Software Requirements Specification V.4

COS 301 Capstone Project: Demo 4

Members:

Maria Petronella Laura-Lee Strydom *u04974359*

LightBot Adaptive Traffic Control System

Contents

1. Introduction

1.1 Purpose

The purpose of LightBot is 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 fuzzy logic 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 scope of the system would be to provide simulations to demonstrate the effectiveness in the case of the implementation of a fuzzy logic algorithm for the traffic lights of a given intersection layout. This 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. MVP - Model-View-Presenter

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. Finally, any works cited are referenced in Section 7.

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) 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 an administrator 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 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 fuzzy logic algorithm.
V. As an administrator of the system, I would like to be able to download the simulation software onto my remote desktop so that I can access the simulation from anywhere.
VI. As an administrator of the system, I would like the system to simulate a real-world intersection, so that I can see the benefits of having the fuzzy logic 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 run on a Fuzzy Logic Algorithm for the automatic simulation of an intersection.
    - 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 through the database system itself, and not on the web application.

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 Log out of system
- U3: System Management
    - U 3.1 Manual control of traffic simulation
    - U 3.2 Automated control of traffic simulation
    - 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 users
    - U 4.8 View user role
    - U 4.9 Edit user role
- U5: Simulation Control
    - U 5.1 Re-create real-world scenarios through input parameters
    - U 5.2 View scenario results
    - U 5.3 Test situational scenarios

