LionQ

Jeanna Maye Benitez

CST-452 Capstone Project Final Architecture & Design

Grand Canyon University

Professor Mark Reha

v5.0

April 24, 2022

**ABSTRACT**

This project is a social media website where users can create posts, comment on other user’s posts and give feedback, private message other users, and personalize their profile to their liking. The user’s posts will be stored in a database that will be saved, so that their posts will show up when another user clicks on their profile, like a timeline. The users can search posts by the title of the post and search users as well.

The website allows users to connect with other people around the world. It allows for them to create a community and meet new people with common interests. Users can message each other and the messages will save to a database, so that users can search for previous messages. The website will then be uploaded to the cloud, which will then allow users to access the site through a global link.

| History and Signoff Sheet |
| --- |

**Change Record**

| **Date** | **Author** | **Revision Notes** |
| --- | --- | --- |
| 10/26 | Jeanna Maye Benitez | Notes |
| 11/17 | Jeanna Maye Benitez | UML, Logical, & Physical Diagrams, POCs, Technologies |
| 11/21 | Jeanna Maye Benitez | Written paragraphs |
| 04/05/2022 | Jeanna Benitez | Updated Abstract, Data Dictionary Excel, Hardware & Technologies, & DDL Script |

| **Overall Instructor Feedback/Comments** |
| --- |

| **Overall Instructor Feedback/Comments** |
| --- |

**Integrated Instructor Feedback into Project Documentation**

☐ Yes ☐ No

**TABLE OF CONTENTS**

**Design Overview 4**

**Detailed High-Level Solution Design 5**

**Detailed Technical Design 6**

**Appendix A – Technical Issue and Risk Log 7**

**Appendix B – References 8**

**Appendix C – External Resources 9**

**Design Introduction**

We are going to be using Spring Boot Java, which is a tool that helps make developing web applications with the Spring Framework faster and easier. We will also be using external libraries like websocket, sockjs, stomp, and more, to help create this application to what we envision it to be. We will be using the MVC framework to implement basic features such as inversion of control and dependency injections. The application will follow a basic flow and the user will be able to navigate between all the features and pages easily. The navbar that will be developed will help the user with that. There will be security that is provided with Spring, and it will have authentication, which we will be using to authenticate user log-ins and make sure the user will have to log in or create an account before being able to access the entire web application. View the flowcharts on pages 12-17 to see the process flow and how the web application will be mapped out.

| Deliverable Acceptance Log | | | | | |
| --- | --- | --- | --- | --- | --- |
| ID | Deliverable Description | Comments | Evaluator (internal or external as applicable) | Status | Date of Decision |
| 1 | Data Dictionary | The database layout for the tables | internal | Complete | 11/01 |

View the Data Dictionary file: ‘LionQ Data Dictionary.xlsx’ to see the data dictionary of the application.

**Detailed High-Level Solution Design**

The following design architectures and tables you will see are blueprints for how the web application will be developed. Also, there are libraries listed that will be used for the project.

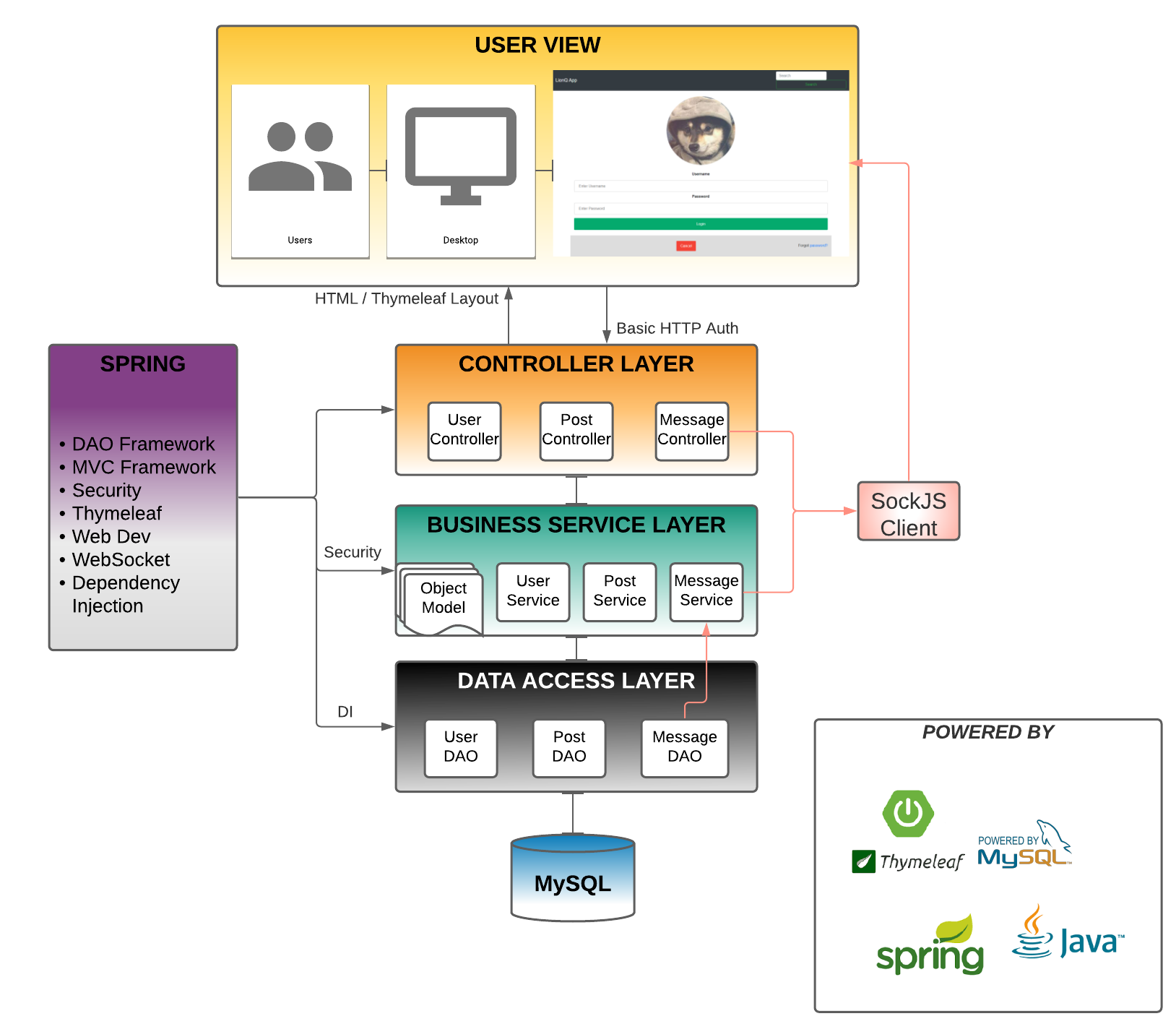
| Proof of Concepts | |  |
| --- | --- | --- |
| **Description** | **Rationale** | **Results** |
| 1 - Using WebSocket to create interactive web application between users |  | It works with the STOMP Client and SockJS |

