR) system crops the image to extract the plate number for processing. However, as our system only captures a limited number of characters from the plate, such as the first five characters, it significantly reduces the effectiveness of SQL Injection attacks through image scanning.

Moreover, even if an ANPR system captures the full plate number, exploiting SQLi vulnerabilities through image scanning would require intricate manipulation of the captured data to inject SQL code effectively. This process would likely involve converting image data to text, identifying and isolating the plate number characters, and crafting a payload that fits within the constraints of the input field.

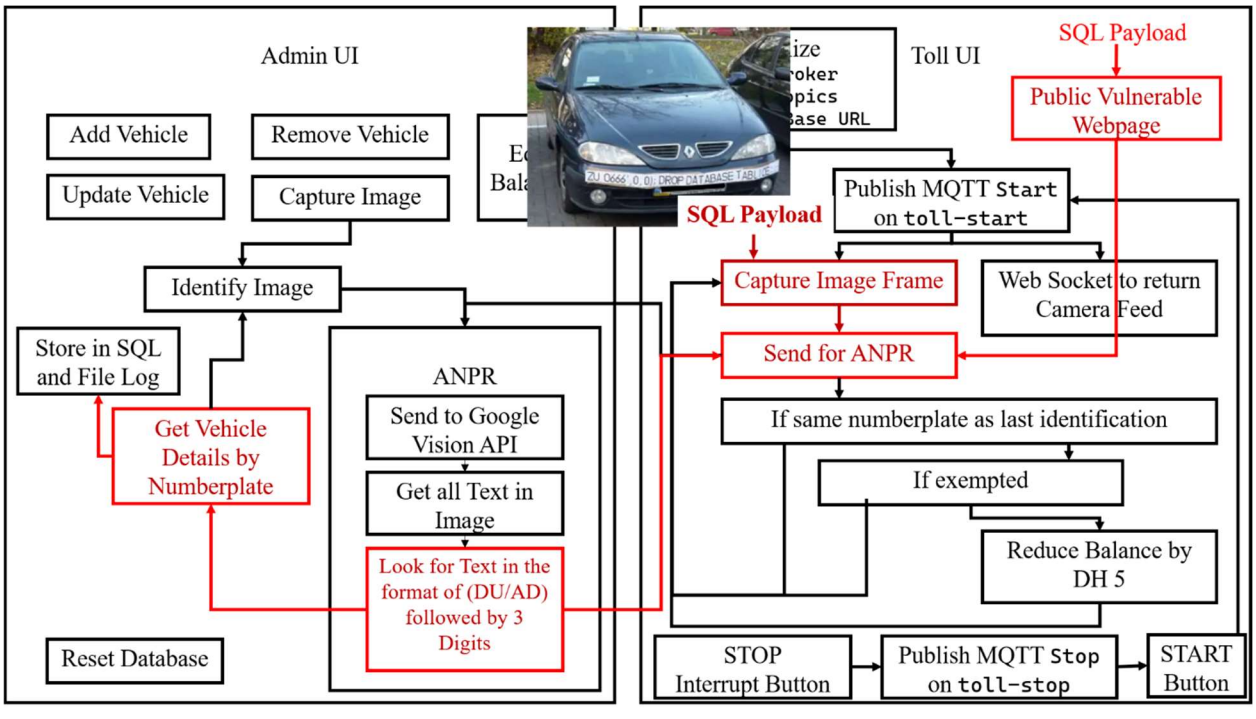

Figure 3: Adversarial Attack Techniques

CONCLUSION

The Toll Plaza Management System Web Service represents a significant leap forward in the management and operation of toll collection systems. By harnessing the capabilities of FastAPI for backend services, along with advanced vehicle identification technologies like ANPR and RFID, the system sets a new standard for efficiency, security, and user experience in toll plaza operations.

The system's design emphasizes security and data integrity, utilizing OAuth2PasswordBearer with JWT tokens for authentication and meticulously logging transactions and events for auditability. This focus on security is paramount, given the sensitivity of financial transactions and personal data handled by the system.

The flexible and scalable nature of the Toll Plaza Management System Web Service, coupled with its comprehensive feature set, makes it an invaluable tool for toll plaza administrators. It offers a streamlined interface for vehicle processing, transaction management, and administrative tasks, significantly reducing operational complexities and enhancing efficiency.

In conclusion, the Toll Plaza Management System Web Service exemplifies a balanced integration of technology and operational requirements, providing a robust, secure, and user-friendly platform for toll plaza management. Its deployment not only promises to improve operational efficiencies but also to enhance the driving experience for millions of users, marking a significant step forward in the modernization of toll collection infrastructure.