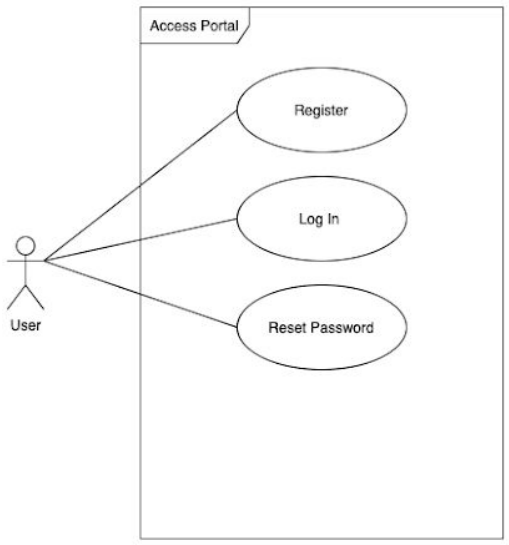

Figure 1: Use Case 1 Diagram

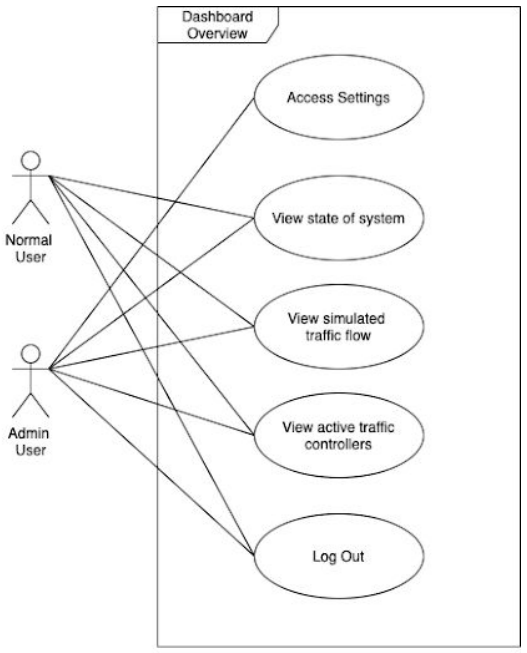

Figure 2: Use Case 2 Diagram

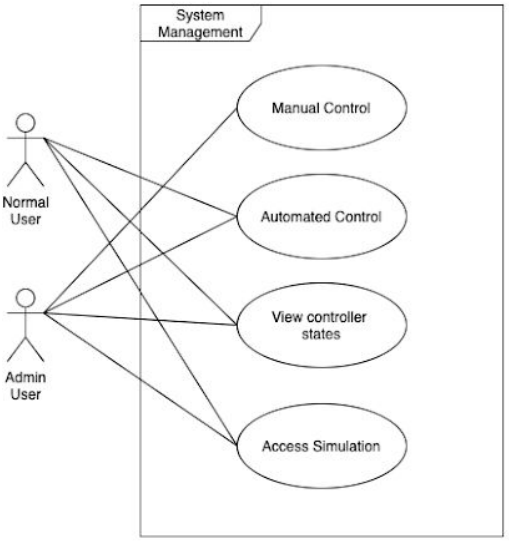

Figure 3: Use Case 3 Diagram

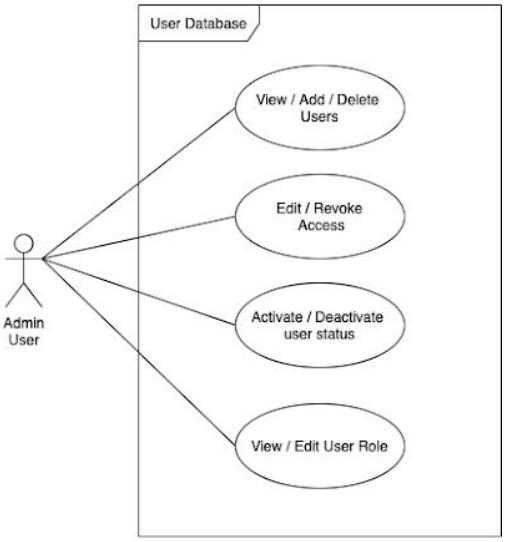

Figure 4: Use Case 4 Diagram

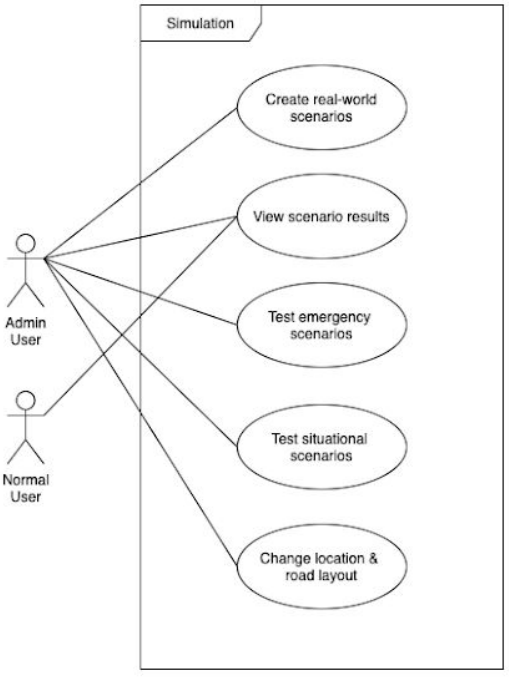

Figure 5: Use Case 5 Diagram

4. Domain Model

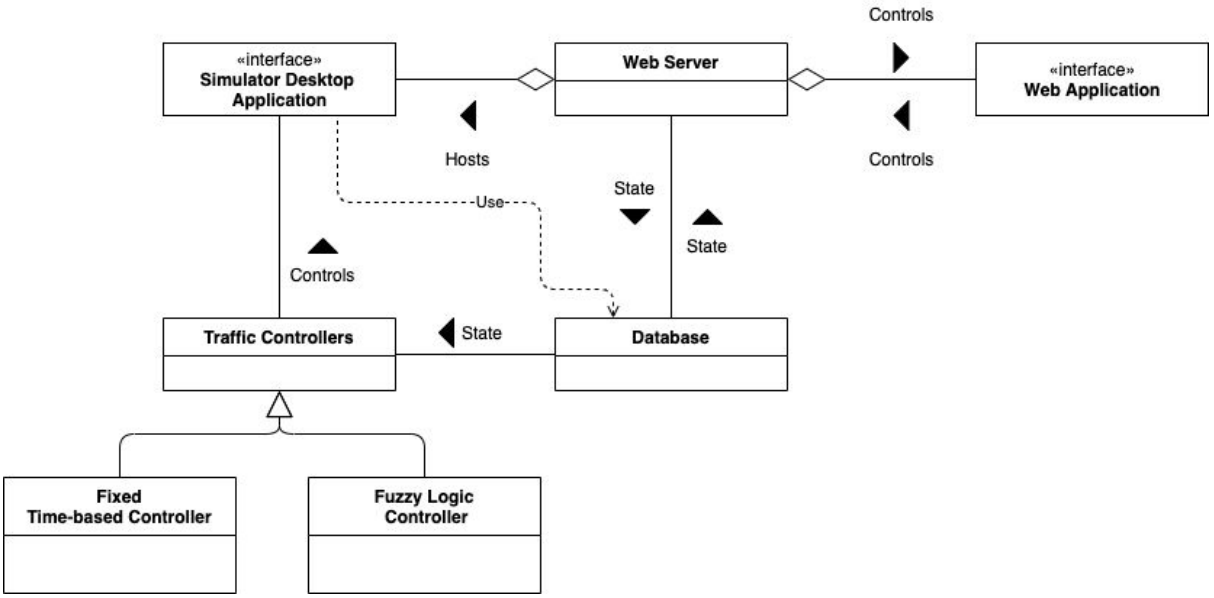

Figure 6: Domain Model of LightBot System

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 high-level architecture that is used for the LightBot system is a client-server architectural pattern. The system allows the clients to interact by requesting services of the server, which provide a set of services. This architecture was chosen to promote modifiability and reuse, by factoring out common services and having to modify these in a single location. Improvement of scalability and availability is achieved by centralising the control of these resources and services, while distributing these resources across a physical server.
- The Client is a component that invokes the services of the server component. The first client is the LightBot Web Application, with which the users interact on a web browser. The second client is the LightBot Simulator Desktop Application, which is an application that runs on a local machine. From the LightBot Web Application and LightBot Simulator Desktop Application, HTTPS responses will be sent between these clients and the LightBot Web Server. This will then in turn invoke a response from the server.
- The Server is a component that provides services to the clients. The LightBot Web server is the Server component in this architecture.
- Another important element of the Client-server architecture, is the use of a Request/Reply connector, which is a data connector employing a request/reply protocol, used by a client to invoke services on the server.
- As the server can be accessed by any number of clients, in this case 2 clients, it's easy to add more clients to the system.

LightBot Web Application

- Patterns chosen include Data-Down, Actions-Up (DDAU) design together with Higher-Order Component design.
- The above mentioned patterns are incorporated into the MVP (Model-View-Presenter) architecture, which is used for building the user interface.
- 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.
- MVP is a user interface architectural pattern that facilitates automated unit testing as well as improving the separation of concerns present in the Presentation layer found in the LightBot Server.
    - The Model is the interface that defines the data to be displayed or acted upon in the web application user interface.
    - The View is the passive interface that displays the Model and routes user commands to the Presenter to act upon that data.
    - The Presenter acts upon the Model and the View. It retrieves data from the Model and formats it for display in the View.

LightBot Web Server

- The patterns chosen for the server is a 2-Tier architectural pattern. This architecture was 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. Due to the simplicity of this architecture, there is no need for a 3rd tier.
- 2-Tier architecture also symbiotically uses the principle of **separation of concerns**
    - Presentation layer (the server listener require operational/business logic to perform actions required by requesters)
    - Data 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. This layer also provides business logic as services via utilities, services, and handlers to the controllers)
- 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 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.
- 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.