The Proof of Concepts, or POCs, are concepts that need or were proved to have worked in the program.

| Hardware and Software Technologies |
| --- |
| 1 - Java (v11) |
| 2 - Spring Boot Starter Thymeleaf |
| 3 - Spring Boot Starter Web |
| 4 - Spring Boot DevTools |
| 5 - Spring Boot Starter Validation |
| 6 - javax Validation API |
| 7 - Thymeleaf Layout Dialect |
| 8 - Spring Boot Starter JDBC |
| 9 - MySQL Connector Java |
| 10 - Spring Boot Starter JPA |
| 11 - Spring Boot Starter Webflux |
| 12 - Spring Boot Starter Security |
| 13 - Spring Boot Starter Websocket |
| 14 - javax jws API (v1.1) |
| 15 - Google GSON (v2.8.5) |
| 16 - Project Lombok |
| 17 - Spring Boot Starter Reactor Netty |
| 18 - Micrometer Registry New Relic (v0.5.0) |

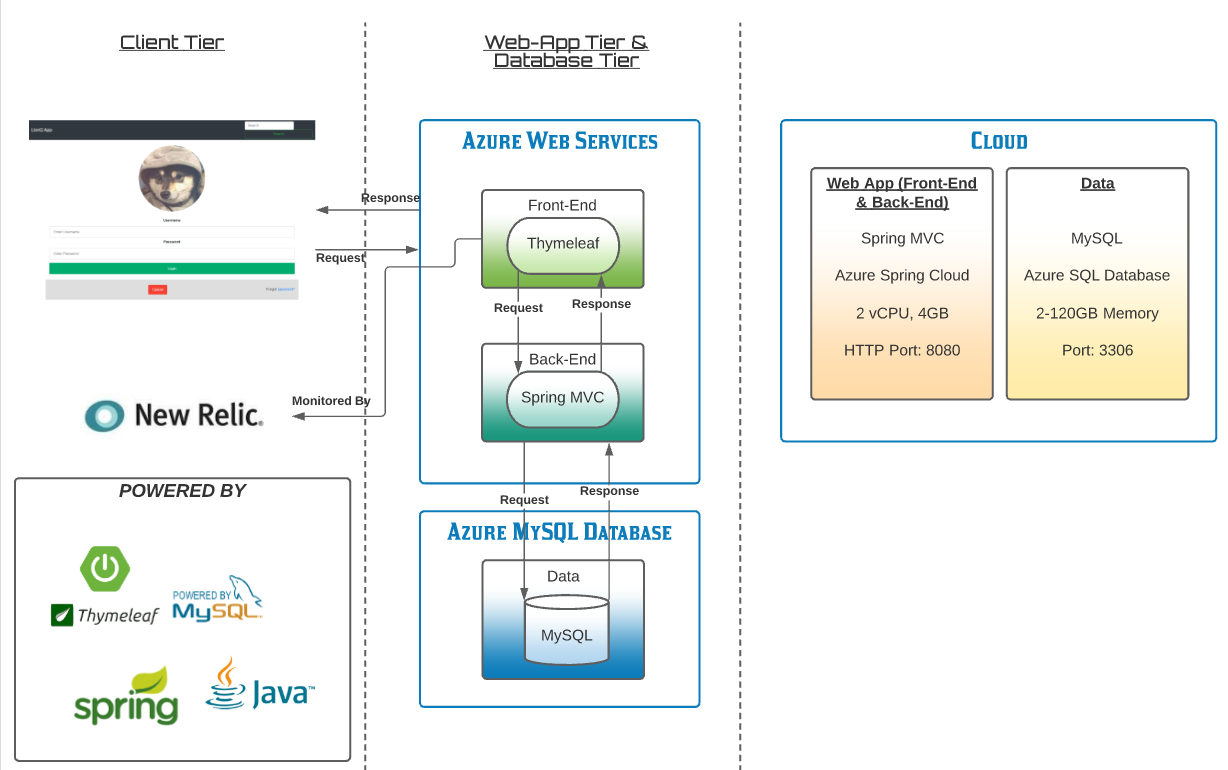
The hardware and software technologies that are needed for the web application project.

**Logical Solution Design:**



Java Spring Boot provides many libraries such as web development, thymeleaf, and more. We will manually add the websocket, sockjs, and stomp client dependencies into the application. It also provides an MVC (Model-View-Controller) framework that helps tremendously when building this application. We are able to divide the work of the application and the developed code will be easier to read for other developers that will be doing the project. There will also be a database that will be provided by MySQL to store data that is being registered or requested from the user.

**Physical Solution Design:**



The Azure Cloud Services will be able to store the data and have the application running through the Azure Spring Cloud Service. The Azure Spring Cloud service provides an easier way for the Spring Boot application to be installed. The user will be able to request data, which happens at the front-end of the application, using Thymeleaf for the design, and it then goes to the back-end of the application where the Spring MVC framework lies. From the Spring MVC framework, it will request data from the MySQL database, and it will respond to the back-end and then to the front-end, where the data will be outputted with the results or an error page. New Relic is what will monitor the front-end of the application, and the application itself will all be going into the Azure Cloud.

