**MicroChat**

Brady Berner

CST-451 Capstone Project Final Architecture & Design

Grand Canyon University

Instructor: Professor Mark Reha

Revision: 1.0

Date: 11/29/20

**ABSTRACT**

Communication is vital in today’s world. Companies rely on it to stay operating at break-neck paces. Consumers have grown so used to near-instantaneous communication that it is a must-have for many. However, many of the market applications today have security flaws, reliability issues, or cost consumers to use. This project will provide uses with a messaging application built to be fast, reliable, and secure MicroChat will allow users to easily communicate with each other, potentially from virtually any platform they desire. The application will remain simple to appeal to as many consumers as possible, regardless of demographics. To operate the application, a consumer can go to the website, log in, or create an account, select whom they would like to contact, and begin sending and receiving messages. The application could also serve as a form of consumable entertainment with a feature that would enable users to select whether they would like to start a conversation with another user chosen at random. The application could be expanded upon to allow for encrypted messaging to ensure user security and privacy and support multimedia messages such as pictures, videos, files, et cetera.

|  |
| --- |
| History and Signoff Sheet |

**Change Record**

|  |  |  |
| --- | --- | --- |
| **Date** | **Author** | **Revision Notes** |
| 11/29/10 | Brady Berner | Initial draft for review/discussion |
|  |  |  |
|  |  |  |

|  |
| --- |
| **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 6

Detailed Technical Design 10

Appendix A – Technical Issue and Risk Log 17

Appendix B – References 19

Appendix C – External Resources 20

Design Introduction

The primary purpose of this project is to realize an application that utilizes a microservice based architecture from design to realization. Compared to a traditional monolithic software architecture microservices take the traditional monolith and divide it horizontally across. The same applies to upgrading/expansion, while traditionally more code would be added on to the existing code base creating an ever-expanding application adding to microservices is instead done by adding on small amounts to existing microservices or creating a new service. With each service being separate however there are some major variations to consider such as the fact that each service needs its own database. Since each service needs its own database and the services are only loosely connected to each other the benefit of relational databases are mostly lost, meaning that non-relational databases are usually the go to solution as their drawbacks are already present so the design is left only really taking their benefits. With all this in mind below is a figure depicting a simple application built using a microservice based architecture.

1. 

(Richardson, 2020)

Looking at this figure shows how REST APIs are typically used to tie the architecture together making the application even more flexible. Some of the biggest challenges with a microservice based architecture however has to do with its inherent higher level of complexity. This complexity creates issues like maintaining data consistency between each services database, implementing some form of service discovery to allow services to communicate with one another when necessary, and many other unique issues.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Deliverable Acceptance Log | | | | | |
| ID | Deliverable Description | Comments | Evaluator (internal or external as applicable) | Status | Date of Decision |
| 1 | DDL Scripts | Since I am using MongoDB instead of SQL for my database these are instead JSON exports of the database structure. As these are still text heavy much like a DDL script these have been included in a folder accompanying the document instead of contained within | Internal | Complete | 11/27/20 |
| 2 | API Documentation | Included alongside the document is a generated html documentation page for the API from Swagger. | Internal | Complete | 11/27/20 |
| 3 | Database Dictionary | Included alongside the document is an excel spreadsheet containing the Database Dictionary for the project | Internal | Complete | 11/27/20 |