LightBot Simulator Desktop Application

- For the simulation I used a Request Response architecture, more commonly known as the message exchange pattern, in which a requestor sends a request message to a replier system, which receives and processes the request, returning the required data as a message in response.
- The request-response is implemented asynchronously, with a response being returned at some unknown later time. The usage of an asynchronous request-response comes involves the following two participants:
    - The Requestor - This is the initiator of the communication. It sends the request message and waits for the response message. This will be the LightBot Simulator Desktop Application.
    - The Provider - This participant waits for the request message and replies with the response message. This will be the LightBot Web Server.
- By using this pattern, it invokes the advantage where the server can easily be increased in scale when there is an increase of data or user load.

6.2 Architectural Design Diagram

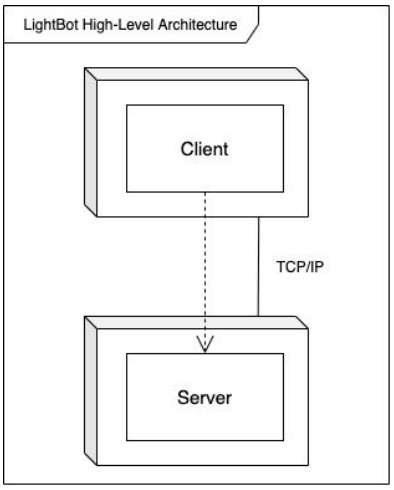

Figure 8: High-Level Architectural Design Diagram

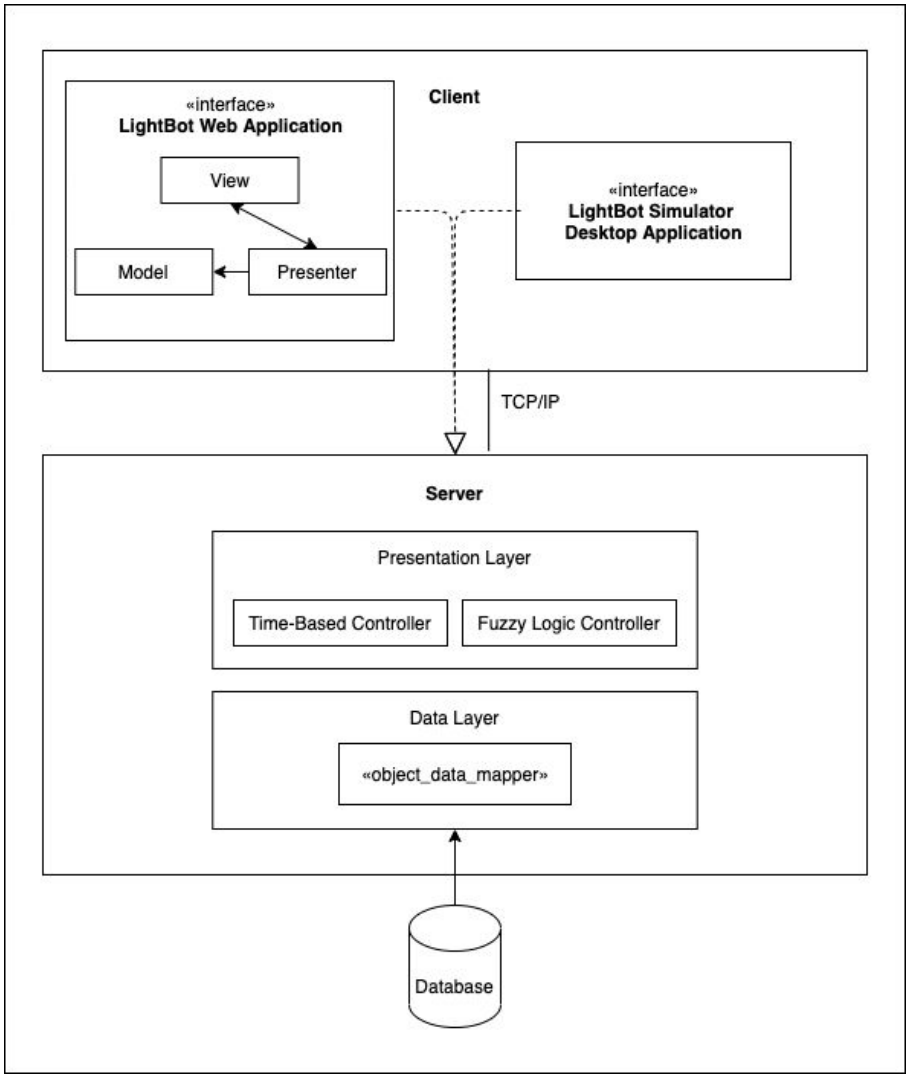

Figure 9: LightBot Architectural Design Diagram

6.3 Deployment Model

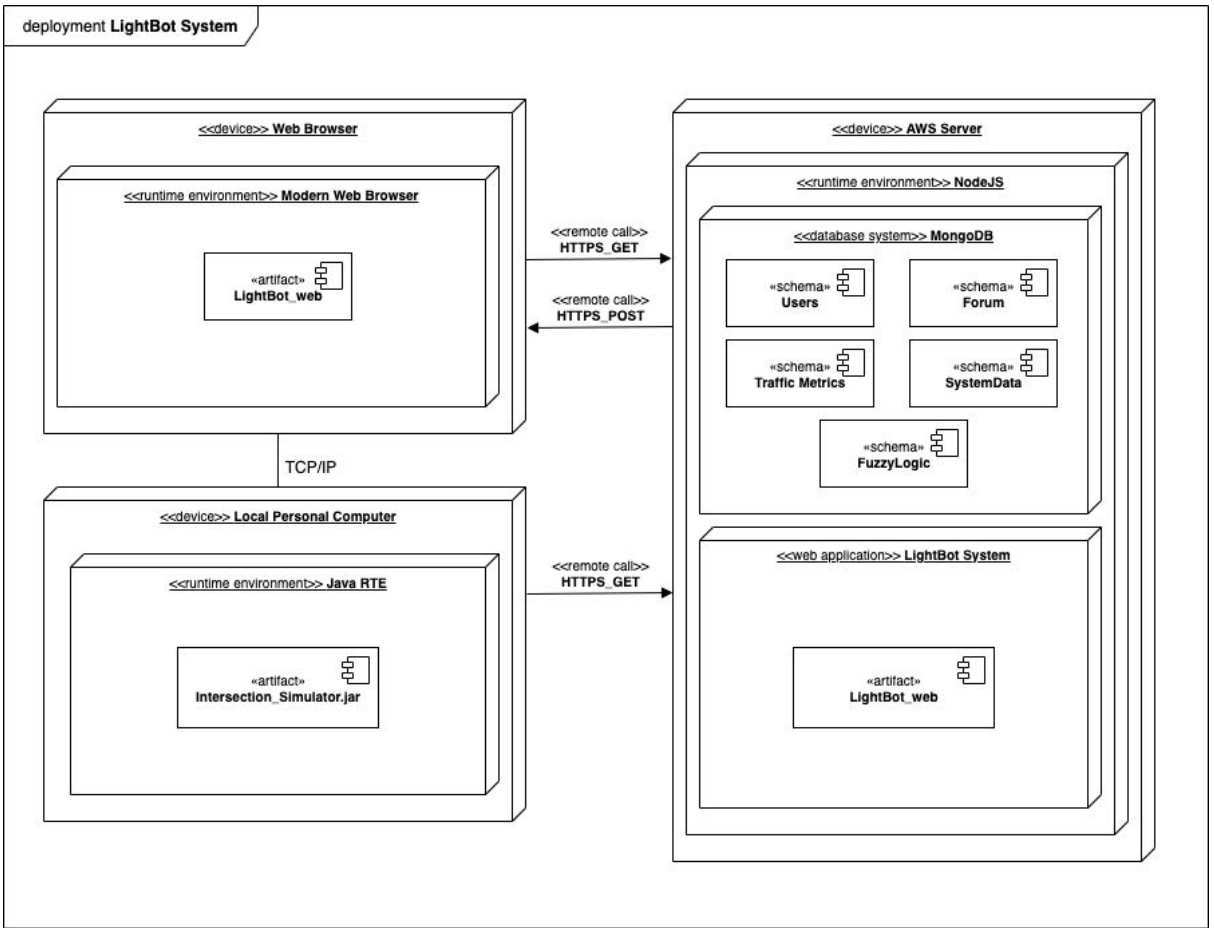

Figure 10: Deployment Diagram

7. Non-functional Requirements

7.1 Quality Attributes/Requirements

Usability

- The amount of interaction with the test 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 to enhance 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 view also gives the user the options to pause, resume and reset the simulation interaction.

- The Model-View-Pattern pattern makes it easy to provide multiple views of the data, which supports user initiative and feedback.
- To quantify the usability of the system, it can be measured using a number of observable and quantifiable metrics that overcome the need to rely on simple intuition' (Justin Mifsud n.d.). There are 3 main metrics to calculate:
    - Completion Rate: The below calculation is used to calculate the effectiveness of the system. As stated by Mifsud, the average task completion rate is 78%, but the aim for this system is no less than 80%.
        - Effectiveness = (Nr. of tasks completed successfully / Total nr. of tasks undertaken) * 100%
        - Aim for the Lightbot systems' effectiveness = (4 / 5) * 100% = 80%
    - Overall-relative efficiency: The efficiency of the system can be measured in terms of task time, which is the amount of time (in seconds) it takes the user to complete a task. As this is a rather complex calculation that needs to be based on more than approximately 2 or more users completing tasks, the average amount of seconds that is needed to complete a task will be taken into account, which is between 1 - 6 seconds.
    - User satisfaction: This can be measured through standardized satisfaction testing, usually in the form of questionnaires, which can be given to a user after system testing. These tests can measure how difficult it was to complete a task, if the user understood the layout of the system and how they would rate the system based on different aspects.
        - The standardized questionnaire that can be used to calculate the user satisfaction of the system, is the SUS (System Usability Scale) questionnaire, which consists of 10 questions.
        - The desired outcome of the results is 80.3 or higher out of a score of 100. This means that the system has a high user satisfaction score.

Security

- The test system will grant each registered user secure access to the dashboard, where the user's login data (email and password) will be encrypted and stored securely on the User database. New users' data will be stored in the same way.
- Hashing and salting of login and register passwords will occur and be saved in the User database.