**Detailed Technical Design**

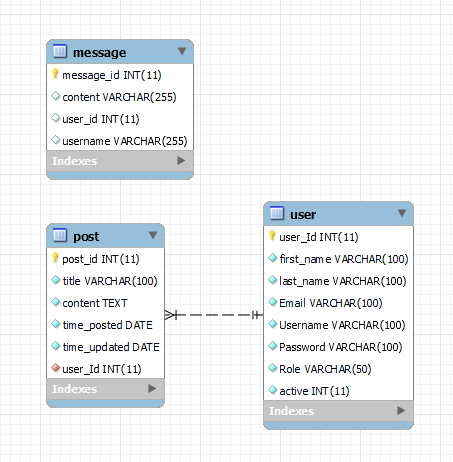
**General Technical Approach:**

We want to create a website that can be easily navigated by a user and has an appealing design to the users. We would need a navbar to navigate between the pages of the application and for there to be user authentication for security of the application. From there, the user will be able to access features such as creating a post, editing your account, searching for posts or users, editing or deleting your posts, and messaging other users. In the search page, the user will be able to see the title of the post they searched and which user created the post. The username will have a link to their profile page where a user will be able to see their timeline of posts, similar to the account page. If the user searches for other users they will be able to see their profile picture, their username, and a ‘direct message’ link. The message link will prompt the user to a messaging template where users can converse with each other privately. A user will also be able to see all their messages if they click on their account page and click on ‘messages’. The user will also be able to edit their account by clicking on ‘edit profile’ from the account page or by clicking on ‘my account’ in the navbar. The users will be able to easily navigate back and forth between the pages in the application.

**Key Technical Design Decisions:**

We will be using Thymeleaf to design the project, as well as bootstrap for the color scheme. For the messaging layout we will use outside clients such as SockJS, WebSocket, and STOMP. Each page will have default footers and headers so that all the pages will match. Also, the application will have a default template that will include the navbar for the users to easily access the other features of the application. Bootstrap makes the creative development of the project go smoother and Thymeleaf goes hand-in-hand with Spring Boot and allows the developers to utilize its template engine for web development.

**Database ER Diagram:**



The ER diagram shown is how the tables of data will be connected. A user will have data like the id (which will be auto incremented, along with post\_id), firstname, lastname, etc. When a user creates a post, the post\_userid will be filled out with the id of the user who created the post. The time\_posted and time\_updated will automatically be filled with data of the time when the user updates or creates the post. When a message is made, the user\_id and username of the user who made the message is stored in the database along with the contents of the message.

**Database DDL Scripts:**

--

-- Database: `lionqdb`

--

CREATE DATABASE IF NOT EXISTS `lionqdb` DEFAULT CHARACTER SET utf8 COLLATE utf8\_general\_ci;

USE `lionqdb`;

-- --------------------------------------------------------

--

-- Table structure for table `message`

--

CREATE TABLE `message` (

`message\_id` int(11) NOT NULL,

`content` varchar(255) DEFAULT NULL,

`user\_id` int(11) DEFAULT NULL,

`username` varchar(255) DEFAULT NULL

) ENGINE=MyISAM DEFAULT CHARSET=utf8;

-- --------------------------------------------------------

--

-- Table structure for table `post`

--

CREATE TABLE `post` (

`post\_id` int(11) NOT NULL,

`title` varchar(100) NOT NULL,

`content` text NOT NULL,

`time\_posted` date NOT NULL,

`time\_updated` date NOT NULL,

`user\_Id` int(11) NOT NULL

) ENGINE=InnoDB DEFAULT CHARSET=utf8;

-- --------------------------------------------------------

--

-- Table structure for table `user`

--

CREATE TABLE `user` (

`user\_Id` int(11) NOT NULL,

`first\_name` varchar(100) NOT NULL,

`last\_name` varchar(100) NOT NULL,

`Email` varchar(100) NOT NULL,

`Username` varchar(100) NOT NULL,

`Password` varchar(100) NOT NULL,

`Role` varchar(50) NOT NULL DEFAULT 'USER',

`active` int(11) NOT NULL

) ENGINE=InnoDB DEFAULT CHARSET=utf8;

--

-- Indexes for dumped tables

--

--

-- Indexes for table `message`

--

ALTER TABLE `message`

ADD PRIMARY KEY (`message\_id`),

ADD KEY `FKb3y6etti1cfougkdr0qiiemgv` (`user\_id`);

--

-- Indexes for table `post`

--

ALTER TABLE `post`

ADD PRIMARY KEY (`post\_id`) USING BTREE,

ADD KEY `user\_Id` (`user\_Id`) USING BTREE;

--

-- Indexes for table `user`

--

ALTER TABLE `user`

ADD PRIMARY KEY (`user\_Id`);

--

-- AUTO\_INCREMENT for dumped tables

--

--

-- AUTO\_INCREMENT for table `message`

--

ALTER TABLE `message`

MODIFY `message\_id` int(11) NOT NULL AUTO\_INCREMENT;

--

-- AUTO\_INCREMENT for table `post`

--

ALTER TABLE `post`

MODIFY `post\_id` int(11) NOT NULL AUTO\_INCREMENT;

--

-- AUTO\_INCREMENT for table `user`

--

ALTER TABLE `user`

MODIFY `user\_Id` int(11) NOT NULL AUTO\_INCREMENT;

