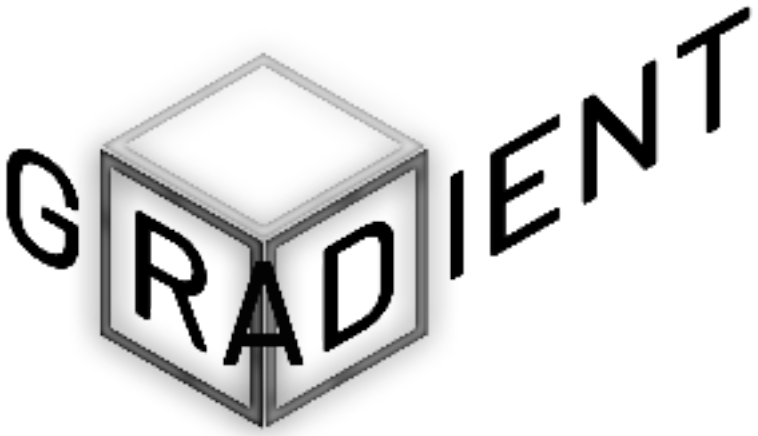

Software Requirements Specification

LightBot Adaptive Traffic Control System

Members:

Mohammed Gangat	*u17058849*
Rahul Kapoor	*u16034130*
Thiveshan Pillay	*u15007911*
Jared Gratz	*u16054972*

Team Gradient
COS 301 Capstone Project

Contents

1. Introduction

1.1 Purpose

LightBot aims to minimize the delays that are caused by traffic congestions at various intersections. The LightBot system will be used to prove a concept that the addition of a swarm reinforcement learning algorithm will develop a good policy for which to control the traffic lights at a given intersection. This policy should minimize the possibility of traffic congestion of an intersection, thus having a positive impact on productivity and maximise flow of traffic, particularly during peak hours.

1.2 Scope

The system will provide simulations to demonstrate the effectiveness in the case of the implementation of a swarm reinforcement learning algorithm for the traffic lights of a given intersection layout. This machine learning algorithm will use statistical metrics from a simulated traffic flow and real-time traffic flow data. The system will be accessible through a web application interface, specifically designed to provide an overview and interactive control of the state of the system and simulation.

1.3 Definitions, Acronyms and Abbreviations

1. API - Application Programming Interface
2. GUI - Graphical User Interface
3. MVC - Model-View-Controller

1.4 Overview

The remainder of this document is structured as follows: Section 2 provides the User Characteristics of the system. In Section 3, the Functional Requirements are listed, along with Constraints, Use Cases, Use Case Diagrams and a list of the Subsystems. The Domain Model of the system is depicted in Section 4. Section 5 deals with the Quality Requirements of the system. Traceability Matrices pertaining to Use Cases and Subsystems are shown in Section 6. The Non-Functional and Technology Requirements are described in Sections 7 and 8 respectively. Finally, any works cited are referenced in Section 9.

2. User Characteristics

2.1 User Stories

The intended client user base consists of the employees from 5DT.

I. As an administrator of the system, I would like a web portal application that can be accessed from anywhere (remotely) so that I can view a system state of the lightbot system, create and manage simulation, and see the effectiveness of a swarm reinforcement learning algorithm on reducing congestion.
II. As an administrator of the system, I would like to be able to create a profile (to store my information as well as roles, current status etc.) which will be used to give me access to the system as well as view other administrators profiles and elevate and demote roles.
III. As a user of the system, I would like a hassle free overview of the current state of the system, that being the status of the optimization controller as well as intersections that may be being managed by it so that I may better describe the state of the system.
IV. As an administrator of the system, I would like to be able to configure the current state of the system components individually so that I am able to switch between automated control of the traffic lights and manual control so that I can see the effectiveness of the swarm reinforcement learning algorithm.
V. As a regular user, I would like to have access to the system via a web portal, so that I can view the simulation, and see the effectiveness of the swarm reinforcement learning algorithm on reducing congestion.
VI. As a user, I would like the system to simulate a real-world intersection, so that I can see the benefits of having the swarm reinforcement learning optimized traffic light system in the real world.

3. Functional Requirements

3.1 Requirements & Constraints

Requirements:

- R1: The system needs to gather traffic data
    - R1.1: The system needs to gather data from a real-world source for the simulation.
    - R1.2: The system needs to randomly generate data for the simulation.
- R2: The system needs to have a graphical user interface (GUI)
    - R2.1: The system must be able to display the traffic simulation on the web application.