- The web application will adhere to having a secure connection over the internet, as the test system needs to run on a web browser on a remote machine. As well as for server connection and api calls made.
- Role based access is used to access the web application.

- I also implemented web tokens for the communication between the web application and the web server. It encrypts the contents of a user session and stores them directly in a cookie.
- A broker pattern can be utilized when clients want to gain access to their profile on the web application. This can be supported through a client-server style architecture to carry out the view and interaction of the simulation, view and interaction of the data in the overview, forum and user profile components.

Performance

- The average page load time will be used to test the latency of the system between the database, server, simulation and web application. The aim is to reach 500 milliseconds.
- 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.
- The response time of the web application is expected to be 3 seconds, which is the maximum limit set.
- The capacity of the system is justified by the amount of average/maximum 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. No outside users will have access to this system.
- The simulation needs to ensure that simulated vehicle metrics, such as acceleration, are as accurate as possible so that the statistics of the optimized traffic light system have real world value.
- The Apdex score is used to calculate and track the relative performance of an application. It is done by specifying a goal for the amount of time it takes a specific web request or transaction to be completed. It gives a score from 0-1, where 1 is the best possible score, where a 100% of the response times were satisfied.
    - Apdex = (SatisfiedCount + (ToleratingCount / 2)) / TotalSamples
    - The aim of the systems Apdex score is to be at least 0.8