--

-- Constraints for dumped tables

--

--

-- Constraints for table `post`

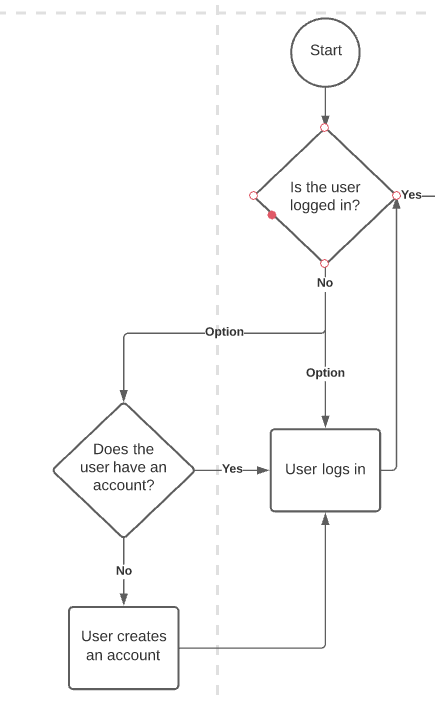
--

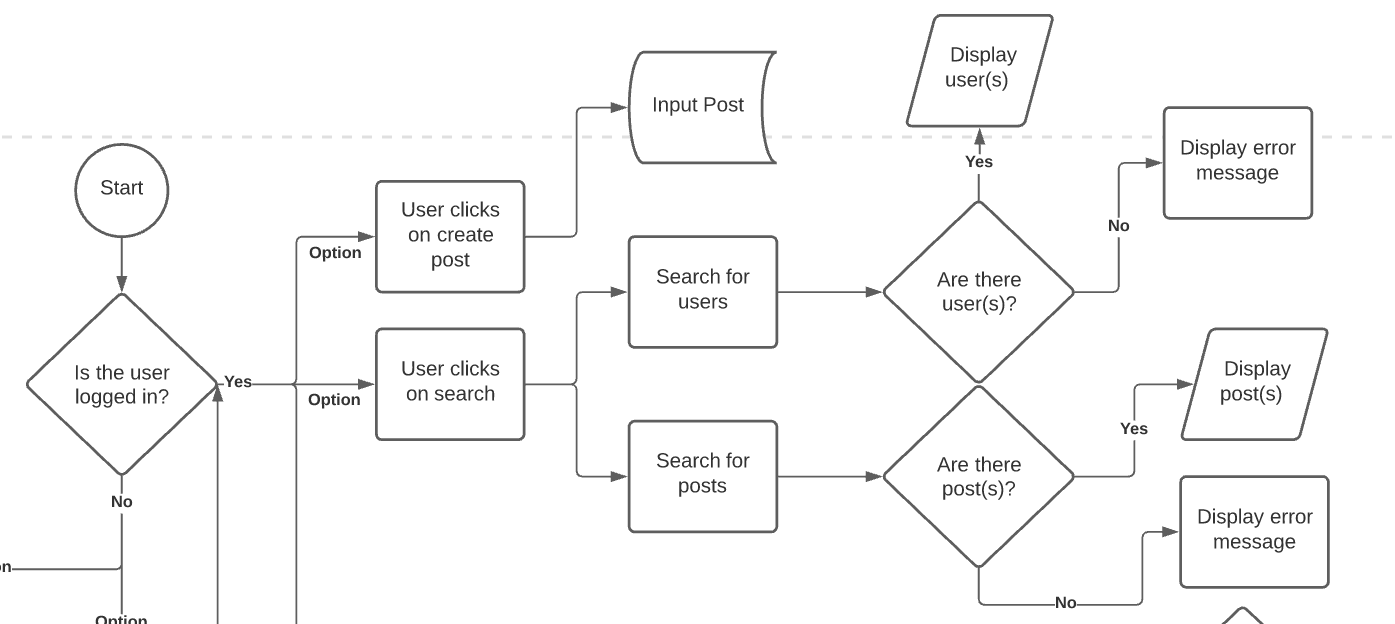
ALTER TABLE `post`

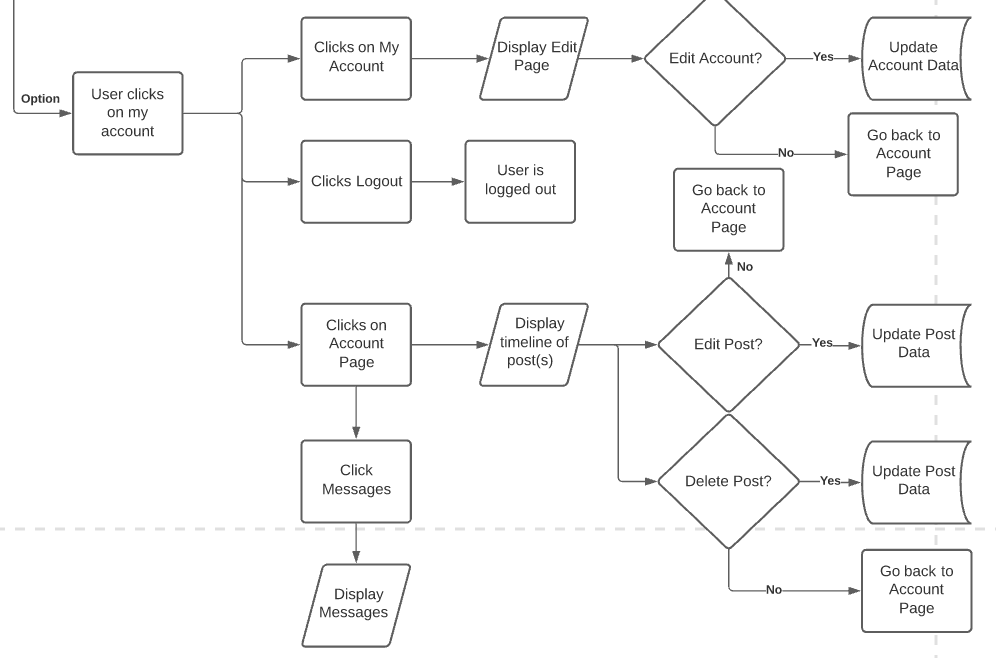
ADD CONSTRAINT `user\_Id` FOREIGN KEY (`user\_Id`) REFERENCES `user` (`user\_Id`) ON DELETE NO ACTION ON UPDATE NO ACTION;

COMMIT;