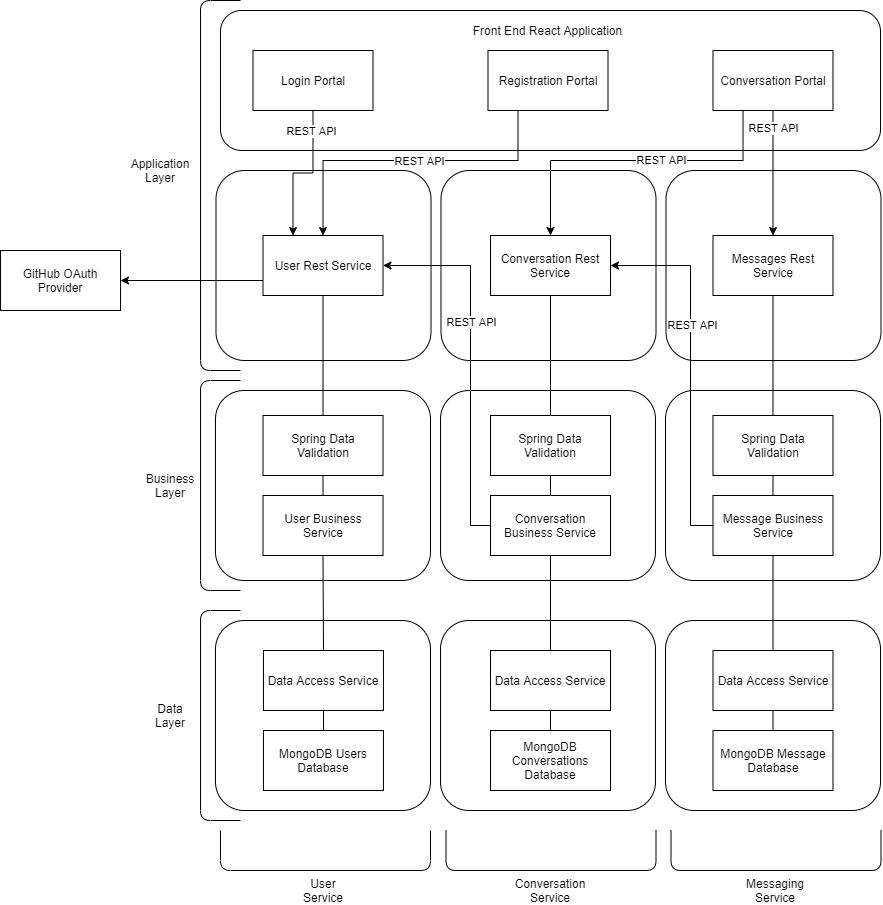
Detailed High-Level Solution Design

The proposed solution design was created with making the application as modular as possible in mind. As the logical design shows between typical MVC service layering and the vertical slicing that micro-service architecture entails, the application is divided into ten different parts. Each small enough that should be necessary to change something out, there should be little refactoring necessary, and problems are easier to isolate. This modularity is better for the long-term scalability and sustainability of the application. The physical design also keeps scalability and sustainability in mind. Using containers within AWS's EKS and Databases within MongoDB Atlas, the application can be scaled as necessary if needed. For example, if the application grows more prominent and more messages are being sent, creating too much strain on the message service or database, another container could be spun up to mitigate the increased load. While the application is not currently designed to support functionality, its modularity makes refactoring to support such cases much more feasible.

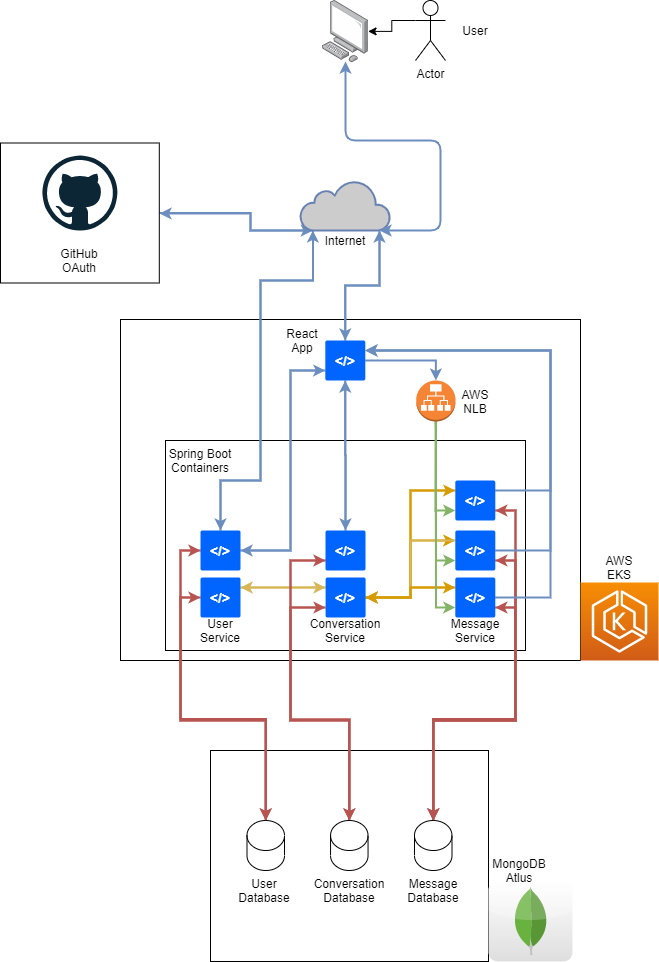
|  |  |  |
| --- | --- | --- |
| Proof of Concepts | | |
| **Description** | **Rationale** | **Results** |
| 1. React proof of concept. This proof of concept involved using the materials provided by Prof. Reha to complete all the activities the university will eventually offer relating to React. | This is my first time working with React so this proof of concept was to learn how to use React. | The proof of concept was completed without issue and was sufficient to learn everything necessary to use React in the final project. |
| 1. Spring Boot proof of concept. This proof of concept involved using the materials provided by Prof. Reha to complete all the activities the university will eventually offer relating to Spring Boot. | While I have worked with Spring MVC before I have not done anything with Spring Boot. So, this proof of concept was intended to reach a level of comfort with the technology that it can be used in the final project. | The proof of concept was completed without issue and was sufficient to learn everything necessary to use Spring Boot in the final project. |
| 1. MongoDB proof of concept. This proof of concept used MongoDB documentation and some resources from the assignments from the Spring Boot proof of concept to have the Spring Boot proof of concept use a MongoDB database. | This is my first-time using MongoDB or any type of non-relational database. As such this proof of concept was meant to learn how to use and design non-relational databases. | The proof of concept was completed without issue and was sufficient to learn everything necessary to use MongoDB in the final project. |
| 1. OAuth proof of concept. This proof of concept integrated OAuth into the Spring Boot proof of concept. | I have not used OAuth in any of my previous projects, so this proof of concept was intended to learn everything necessary to use it in the final project. | The proof of concept was completed without issue. However, it presented that further research must be done to not only use OAuth for logging in and securing APIs but also for registering users. |
| 1. Vertical proof of concept. This proof of concept put together all of the technologies learned in other proof of concepts. | The purpose of this proof of concept was to test all the new technologies learned for the final project interacting together. It also allowed for further insight into the final projects overall design. | The proof of concept was completed without issue and revealed no immediate flaws in design. |