Scalability

- The scalability depends on the capability of the system to increase in size when the users demand expansion of different functionalities and the capacity of the test 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 serves the information to the users through the web application, which is set up internally. To prepare for user traffic load, I planned for twice as many users as our user base. I plan on routing new requests to the server with the least load. This ensures that each request receives a response as quickly as possible.
- The system need not cater for more than 10 users accessing information at once, as a limited set of users will be introduced to use this system at 5DT.
- Scalability of the system is more important when it comes to the simulation of traffic light intersections, as the amount of intersections, cars, traffic metric data, etc. can increase phenomenally. It should be scalable to allow for the addition of up to 4 additional intersections and up to 1000 cars when running and testing the simulation.
- The Model-View-Controller pattern supports loose coupling between components; as this results in ease of modification, future development and increasement of system size is possible.
- To calculate the scalability of the system, the following techniques can be executed:
    - Testing the response time of the application - it can be measured by the amount of time it takes from the second the user enters the URL into the browser up until the application is done loading the content. To which the set time is 3 seconds.
    - Creating a test environment which is stable enough to withstand an entire scalability testing cycle.
    - Test data load levels in different simulation scenarios.

Availability:

- The availability of the system is measured through the probability that it does not fail under certain circumstances or undergoing maintenance when it needs to be up and running.
- The LightBot dashboard and system controllers are dependent on the database, meaning that the data needs to be available and accessible at all times.
- System availability can be calculated by the following:
    - Availability = Uptime / (Uptime + Downtime)
    - If the system is expected to run for 200 hours in a month (5 hours a day for 20 days), be down for approximately 10 hour in the month for updates and maintenance, the availability would be calculated as follows:
        - Availability = 200 / (200+10) = 0.952 = 95.2%
- Implementing a design pattern to handle error catching and handle dependency failure will increase the performance and availability of the system. One of the

patterns that handle dependency failures is the Circuit Breaker pattern. This pattern reduces the impact which such failures can have on this system. This is accomplished by 'giving up' on a dependency when an error occurs, and waits for that dependency to recover before trying again.

7.2 Architectural Constraints

System Constraints

- The only constraint is that the GUI must be able to run in a web browser on a remote machine.

8. Technology Requirements

8.1 LightBot Web Application

For the development of the LightBot web application, I narrowed it down to three different platforms/frameworks that would work for this applications' requirements.

1. First off, I researched AngularJS. The reason why I choose AngularJS first is that it's a powerful JavaScript based framework that is considered to be the "best Front-end framework for a web application" (Patel, 2020). The main benefits include the multiple built-in features and benefits (easy setup and deployment, server-side rendering and the CLI Builder, to name a few) that it offers as well as its universal recognition. The only problem that I encountered with AngularJS is that it implements a Model-View-Controller pattern. I decided against AngularJS, as I needed a framework that would let me implement a Model-View-Presenter pattern.
2. I moved on to researching ASP.NET. It is very useful for creating dynamic and scalable web applications, and only requires .NET to extend the developer platform. The main reason that I considered ASP.NET is that it enables bi-directional communication between the server and client in real time. As the LightBot System's high-level architecture is a Client-Server pattern, the web application would greatly have benefitted from it. ASP.NET supplies libraries for common web patterns, such as the MVC pattern, and as my requirement, the MVP pattern. The cost of using ASP.NET is very high (this includes the setup of a .NET development base together with the additional licenses for server-side software), and that kept me from deciding against it, as this project cannot use tools that aren't free-to-use or open-source.
3. The last option I chose was React. React is an open-source JavaScript based library that is also used to create web applications. Most importantly, it implements the MVP pattern. Another benefit is that it's component based, meaning the whole web page will be divided into separate components which can be coded individually and re-used right across the application. React supports the features that are needed for the development of the web

application and suits my requirements the best. And so I chose React for the front-end development.

The following additional libraries where chosen and implemented as they complemented the React library:

- **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 I 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 me to shorten the code.

8.2 LightBot Web Server

For the development of the LightBot web server, I narrowed it down to two different languages, as they are the languages that I am more familiar with when it comes to server-side development.