**Flow Charts/Process Flows:**

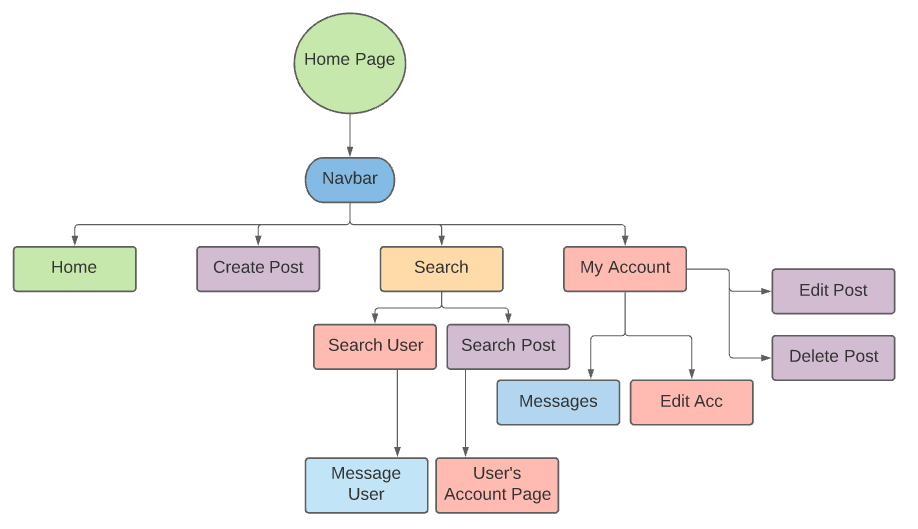






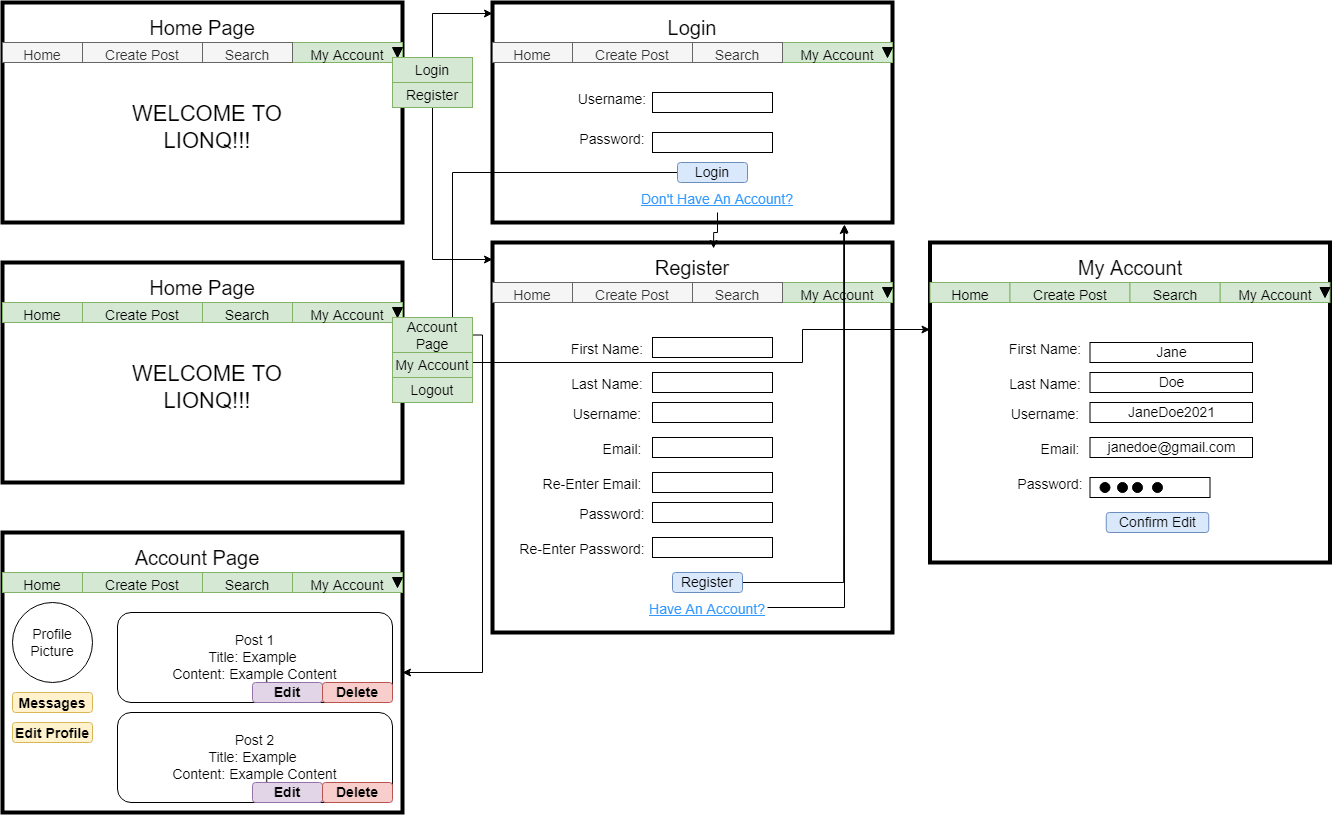
This is how the program will be processed. If the user is logged in, they will be able to access the features of the web application, and if they are not, they will be prompted to login with an existing account or create an account. From there they will be able to create, edit, and delete their posts, and search for other posts as well. They will also be able to view their posts via their account page and view their messages.