|  |
| --- |
| Hardware and Software Technologies |
| 1. React – Version 16.13.1 |
| 1. Spring Boot – Version 2.4.0 M3 |
| 1. MongoDB – Version 4.4 |
| 1. Kubernetes – Version 1.19.2 |
| 1. OAuth 2.0 |
| 1. Material-UI |
| 1. IntelliJ |
| 1. WebStorm |
| 1. Mongo Compass |
| 1. Java 8 |
| 1. GitHub |
| 1. Log4j2 |

**Logical Solution Design:**



**Physical Solution Design:**



Detailed Technical Design

**General Technical Approach:**

Keeping in mind that a large part of this project’s goal was to create an application utilizing a micro-serviced-based architecture, many of the project’s design decisions were made concerning this. For example, the decision to use MongoDB for the application’s databases was based mainly in part that the application would be built using micro-services, so the significant benefits of a traditional relational database would be useless in this case. Many of the other design decisions that were not directly influenced by the application’s micro-service nature were made due to standardization throughout the industry. An excellent example of this would be Spring Boot, Spring Boot is rapidly growing throughout the programming industry, and as such, it is likely to receive support for a long time to come. Choosing such technologies (another example being React) leads to a likely overall increase in the application’s longevity and security. Between this and the overall sustainability and updatability inherent in micro-service-based architectures, most design decisions were made to create a lasting and robust system.

**Key Technical Design Decisions:**

The three main design decisions throughout this project so far would be those in which technology was decided upon. These decisions resulted in the implementation of React, Spring Boot, MongoDB, and OAuth in the application. All of which technologies first went through a proof of concept phase before being decided upon for the final application. These technologies posed a challenge as they were all new to me; however, they were very robust and widespread throughout the industry. Keeping in mind that these technologies were new, it was initially considered that should any of their proofs of concept fail, then technologies that I had worked with before could fill their place, such as Spring MVC for Spring Boot.

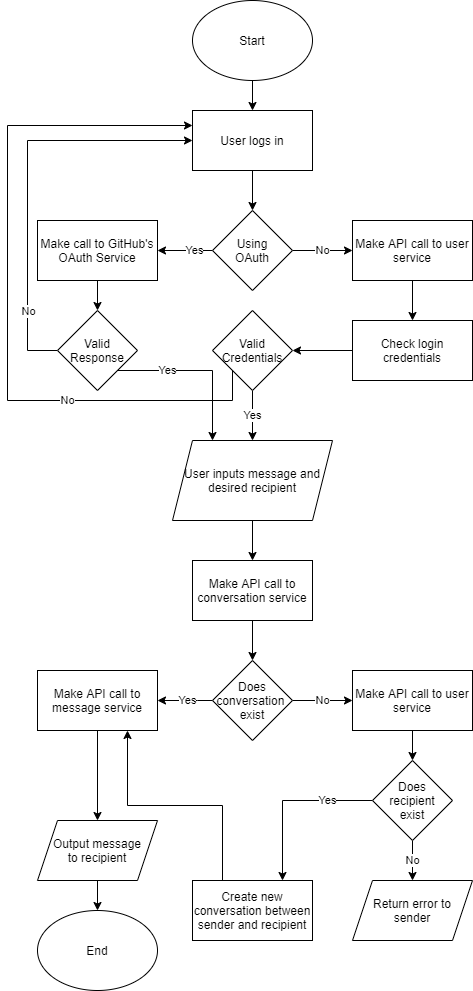
**Database ER Diagram:**

Timeline

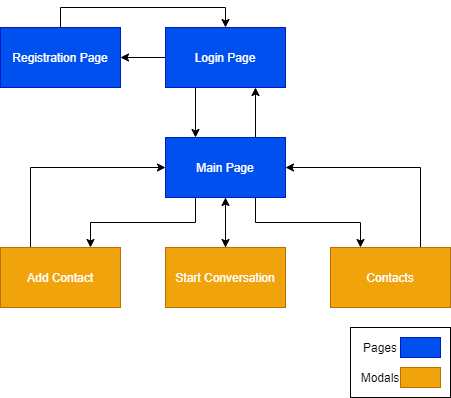
Description automatically generated

**Flow Charts/Process Flows:**

General Application Flow:

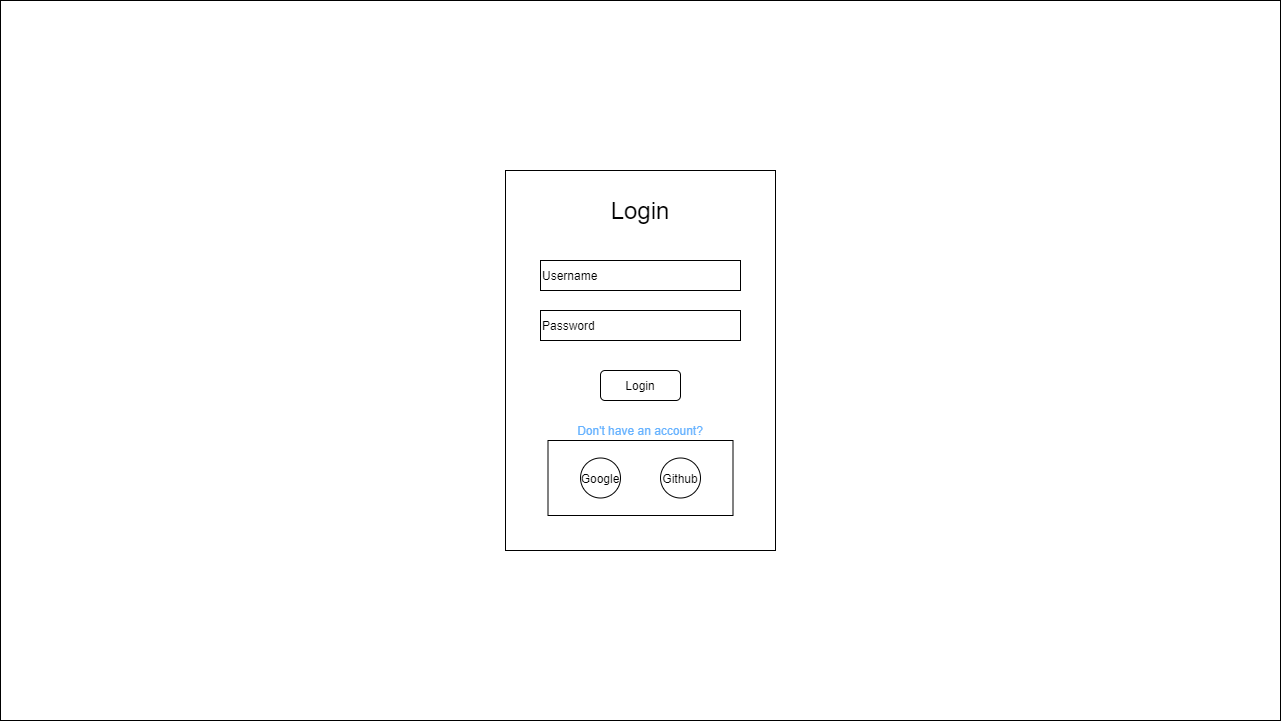


**Sitemap Diagram:**

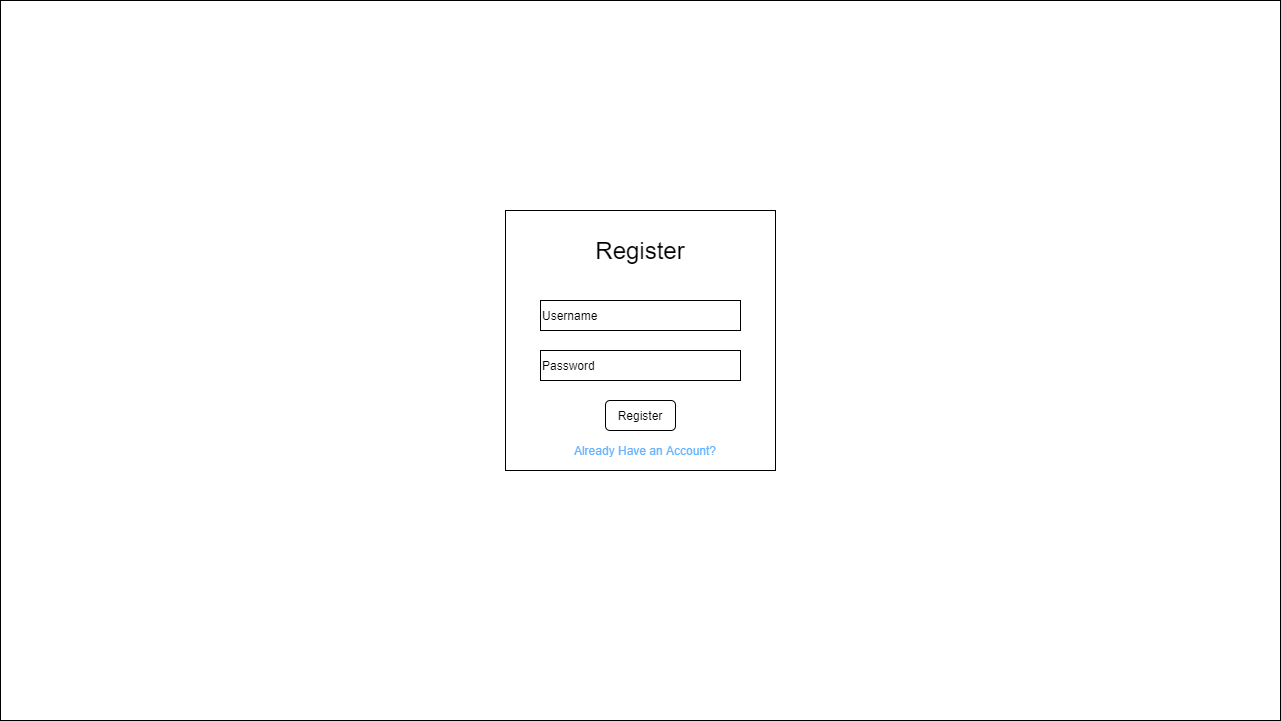