        - R2.2: The system must be able to allow control of the state of the traffic simulation on the web application.
            - R2.2.1: The interface must display a dashboard with statistics.
            - R2.2.2: The interface must display the simulation.
            - R2.2.3: The interface must display the controller options.
- R3: The system must have user controller options for the traffic lights.
    - R3.1: The system must allow for manual control of traffic lights.
    - R3.2: The system must allow for automated control of traffic lights.
    - R3.3 The system must allow for switching between manual and automated control
        - R3.3.1: The controller can be changed on the interface.
        - R3.3.2: The controller change should notify the system.
- R4: The system must collect statistical metrics that describe the efficiency of the current traffic flow.
- R5: The system must provide an API for accessing the current state of the traffic system.
    - R5.1: The interface must use the API to retrieve state data.
- R6: The system must have controllers for the traffic lights.
    - R6.1: It should have at least one controller that actively controls the state of traffic lights.
    - R6.2: It should also have a fixed time controller.
- R7: The system should be able to allow management of the databases.
    - R7.1: An administrator must be able to access and edit any data stored in the databases.

3.2 Use Cases & Diagrams

- U1: Access System GUI
    - U1.1 Register
    - U1.2 Log In
    - U1.3 Reset Password
- U2: View System GUI
    - U 2.1 View state of system
    - U 2.2 View simulated traffic flow
    - U 2.3 View active traffic controllers
    - U 2.4 Access settings
    - U 2.5 Log out of system
- U3: System Management
    - U 3.1 Manual control of traffic lights
    - U 3.2 Automated control of traffic lights
    - U 3.3 View controller states
    - U 3.4 Access simulation
- U4: User Database Management
    - U 4.1 View users
    - U 4.2 Add users
    - U 4.3 Delete users
    - U 4.4 Edit access

    - U 4.5 Revoke access
    - U 4.6 Activate user status
    - U 4.7 Deactivate user status
    - U 4.8 View user role
    - U 4.9 Edit user role
- U5: Simulation Control
    - U 5.1 Create real-world scenarios
    - U 5.2 View scenario results
    - U 5.3 Test emergency scenarios
    - U 5.4 Test situational scenarios
    - U 5.5 Change location & road layout