1. The first is PHP. It is a server-side scripting language, meaning that it interprets script on the server side rather than on the client side. This in turn provides each user with a customized interface. This is a great feature, but not necessary for the application that needs to be developed. As PHP is a highly flexible and well-maintained and supported language for server side development, the fact that "bad" packages exist and that it is a little too forgiving of errors, I didn't want to create the opportunity of implementing bad solutions creating security risks.
2. Alternatively I moved towards NodeJS. It is an open-source, cross-platform, JavaScript runtime environment (Framework) that executes JavaScript code outside a web browser. It provides the ability to run JavaScript code server-side. Meaning that I can use JavaScript for front-end rendering and then reuse it in the back-end. It also comes with the default package manager NPM. This gives me the 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. Hence, I chose to use NodeJS for server-side development.

The following packages where chosen and implemented as they were needed to extend the use of NodeJS:

- **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.

- **Mongoose with MongoDB:** Mongoose is a schema-based solution to model application data. It is implemented as the Object Data Mapper. 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 for accuracy verification and storage as well as providing a form of relational links that is often not included in the NoSQL type databasing solutions like MongoDB. An SQL database, like MySQL, was considered at first, but I decided against it as I needed a database that supports unstructured data and is horizontally scalable, which means that any amount of servers can be added later on if needed. MySQL would require predefined schemas to determine the structure of the data, causing a multitude of preparations.

- **Nodemailer:** An SMTP mailing package which allows the 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:** It's 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, I enable users to sign an auth key which can then be used later by the user to make requests of private access type.

8.3 LightBot Simulator Desktop Application

- **Intersection Simulator:** This is an open source traffic simulator application. It allows the user to simulate traffic flow through the manual control of input parameters. I am using the Intersection Simulator as the basis of the simulation, as it accurately models vehicle parameters such as speed, average waiting time in seconds and braking in traffic conditions. It allows manual control over the road layout of the intersection and gives the user the ability to change the following conditions:
    - The length of the green light of every directions' traffic light in seconds.

    - The length of the green light of a traffic light for turning in seconds (only if there are two output tracks and it is a type 1 track)
    - The length of an additional arrow for turning right (last n seconds of a red light)
    - The probability for going in other directions.
    - The density of the cars per minute.
    - The number of input and output tracks in some direction.
    - The type of output tracks, which can be different directional lanes.
- This application also allows the user to export simulation results, export parameters used for each simulation or to import pre-determined parameters.
- I will be adding on to this simulation project, where I will implement a Fuzzy Logic algorithm to create the automatic simulation of an intersection.
- This project is written in Java, and the interface of the GUI is Java AWT.
- The open source project can be found at this link: https://sourceforge.net/projects/traffic-simulator/
- The author of this project is Krešimir Kovačić.

As an individual developing this project, I have chosen to use an open-source traffic simulator application as I have no prior background knowledge in the field of engineering or artificial intelligence to contribute to and develop a traffic intersection simulator on my own. I have done research on a lot of simulators and have chosen the one developed by Krešimir Kovačić as I have reviewed his code and feel fairly confident with it. The results generated by his simulator are needed as functional requirements specified, and I am assured that I am capable of altering his code without breaking the build. I have chosen to improve on his application by adding a controller based execution depending on the user's choice of controller. I will be adding a Fuzzy Logic algorithm that will automatically run the simulation, as the Fixed Time-Based controller already provides the manual override of the simulation.

9. References

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

Patel, R., 2020. 7 *Best Platforms To Build Web Applications*, viewed 2 September 2020, <https://www.crestinfotech.com/7-best-platforms-to-build-web-applications>

Justin Mifsud n.d., *Usability Metrics - A Guide To Quantify The Usability Of Any System*, viewed 10 August 2020, <https://usabilitygeek.com/usability-metrics-a-guide-to-quantify-system-usability/#:~:text=The%20ISO%209241%2D11%20standard,a%20specified%20context%20of%20use%E2%80%9D.>