

Software Requirements Specification V.3

COS 301 Capstone Project: Demo 3

Members:

Maria Petronella Laura-Lee Strydom

u04974359

LightBot Adaptive Traffic Control System

Contents

| | |
|---|-----------|
| 1. Introduction | 2 |
| 1.1 Purpose | 2 |
| 1.2 Scope | 2 |
| 1.3 Definitions, Acronyms and Abbreviations | 2 |
| 1.4 Overview | 2 |
| 2. User Characteristics | 3 |
| 2.1 User Stories | 3 |
| 3. Functional Requirements | 3 |
| 3.1 Requirements & Constraints | 3 |
| 3.2 Use Cases & Diagrams | 4 |
| 4. Domain Model | 7 |
| 5. Traceability Matrix | 8 |
| 5.1 Traceability Matrix for Use Cases | 8 |
| 6. Architectural Design | 9 |
| 6.1 Architectural Design Patterns | 9 |
| 6.2 Architectural Design Diagram | 12 |
| Data Flows | 13 |
| Diagram | 13 |
| 6.3 Deployment Model | 13 |
| 7. Non-functional Requirements | 14 |
| 7.1 Quality Attributes/Requirements | 14 |
| 7.2 Architectural Constraints | 16 |
| 8. Technology Requirements | 16 |
| 8.1 Web Application & Interface | 16 |
| 8.2 Web Server | 16 |
| 8.3 Simulation and Simulation Server | 17 |
| 8.5 Reinforcement Learning Server | 18 |
| 9. References | 18 |

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. 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. 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 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 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 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