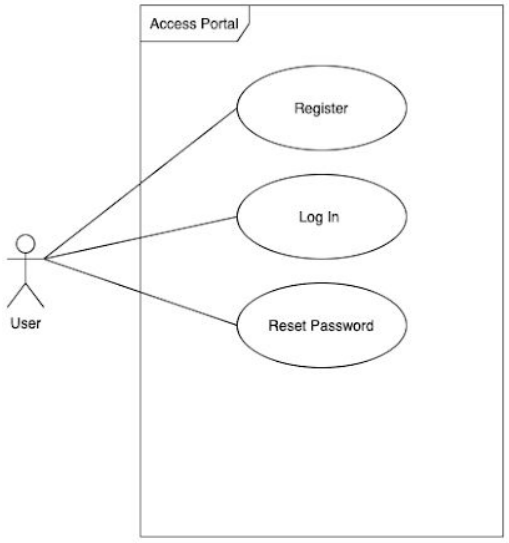

Figure 1: Use Case 1 Diagram

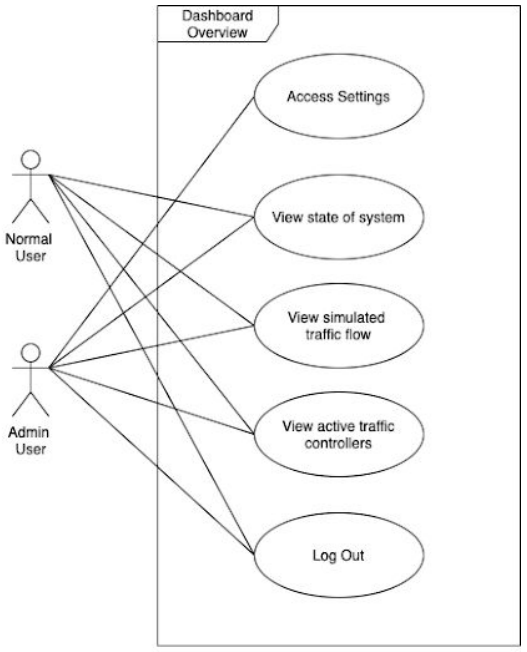

Figure 2: Use Case 2 Diagram

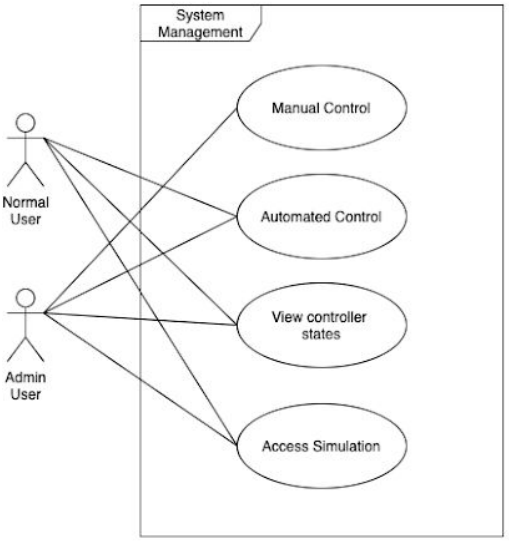

Figure 3: Use Case 3 Diagram

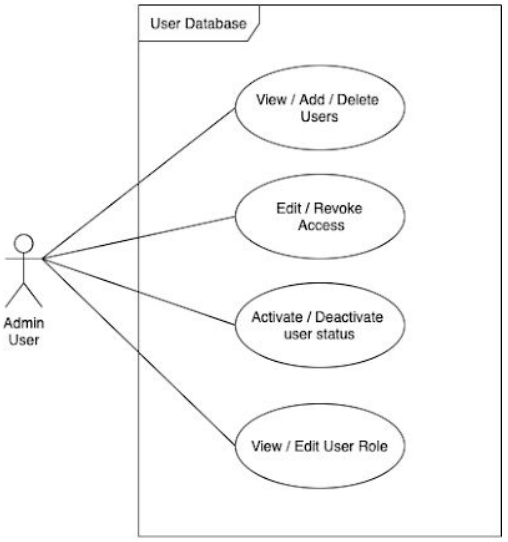

Figure 4: Use Case 4 Diagram

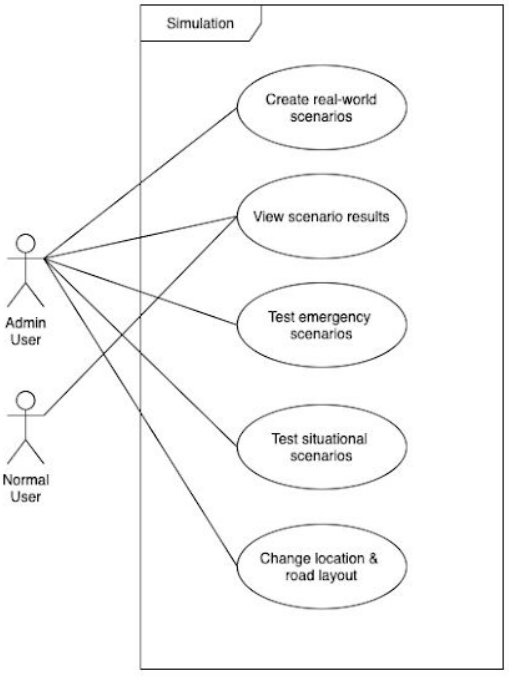

Figure 5: Use Case 5 Diagram

4. Domain Model

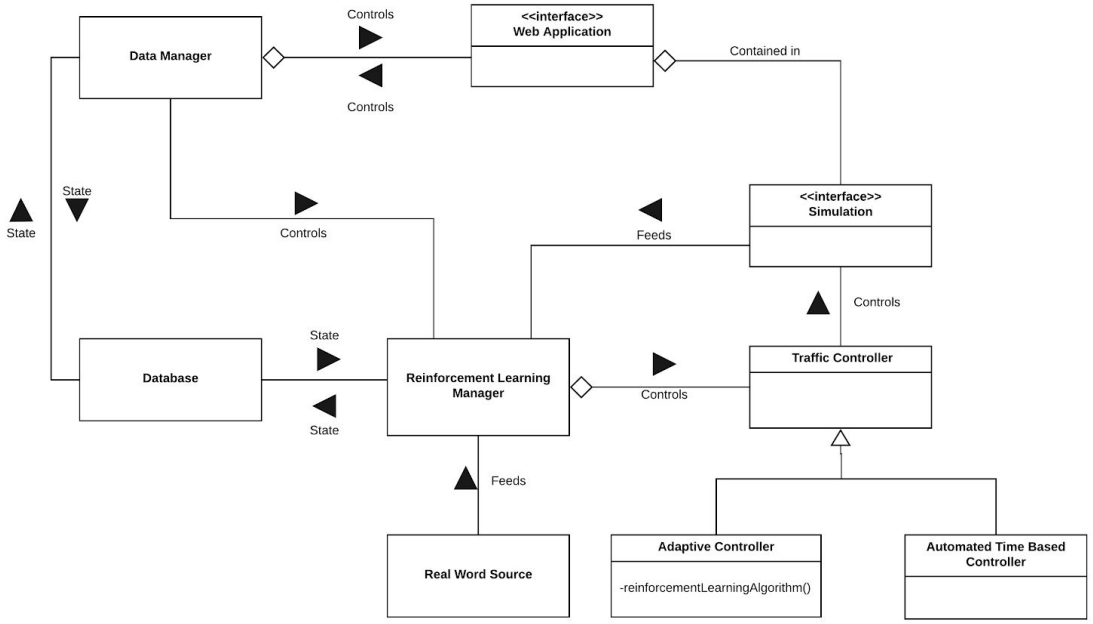

Figure 6: Domain Model of LightBot Adaptive Control System

5. Traceability Matrix

5.1 Traceability Matrix for Use Cases

Use Case / Requirement	U1	U1.1	U1.2	U1.3	U2	U2.1	U2.2	U2.3	U2.4	U2.5	U3	U3.1	U3.2	U3.3	U3.4	U4 - 4.9	U5	U5.1	U5.2	U5.3	U5.4	U5.5
R1.																	X	X		X	X	X
R1.1																	X	X		X	X	X
R1.2																	X			X	X	X
R2.	X	X	X	X	X	X	X	X	X	X												
R2.1					X		X															
R2.2					X	X		X									X					
R2.2.1					X	X		X							X							
R2.2.2					X		X								X				X			
R2.2.3					X				X													
R3.											X	X	X	X	X							
R3.1											X	X										
R3.2											X		X									
R3.3											X	X	X									
R3.3.1											X	X	X	X								
R3.3.2											X	X	X									
R4.																	X	X		X	X	X
R5.																	X	X		X	X	X
R5.1																	X	X		X	X	X
R6.													X									
R6.1													X									
R6.2													X									
R7.																X						
R7.1																X						

Figure 7: Traceability Matrix for LightBot Use Cases & Requirements

6. Architectural Design

6.1 Architectural Design Patterns

LightBot System

- The system as a whole follows a 5-Tier architecture. The corresponding layers are the Presentation, Business, Services, Data Access and Database layers. See figure 8 on page 12.
- Using this pattern will loosely couple the different components, simplifying the development and testing of the different components so that the changes have minimal impact on the other components.

Web Application & Interface.

- Patterns chosen include Data-Down, Actions-Up (DDAU) design together with Higher-Order Component design.
- Higher-Order Component (HOC) is another form of a Decorator pattern. This pattern enables code reuse, logic and bootstrap abstraction, state abstraction and manipulation as well as props manipulation. It's used to distribute complex component structure between different components and decouples the applications logic and GUI.
- The Data-Down, Actions-Up design pattern uses the HOC to accept data in its 'props' and passes the data down into the nested views and components. This pattern simplifies component reusability, separation of concerns and avoids complex data loops.

Web & Socket Server

- Patterns chosen include Model-View-Controller (MVC) in conjunction with 3-Layer (Tier) architecture- these architectures were chosen for the purpose of efficiently satisfying the core quality requirements that being scalability via modularity and security as well as other important aspects including ease of maintenance and testing.
- MVC provides the concept of code separation into
    - Models (our server performs operations on user, graph, notification and forum post models.
    - Views (our server serves the index page generated by the react application build)
    - Controllers (perform functional operations required by different components, endpoints etc)
- 3-Layer architecture also symbiotically uses the principle of **separation of concerns**