**Sitemap Diagram:**

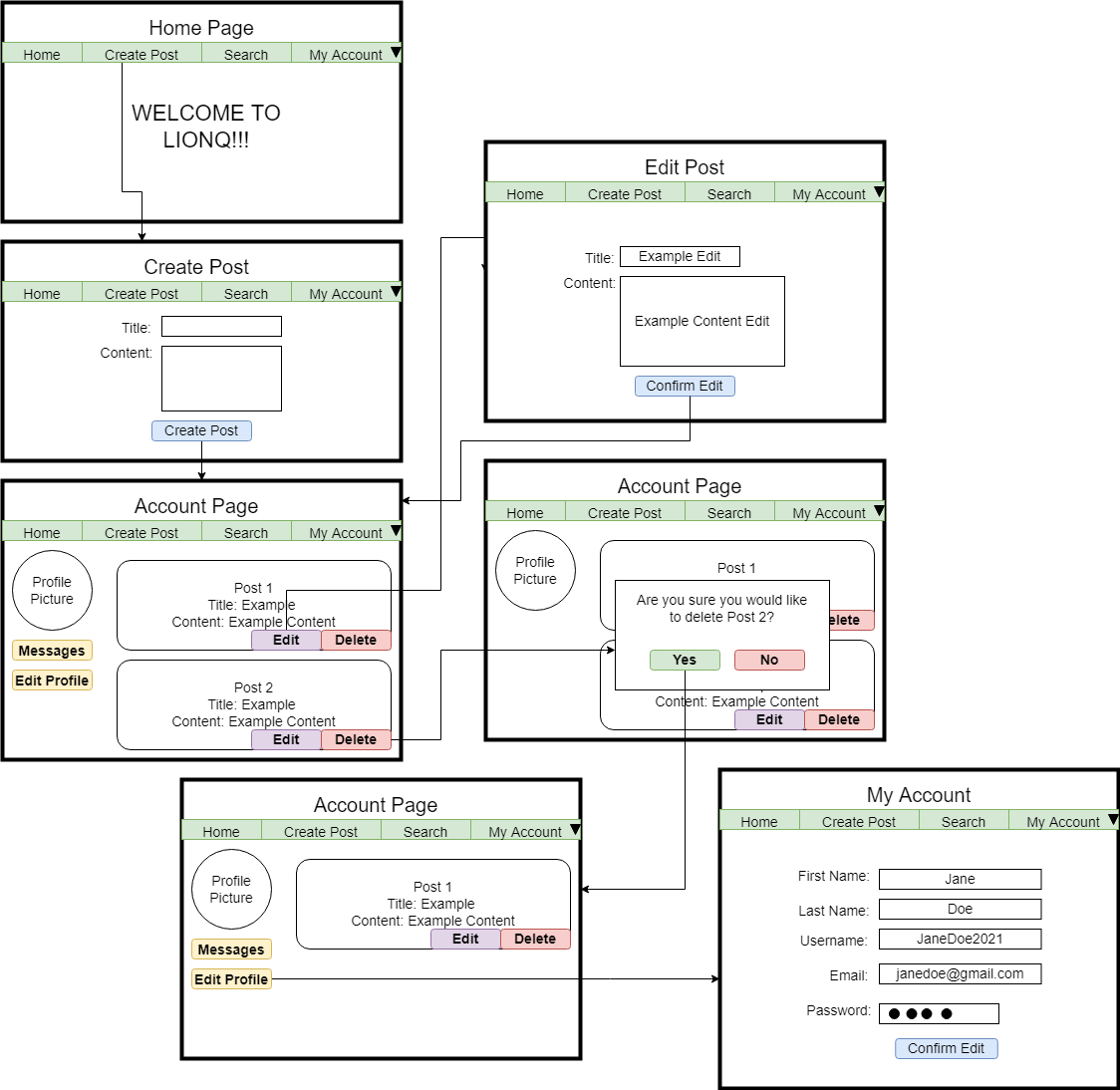


This is how the application will be mapped out and how each page connects to each other. There will be a navbar in the web application so it is easier for the user to navigate between pages.

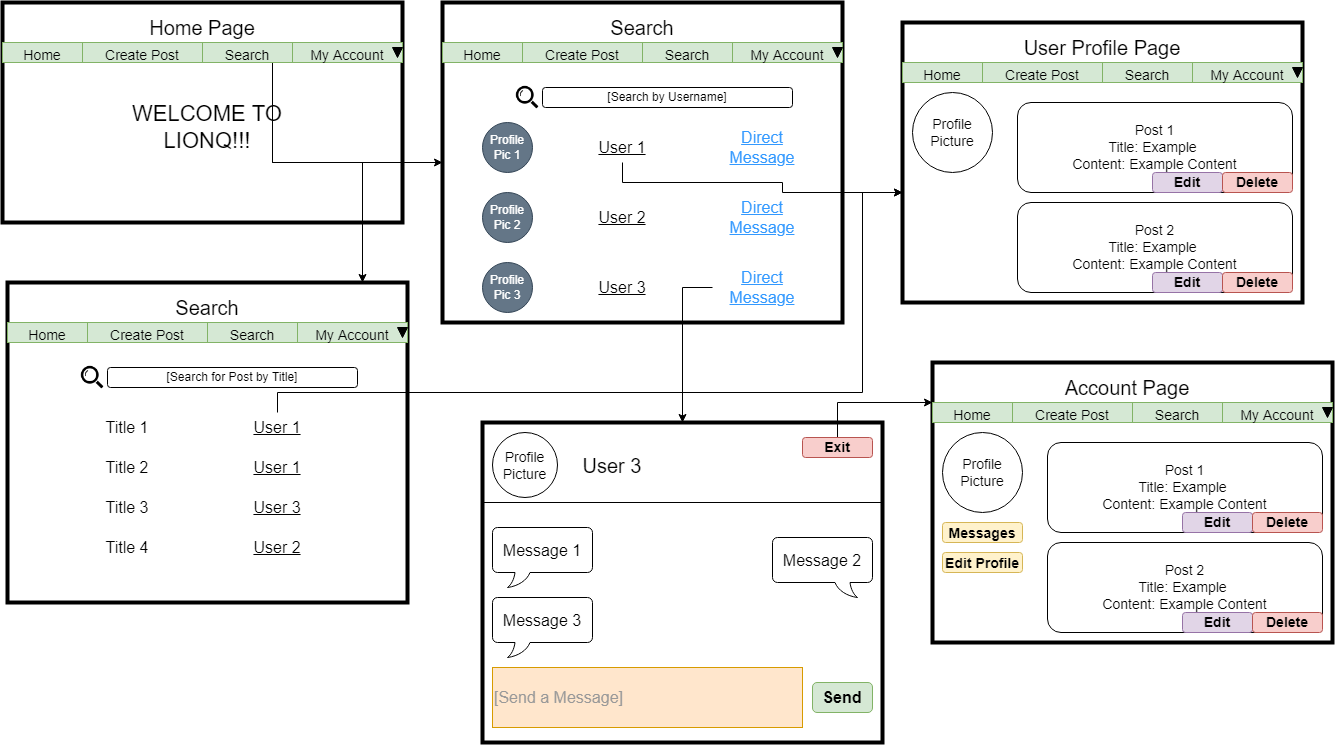
**User Interface Diagrams:**



From the home page of the application, the user will not be able to access features like creating a post until they register or login with an existing account. From there, they will be able to view their own page, where they can edit their profile, view their messages, and edit or delete their post(s).