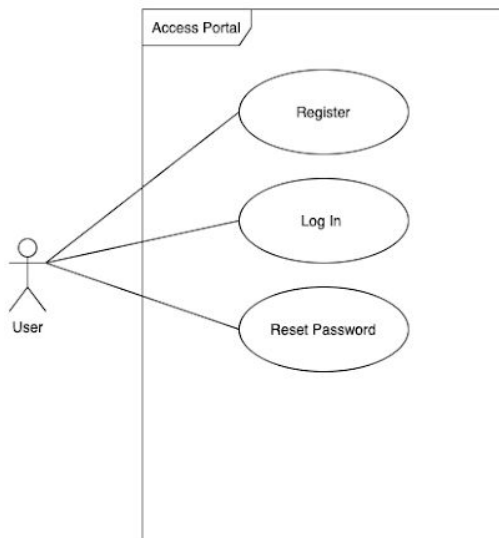


Figure 1: Use Case 1 Diagram

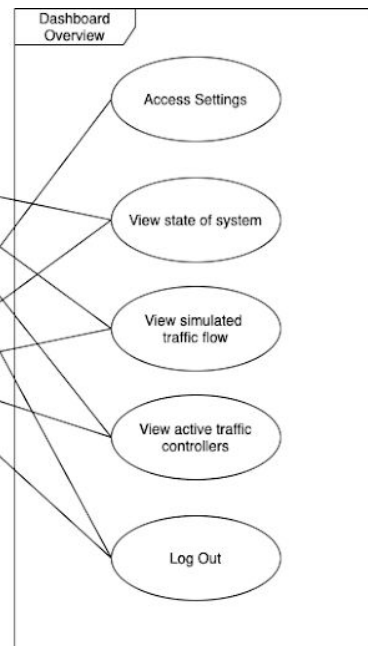


Figure 2: Use Case 2 Diagram

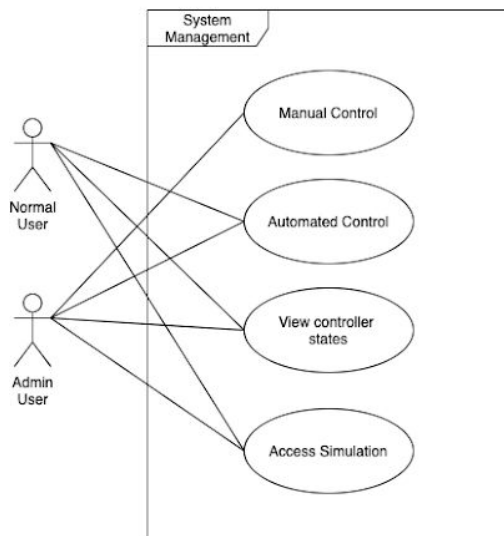


Figure 3: Use Case 3 Diagram

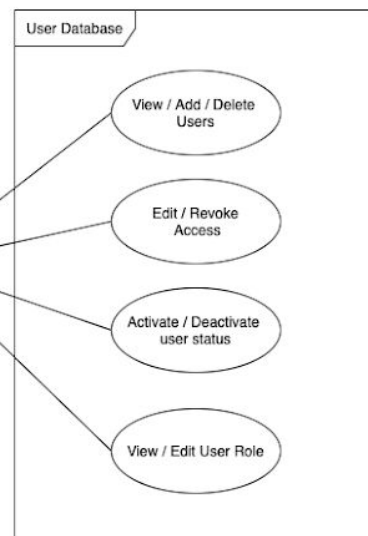


Figure 4: Use Case 4 Diagram

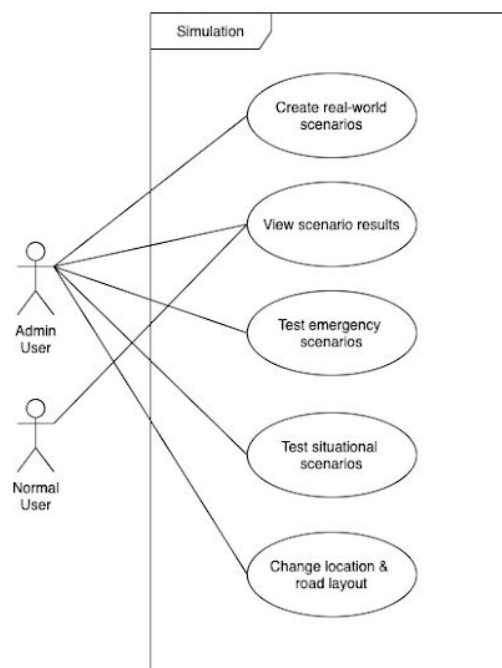


Figure 5: Use Case 5 Diagram

4. Domain Model

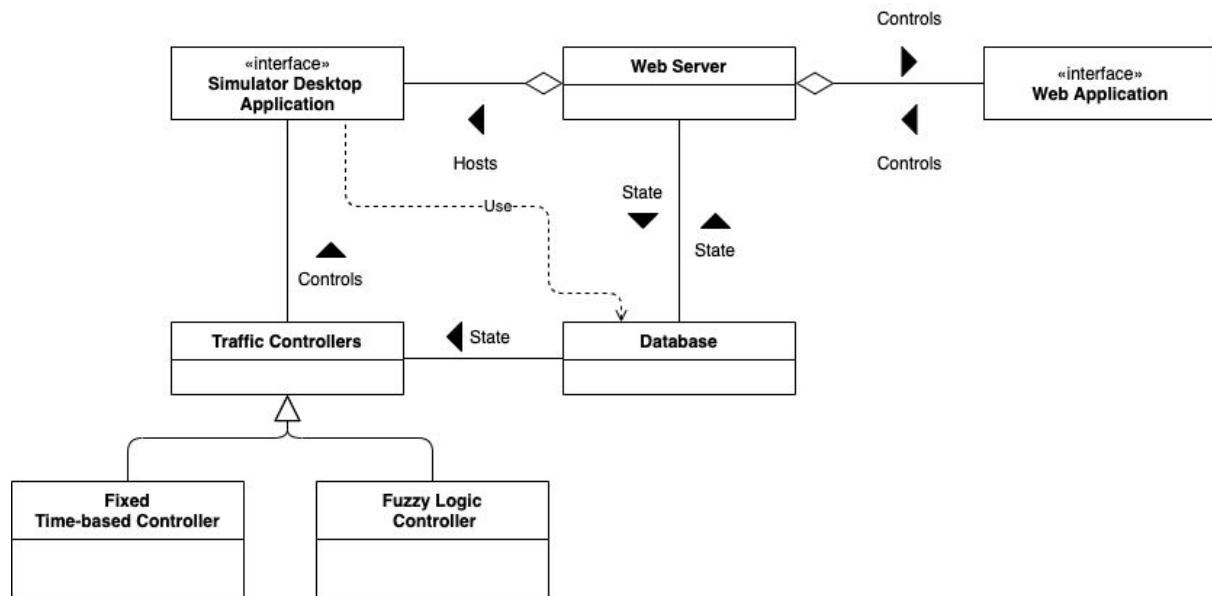


Figure 6: Domain Model of LightBot System

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.
- 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.