    - Controller layer (our server and socket listeners require operational/business logic to perform actions required by requesters and clients on the socket)
    - Service layer (our server provides business logic as services via utilities, services, and handlers to the controller layer)
    - Data-Access layer (our server provides this via a database connection handler as well as data model schemas which are provided as dependency injections to the services layer)
- Due to the code and purpose separation the server is extremely modular and is easily expandable to cover more uses, data model types and operations simply by adding new routes with dependency injections of the necessary services etc.
- All modular services and and utilities are asynchronous and thus can be used over multiple operational controllers providing the ability to handle large numbers of incoming requests simultaneously
- Due to the separation of concerns all the routes are easily manageable and precise security can be provided to sensitive data, access points and services on different scales according to whichever authentication / verification utility is needed
- Rate access limiting, Xss protections, parameter pollution filtrations, bottlenecking etc are applicable to prevent attacks of scale on the system as it is imperative for it to remain accurate to prevent sensitive data theft, traffic manipulation and collisions
- The modularity provided by these architectures also provide for easy familiarity as they are widely known and thus enables people who are new to the system to quickly gain an understanding of how it works as well as simplifies extensions/corrections that maybe required whilst maintaining the system
- The modularity also provides an easier interface for testing and deployment thus unit testing and deployment can easily be managed such that tests can be run on different aspects of the system by directly testing business logic in services individually as imports, testing data type viability or testing routing controllers or the entire server as a whole (the socket and server are provided as dependency imports and can be exported to testing environments if need be)

Simulation and Simulation Server

- For the simulation and simulation server we used a client-server architecture. Since our simulation program, SUMO, does not make provisions for viewing simulations in a web browser, and a core requirement of our app is that it have a web based interface, a client-server architecture is ideal for our purposes, as it was the only solution to viewing our simulations in a web browser. The web app acts as the client, and all interaction with lightbot will be facilitated by the web app. We set up a remote server which will run our simulation software, SUMO and SUMO-WEB3D.

- The client-server architecture allows the use of the simulation software without needing to install said software on the local machine.
- The client-server architecture makes Lightbot very scalable, and improves availability as expensive processes are centralized, providing a single location for upgradability and performance for one or more clients. It allows simulation servers to be added or upgraded, if different cities are to implement Lightbot.
- The client-server architecture does have a drawback in that it has a single point of failure, the server. However if a failure does occur it will be easier to fix, especially if multiple clients interact with Lightbot.

Reinforcement Learning Server

- The patterns used include a modified version of the Model-View-Controller (MVC) such that the View element is not used so it follows a Model-Controller (MC) pattern.
- Another pattern used is the Client-Server pattern, which facilitates communication between the Controller of the Reinforcement Learning Server and the Web & Socket Server.
- MC owing to its ability to abstract the functionality into separate elements:
    - Model: used by the Controller to maintain the current state data of the Reinforcement Learning Server.
    - Controller: defines the functionality of the server, while also maintaining instances of the Model.
- Client-Server pattern to initiate interactions between Web & Socket Server and the Reinforcement Learning Server and to allow services to be invoked and results to be sent from either side.
    - Client: the Reinforcement Learning Server will act as the client, while hosting the functionality for the server to access.
    - Server: the Web & Socket Server will provide the port for which the Client can connect.
    - Request/reply connector: will facilitate bi-directional communications between the Server and the Client.
- The modularity of the Model and Controller, makes it very easy to add additional functionality to the Controller and more sophisticated data structures in the Model as needed.
- One of the weaknesses of the MVC is it's inability to use certain user interface toolkits however this is mitigated in the sense that the View element is not implemented in the modified MC pattern so will not require such toolkits.
- Some of the benefits of the Client-Server pattern are in its scalability. It would require very little extra work to facilitate a few more extra Clients.
- One of the weaknesses is in that the Server will create a single point of failure, however this can be mitigated if extra attention is given to that single point rather than less attention to multiple points.