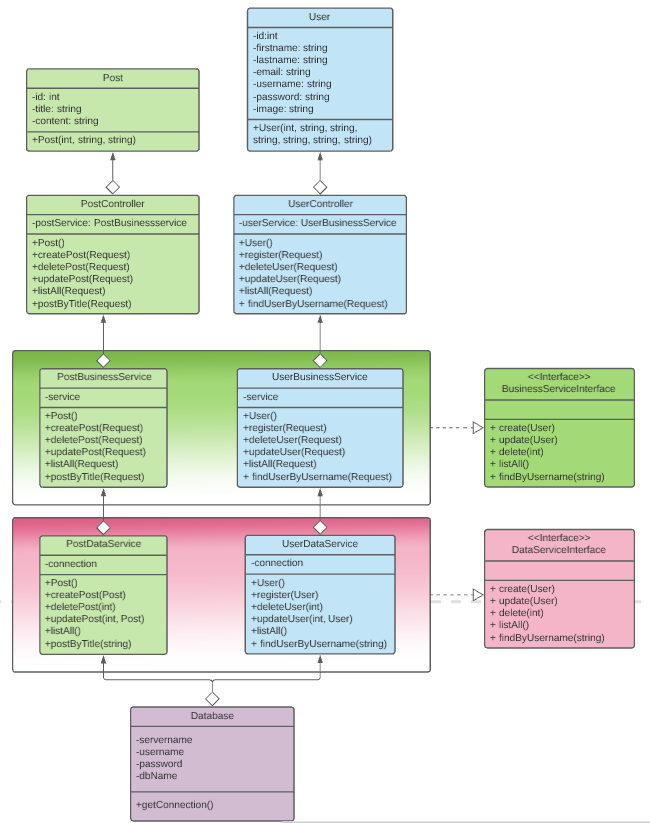


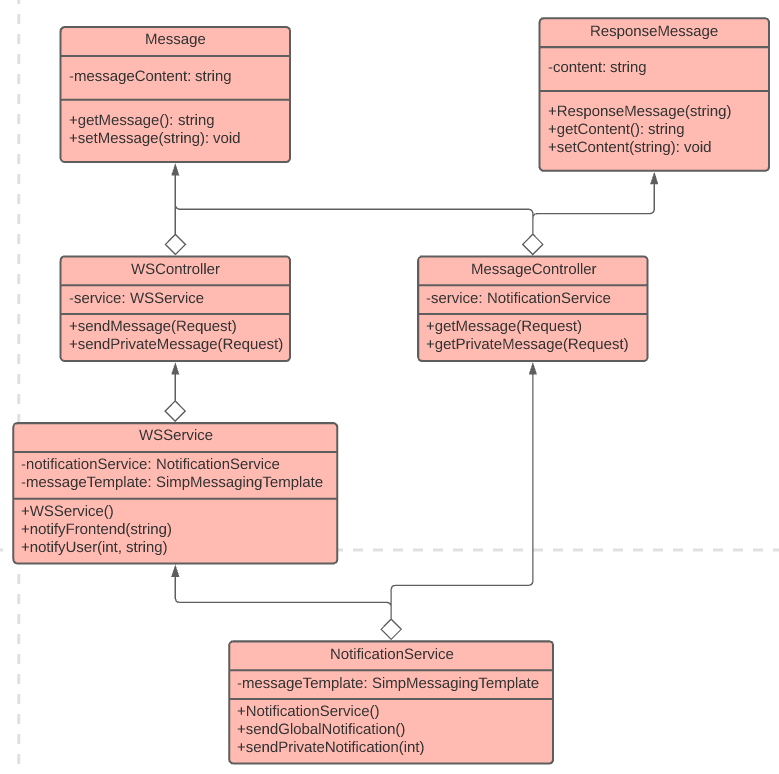
From the home page, the user is able to create a post. They will also be able to edit their post by going to their account page and that will lead them to another page for them to edit. The same thing happens when they choose to edit their profile, they will be redirected to another page for them to edit. When they want to delete a post, a pop-up will happen and prompt the user to confirm whether or not they would like to delete their post.



The user is also able to search for users by their username or search for posts by their title. When a user searches for other users, a profile picture, the username, and a link to directly message the user will pop up, along with similar usernames. When a user searches for posts, the title of the post and the user who created the post will show up. The user is also able to click on the username to show a user’s profile page. When the user clicks “Direct Message”, they will be redirected to a messaging page where users will be able to message each other privately.