LightBot Web Server

- The patterns chosen for the server is a 3-Layer 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.
- 3-Layer 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)
 - Service layer (the server provides business logic as services via utilities, services, and handlers to the controller layer)
 - 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)
- 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

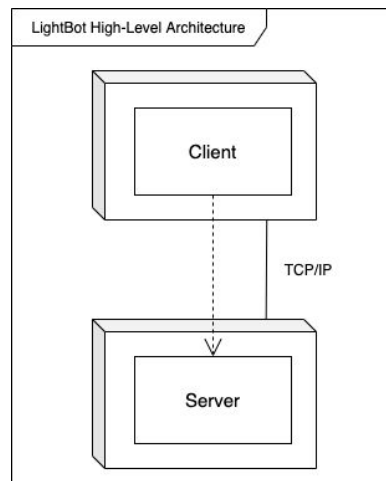


Figure 8: High-Level Architectural Design Diagram

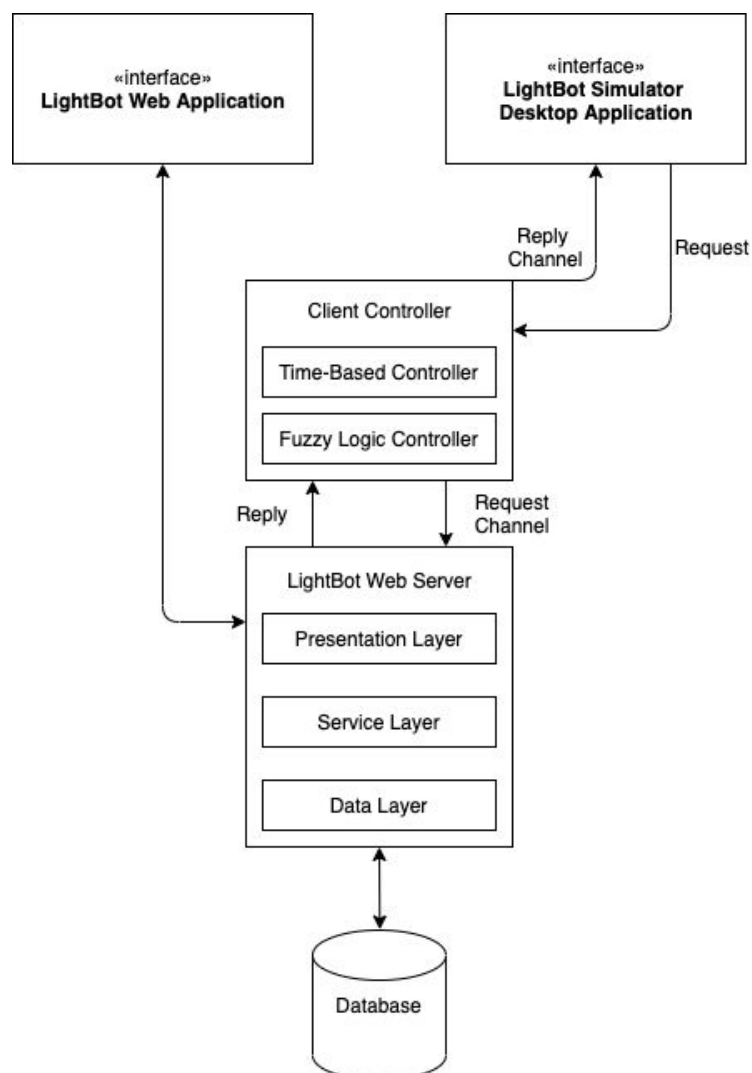


Figure 9: LightBot Architectural Design Diagram

6.3 Deployment Model

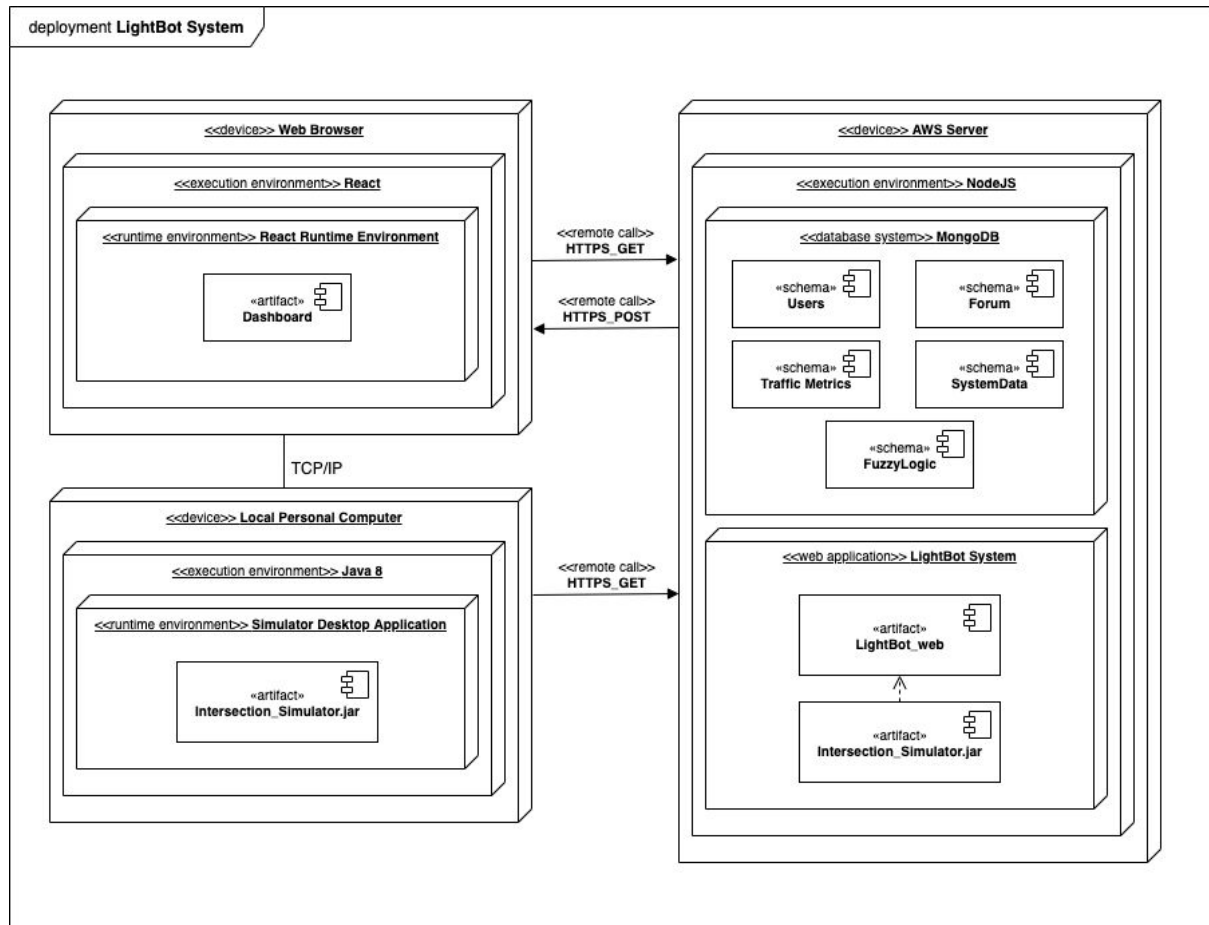


Figure 10: Deployment Diagram

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-Controller 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%.
 - $\text{Effectiveness} = (\text{Nr. of tasks completed successfully} / \text{Total nr. of tasks undertaken}) * 100\%$
 - 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.
 - $\text{Apdex} = (\text{SatisfiedCount} + (\text{ToleratingCount} / 2)) / \text{TotalSamples}$
 - The aim of the systems Apdex score is to be at least 0.9

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:
 - $\text{Availability} = \text{Uptime} / (\text{Uptime} + \text{Downtime})$
 - 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:
 - $\text{Availability} = 200 / (200+10) = 0.952 = 95.2\%$
- 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

- **React:** React, which is also known as ReactJS, is an open-source JavaScript library used to build the web application's interface. I 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. I 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 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 us to shorten our code.

8.2 LightBot Web Server

- **NodeJS:** is an open-source, cross-platform, JavaScript runtime environment (Framework) that executes JavaScript code outside a web browser. This gives us 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.

- **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 providing a form of relational links that is often not included in the NoSQL type databasing solutions like MongoDB.
- **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:** 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 lenable ourselves 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.

9. References

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

Justin Mifsud n.d., *Usability Metrics - A Guide To Quantify The Usability Of Any System*, viewed 10 August 2020,

<[19](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.></p></div><div data-bbox=)