6.2 Architectural Design Diagram

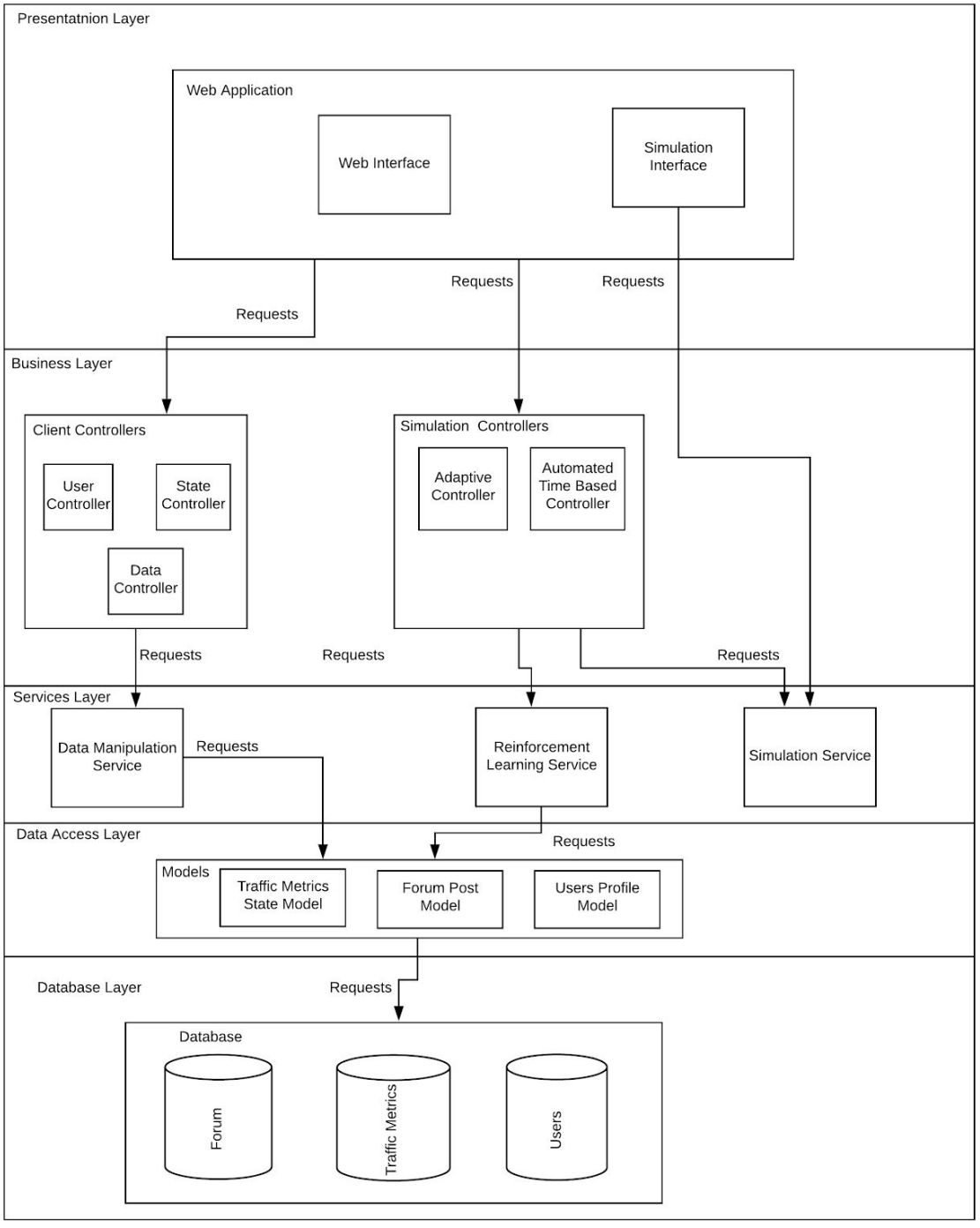

Figure 8: Architectural Design Diagram of the LightBot System

7. Non-functional Requirements

7.1 Quality Requirements

Performance

- The system needs to process data as fast as possible, otherwise this could lead to delays in traffic.
- The response time of the system will be measured according to its real-time data processing and feedback to decide how to change the state of the traffic lights in order to maximize the flow of traffic at all times. This includes measuring the execution speed of the GUI, simulation and database at the same time.
- No outside users will have access to this system. The capacity of the system is justified by the amount of users that the system can support at once, where the amount of users at this time is fairly low. The only users to account for in this sense, would be the users from 5DT that would manage the system.
- The average page load time will be used to test the latency of the system between the database, servers, simulation and web application. Our aim is to reach 500 milliseconds.
- The response time of the web application is expected to be one-second, which is the maximum limit set.
- The application server should be able to handle a rate of 100 requests per IP per 15 minutes.
- To ensure that the simulation is responsive for all clients, it has been implemented using a client-server architecture. This ensures that all clients will have the same experience with the simulation, irrespective of their hardware specification.
- The client-server architecture also allows us to improve performance of the simulation by upgrading server hardware, resulting in a performance enhancement for all clients.

Reliability

- The system needs to consistently provide accurate output under certain conditions.
- The system needs to ensure that vehicle metrics, such as acceleration, are as accurate as possible so that the system operates as intended.
- The accessibility to the database, servers and simulation will be tested and monitored during different types of network connections to verify that the test systems respond correctly.