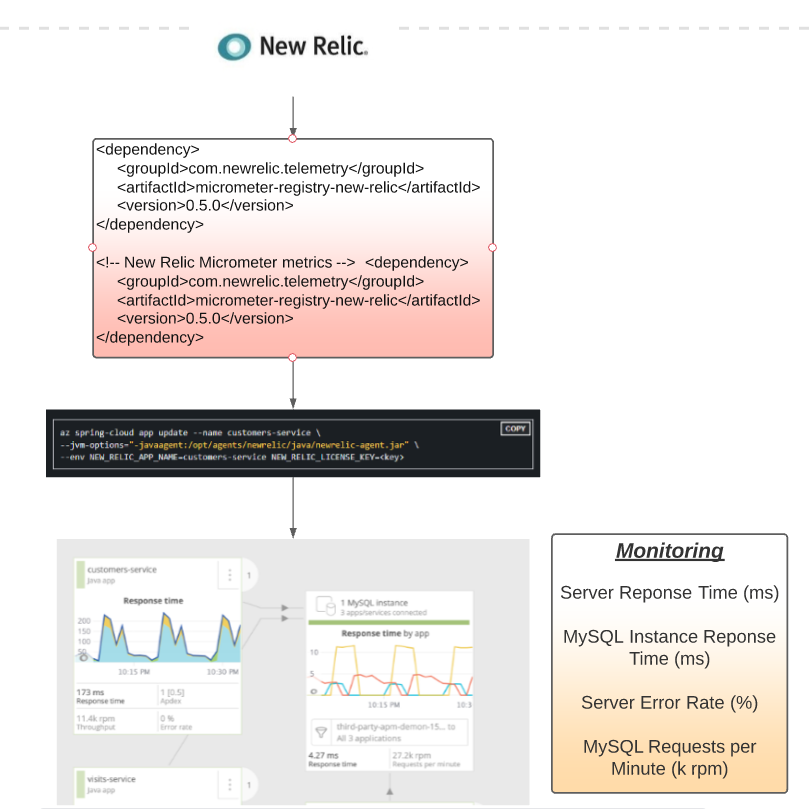
**UML Diagrams:**





These are the classes that will be made for the application and how they will all connect to each other. We have the messaging services with the notification that will pop up, along with the controllers that will see what the user requested and will output the appropriate response, whether it be the data, or page the user wanted, or an error message. We have the database that will be directly connected to the data service classes and those classes will have interface classes that are a template of sorts for the methods that will be used in the classes. The data service is what will pull and push data in and out of the database and it connects to the business service classes, which will output or input data for the user to eventually see and the business service classes will also have interface classes for their methods. The controller classes will control what the user will view on the page and what data is going to be inputted or outputted. For example, if the user wants to view the ‘create post’ page, the program will first check if the user is logged in, if not the user will then be prompted to register or login with an existing account first. Finally, we have our model classes that are there to help guide data through the classes.

**NFR’s (Security Design, etc.):**

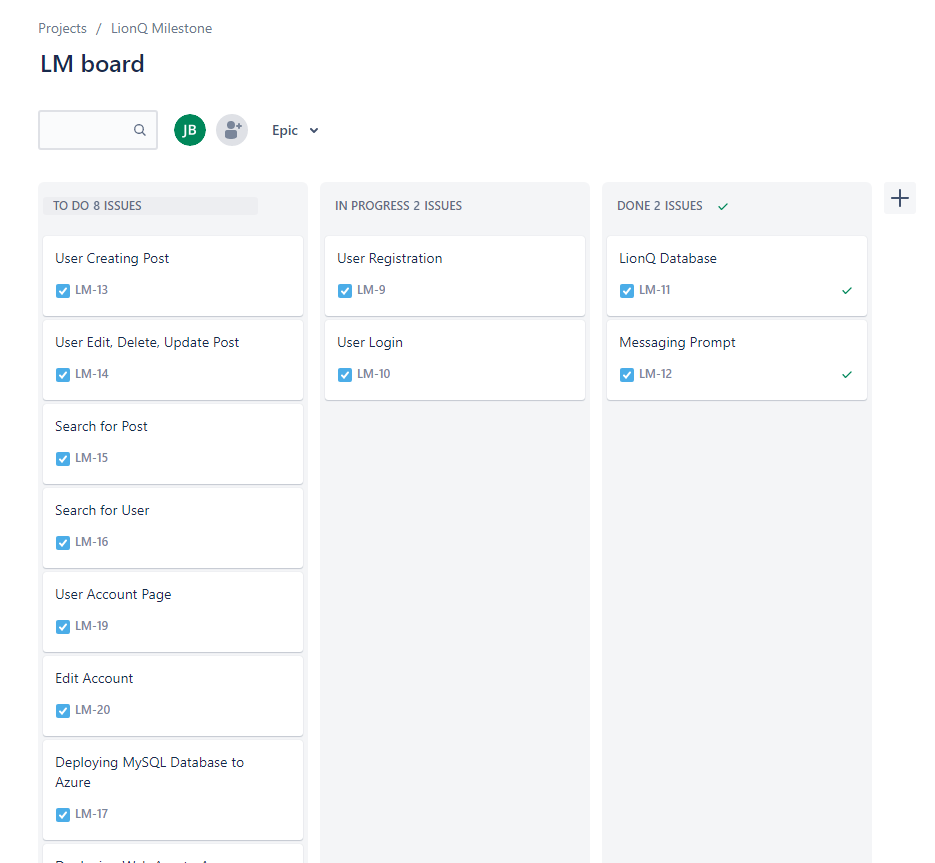


The NFR we will be keeping track of is the deployment of the application to the Azure Cloud service and the response time of the server and MySQL instance. We will be monitoring the server error rate and MySQL requests per minute (rpm), with the help of New Relic.

**Operational Support Design:**

We will be using New Relic to monitor the application. This will help track the performance of the application.

**Other Documentation:**



**Appendix A – Technical Issue and Risk Log**

| Issues and Risk Log | | | | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Issue or Risk** | **Description** | **Project Impact** | **Action Plan/Resolution** | **Owner** | **Importance** | **Date Entered** | **Date to Review** | **Date Resolved** |
| I/R | What is the issue or risk? | How will this impact scope, schedule, and cost? | How do you intend to deal with this issue? | Who manages this issue? |  |  |  |  |
| Issue | Messaging Template w/ MySQL | It will impact the project a little bit, but not a lot | Researching more tutorials on how to connect the WebSocket Client to MySQL data | Jeanna Maye Bentiez | Very Important | 11/18 | 11/20 | 11/21 |
| Issue | Azure Spring Cloud Cost | We will need to pay for the cost of the application when it runs on the cloud | Paying for the cost and making sure it is within budget | Jeanna Maye Benitez | Very Important | 11/19 | 11/19 | 11/21 |

**Appendix B – References**

*Soares, Victor. (n.d.). Monitoring spring boot applications using micrometer metrics in New Relic. New Relic. https://newrelic.com/blog/how-to-relic/how-to-monitor-spring-boot-applications-using-micrometer-metrics.*

*Using websocket to build an interactive web application. Spring. (n.d.). https://spring.io/guides/gs/messaging-stomp-websocket/.*

*Websocket push messages with spring boot - stomp - youtube.com. (n.d.). https://www.youtube.com/watch?v=LdQY-OUM2mk.*

**Appendix C – External Resources**

| **GIT URL:** | [*https://github.com/liliumbosniacum/websocket*](https://github.com/liliumbosniacum/websocket) |
| --- | --- |
| **Hosting URL:** | *N/A* |