- The system will make use of a 5-Tier architecture, which will be used to achieve separation of concerns and ensure that the components are specialised to perform a certain number of functions. This will enable us to quickly identify problematic components and fix said problems.

Security

- The system should only be accessible by the intended users and no outside users.
- The system will grant each user secure access to the GUI, where each users' login data will be encrypted and stored securely on the Management database.
- Hashing and salting of login and register passwords will occur and be saved in the database.
- The system will adhere to having a secure connection over the internet, as it needs to run on a web browser on a remote machine. As well as for server connection and API calls made.
- The system will be designed to detect DDoS attacks as well as prevent SQL injections.
- Role-based access control is used.
- Web tokens for the communication between the web application on the web server will be implemented. This encrypts the contents of a user session and stores them directly in a cookie.
- The application server will have a rate limit of about 100 requests per IP per every 15 minutes to prevent attacks, such as DDoS.
- The 5-Tier architecture of the system allows us to control data access across different tiers to ensure that sensitive data is accessed only by authorised users.

Maintainability

- The system should be easy to update and maintain. If the layout of an intersection needs to be edited, the system will be able to accommodate such changes to the simulation.
- The dependability of the different components in the system will be kept minimal as excessive dependability has a negative impact on maintainability.
- The different components of the system will be kept separate in design, to ensure that future updates and maintainability of different modules can be accommodated.
- The simulation aspect of the system and its use of a client-server architecture provides a single point of failure, for multiple clients. This enables us to improve uptime of the simulation for multiple users, by maintaining and monitoring a single server.
- The reinforcement learning service makes use of the MVC pattern. MVC's modularity, of the Model and Controller, makes it very easy to add additional

functionality to the Controller and more sophisticated data structures in the Model as needed.

Usability

- The system should be user-friendly, accommodating for all intended users.
- The amount of interaction with the system from a user to accomplish a certain task will be kept minimal. This will be accomplished by taking care of where design elements are placed and the layout of the dashboard with the relevant requirements in place.
- One way we will be enhancing the usability of the system is to give the user feedback as to how the system is operating and allowing the user to make appropriate responses. An example is displaying to the user that the simulation server is loading on the web application, asking the user to be patient.
- The simulation gives the user the options to pause, resume and reset the simulation interaction.
- The 5-tier architecture provides the user with a simple graphical interface to use Lightbot, while complex processes are performed in the background. It also negates the need for the user to install anything programmes on their local machine. This allows Lightbot to have a large potential user base, as only basic computer skills are required to interact with the system.

Scalability

- The system should be designed such that it will be able to operate on multiple intersections in the future, not just one. Also, more user features should be easy to add to the system.
- This is done through the use of an object store database, which allows the user to make many changes in a short period of time. It allows for scalability without increasing layers of complexity to the database.
- The web server sends the information to the users through the web application, which is set up internally. To prepare for user traffic load, the system is designed for twice as many users as the user base.
- The system will make use of a microservices architecture to ensure that resources can easily be configured and then deployed.
- The client-server architecture of the simulation, allows Lightbot to be scaled to be implemented at multiple intersections, as servers can be upgraded or added to handle increased performance requirements of the larger system.

Availability:

- The system needs to be constantly running throughout the duration of a user's session, so as to update the traffic at all times.
- The system is expected to be at least 99% uptime.
- This component will have a combination of Passive Redundancy and Cold Spare tactics to maintain its availability.
    - This combination is achieved with the Model element of the component maintaining checkpoint data of the component's state, relieving the need to organically rebuild the state data, this has already been achieved.
    - However, a Cold Spare means that the redundant copy of the component will not be running, to save processing power, and will have to be manually booted, this will only be implemented once on a remote server.
- The availability of the Web Server is independent of the Reinforcement Learning Server's availability.
- In terms of quantification for the Reinforcement Learning Server, we can only estimate what the time may be for the moment.
    - For the Mean Time to Repair, the component is still relatively small so the time could be close to a minute, with most of the time needed to manually reboot the component and the rest of the time used for retrieving the checkpointed state.
    - For the Mean Time Between Failure, since the component is still in early development, a value for the component's up time in testing could be estimated at about an hour.
    - The availability for the component could be: 60min/(1min + 60min)*100 = 98.63%. However, this value is highly likely to change.
- The Client-Server pattern is used to initiate and maintain a constant connection between Web Server and the Reinforcement Learning Server and to allow services to be invoked and results to be sent from either side.

7.2 Architectural Constraints

System Constraints

- The GUI must be able to run in a web browser on a remote machine.
- The system must be designed using open source software.

8. Technology Requirements

8.1 Web Application & Interface

- **React:** React, which is also known as ReactJS, is an open-source JavaScript library used to build the web application's interface. We used it to build a two-page interface application, where it consists of the login page and the dashboard page. React Components are used to interchange the data on these two pages. React supports the creation of large web applications that can change data without having to reload the page. We decided on React as its main purpose is to be fast, scalable, and simple to implement.

- **React Redux:** Redux is a predictable state container for Javascript applications. It maintains the state of an entire application in a single object which can't be changed directly. The reason why we implemented it is that it helps the web application behave consistently, allows it to run in different environments (client, server and native) and simplifies the application testing.

- **SCSS:** SCSS, also known as Syntactically Awesome Style Sheet, is a superset of the known CSS used in front-end development. It's technically a more advanced version of CSS. The reason why it was implemented is that SCSS supports a multitude of advanced features and offers variables, which allows us to shorten our code.

8.2 Web Server

- **NodeJS:** is an open-source, cross-platform, JavaScript runtime environment (Framework) that executes JavaScript code outside a web browser. This gives us ability to use node modules (packages installed via node package manager) which provides a vast array of utility for developing and testing scalable and secure web server applications

- **Express:** is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications. It provides the ability to add middlewares as well includes a myriad of its own as well as all the necessary Http utility methods. This is necessary for implementing a simplified MVC 3 Layer server providing all the security and utility middlewares such as rate limiting cors blocking xss blocking via extensions as well as routing management methods.

- **Mongoose with MongoDB:** mongoose is a schema-based solution to model application data. It includes built-in type casting, validation, query building,

business logic hooks etc. This is necessary for creating the various data models our server requires regarding client , notification, forum, graph, state and other types of information for accuracy verification and storage as well as provide a form of relational links that is often not included in the NoSQL type databasing solutions like MongoDB

- **SocketIO:** The de facto socket server solution package which provides the ability to easily attach a socket server to our express server via a dependency import and also which is completely customizable with lobby control and direct conversations.
- **Nodemailer:** An SMTP mailing package which allows our server to provide password reset tokens to users.
- **BcryptJS:** Bcrypt is an extremely powerful password hashing technique which uses secure random numbers (primes) in order to encrypt passwords. BcryptJS is a zero dependency library that whilst being slightly slower has the ability to hash larger input lengths providing a more secure hashing method
- **jsonwebtoken:** is an Internet standard for creating data with optional signature and/or optional encryption whose payload holds JSON that asserts some number of claims. The tokens are signed either using a private secret or a public/private key. With the jsonwebtoken package we enable our selves to sign an auth key which can then be used later by the user to make requests of private access type

8.3 Simulation and Simulation Server

- **SUMO:** Simulation of Urban Mobility, SUMO, is an open source traffic simulation package. It allows the creation of road networks, and accurate replication of real world traffic on these networks. It works with a variety of interfaces that can remotely control and tweak the simulation. We are using SUMO as the basis of the simulation, as it accurately models vehicle parameters such as speed, acceleration and braking in traffic conditions. It allows dynamic control of intersections through an interface called TraCI, which is particularly important as it will allow us to have multiple controllers for the intersection's traffic lights.

- **SUMO-WEB3D:** SUMO_Web3D enables the visualization of a SUMO simulation in a Web Browser. We used this application as it allows us to build a client-server architecture allowing the user the ability to view SUMO simulations without installing it on their local machines. SUMO-Web3D gives Lightbot greater portability as it allows interaction with Lightbot through any device with a modern web-browser.

8.5 Reinforcement Learning Server

- **Python 3:** Python is an interpreted, high-level programming language that supports both Object-oriented programming and structured programming. Python is a general- purpose programming language but has been designed to be highly extensible. It's use and proficiency in machine learning algorithms made it a good candidate to code our swarm reinforcement learning algorithm and provide functionality around it.
- **Pymongo and MongoDB:** One of Python's many libraries to facilitate data communication to our MongoDB database.
- **SocketIO:** A websocket package used to facilitate real time connections between a client and a server. This bi-directional communication can include predefined events that can be sent across the connection along with data, most common is JSON data.

9. References

Kung, D. C., 2014. *Object-Oriented Software Engineering: An Agile Unified Methodology*. New York: McGraw-Hill.