**User Interface Diagrams:**

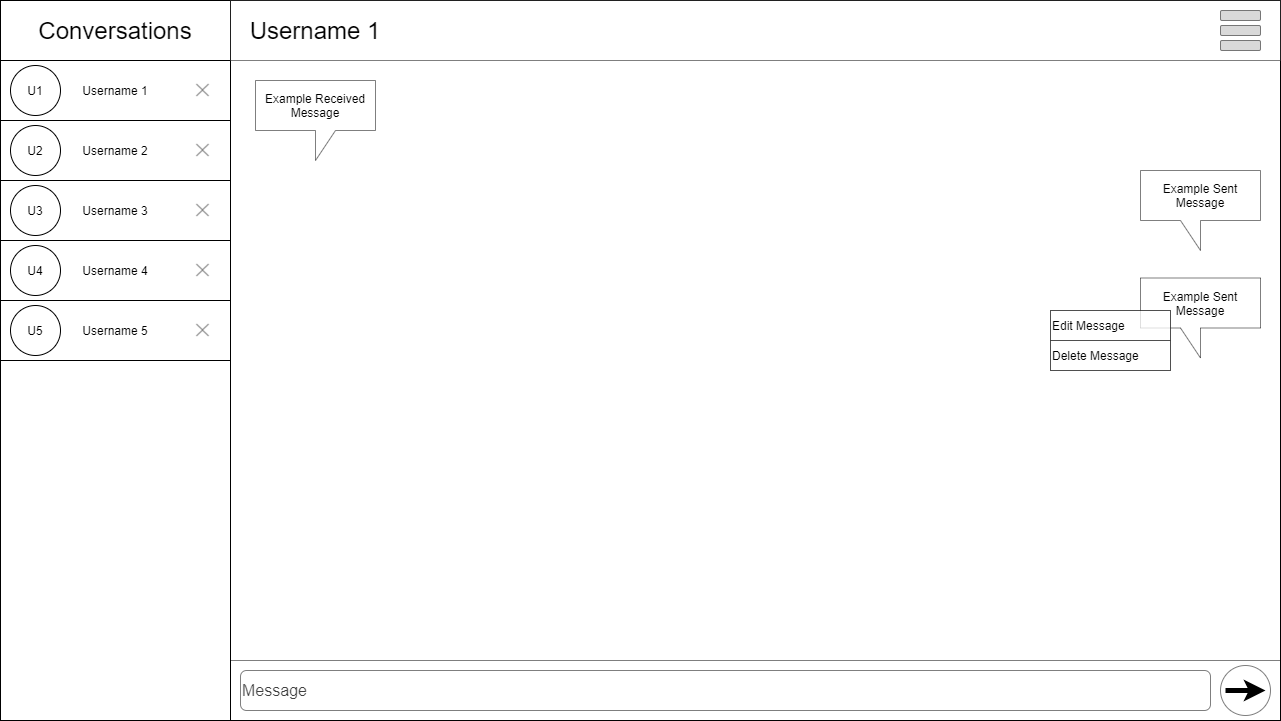
Login Page:



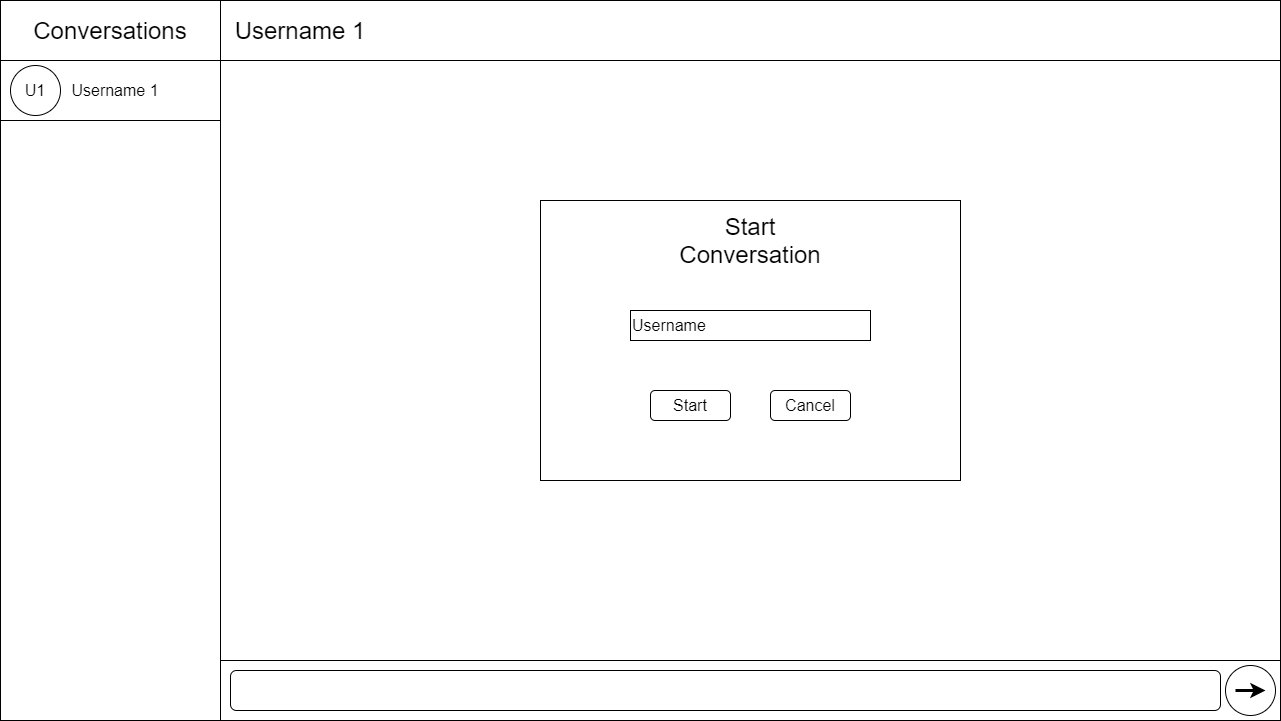
Registration Page:



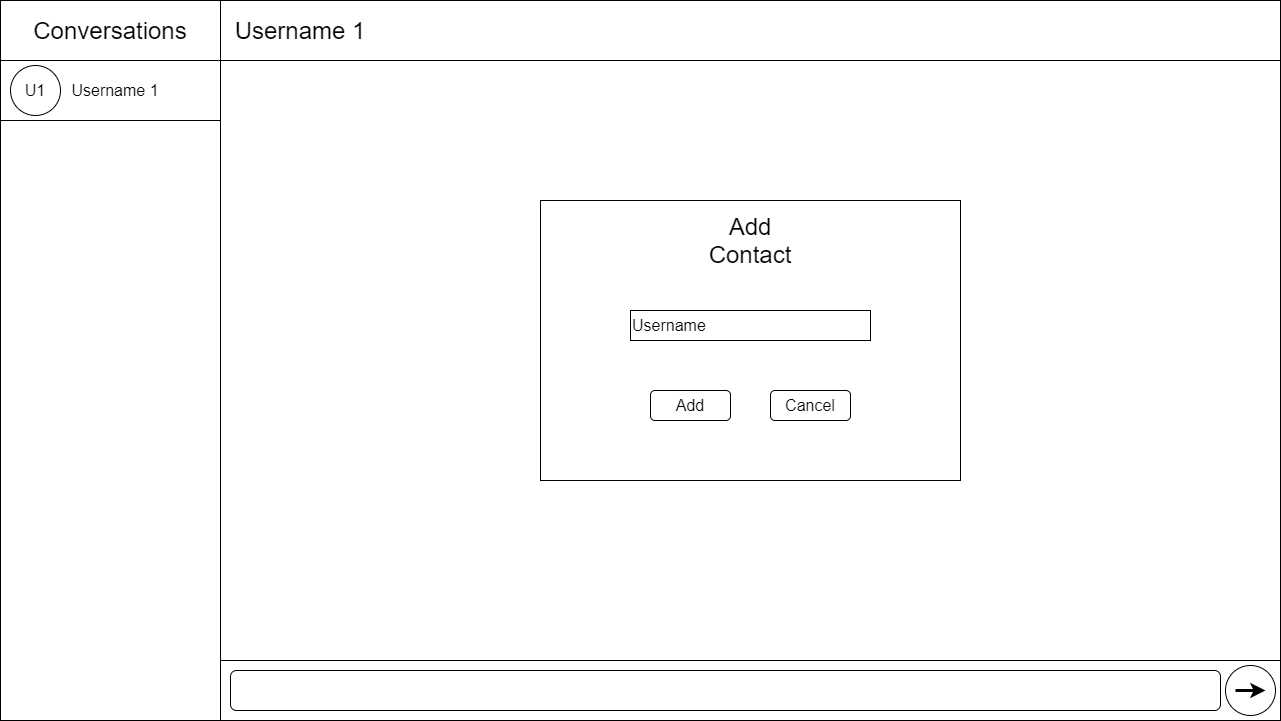
Main Page:



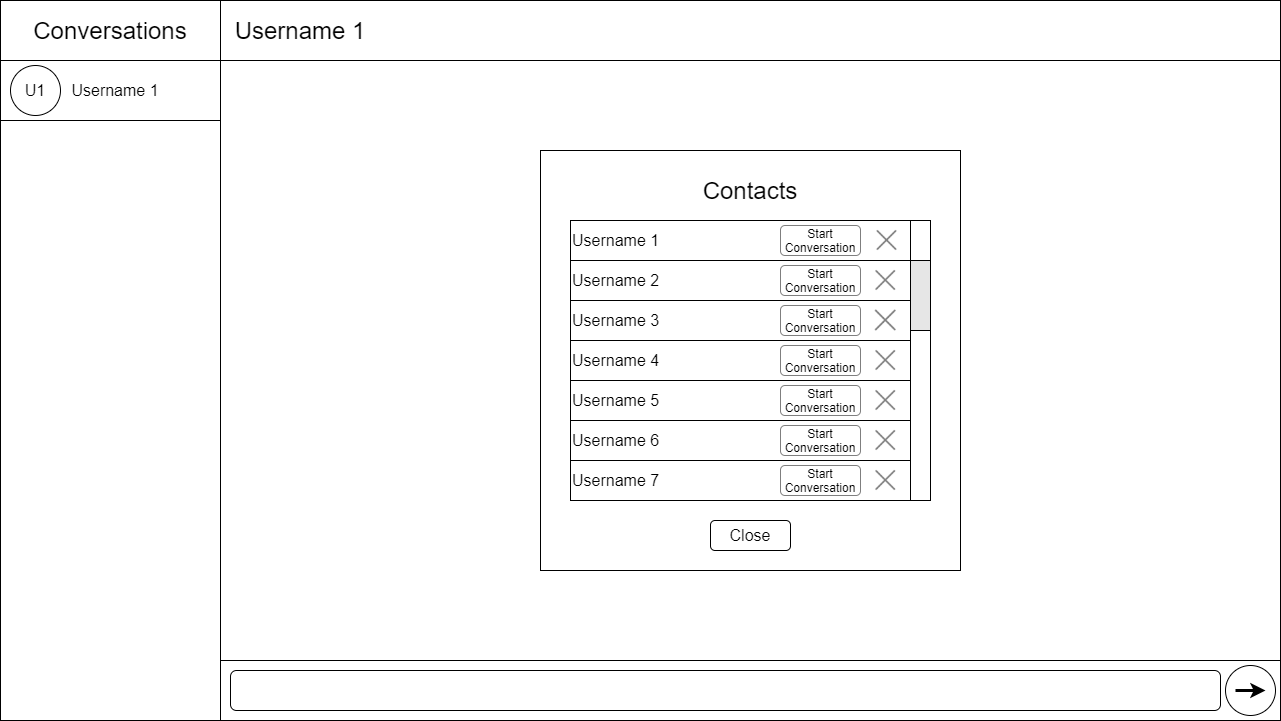
Start Conversation Modal:



Add Contact Modal:

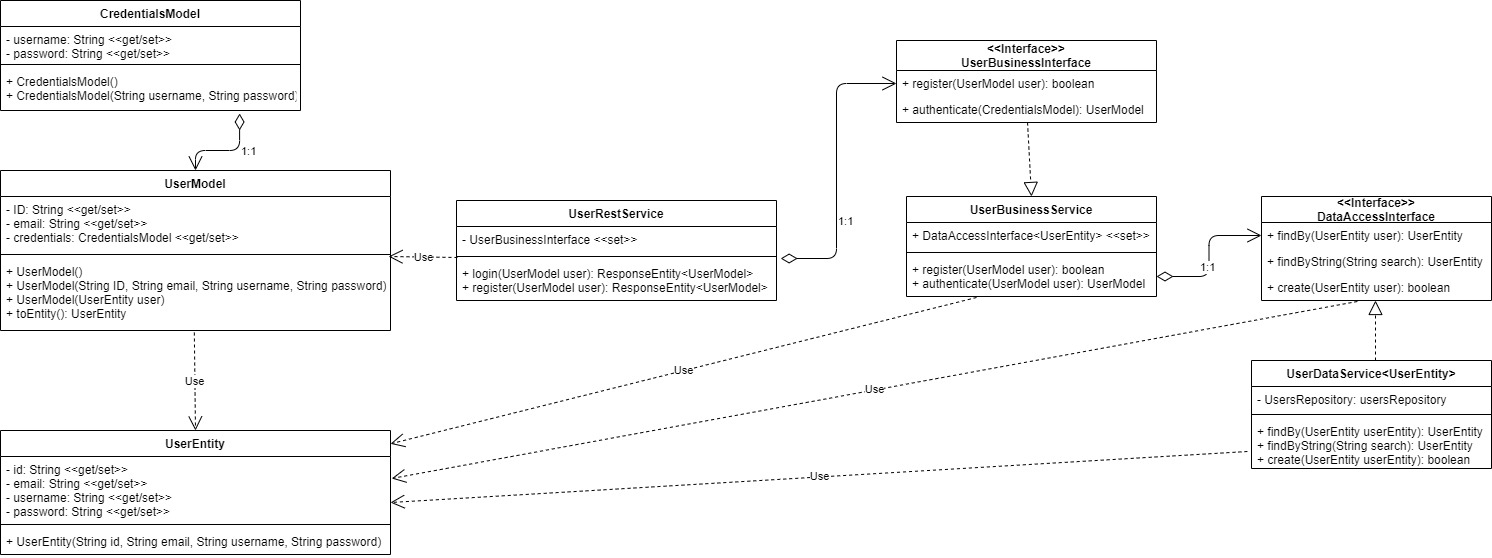


Contacts Modal:

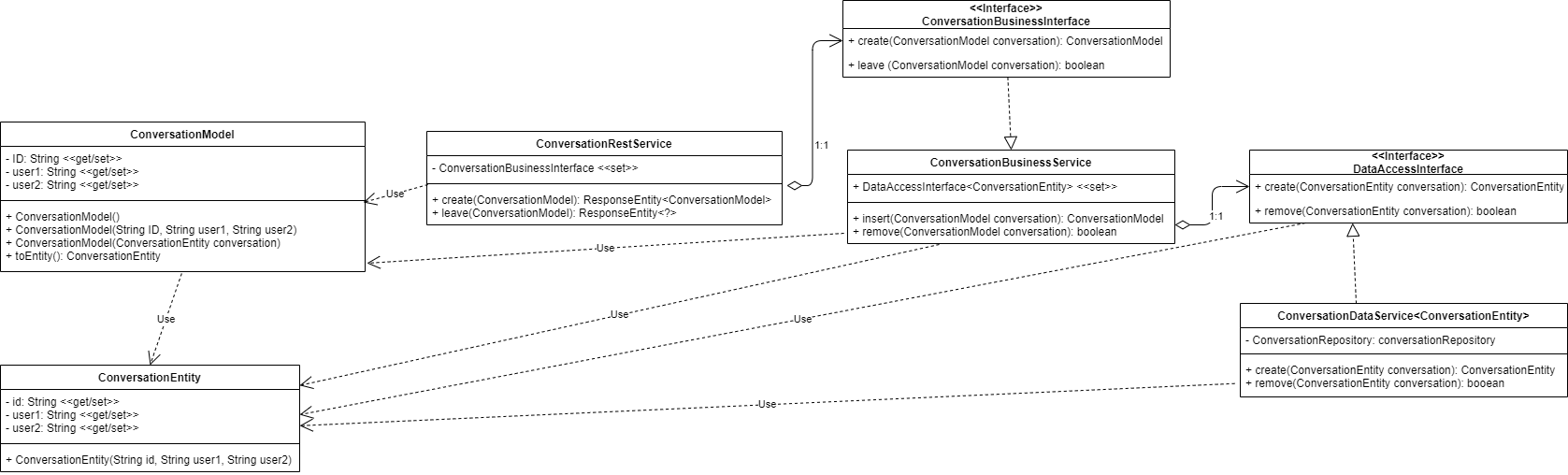


**UML Diagrams:**

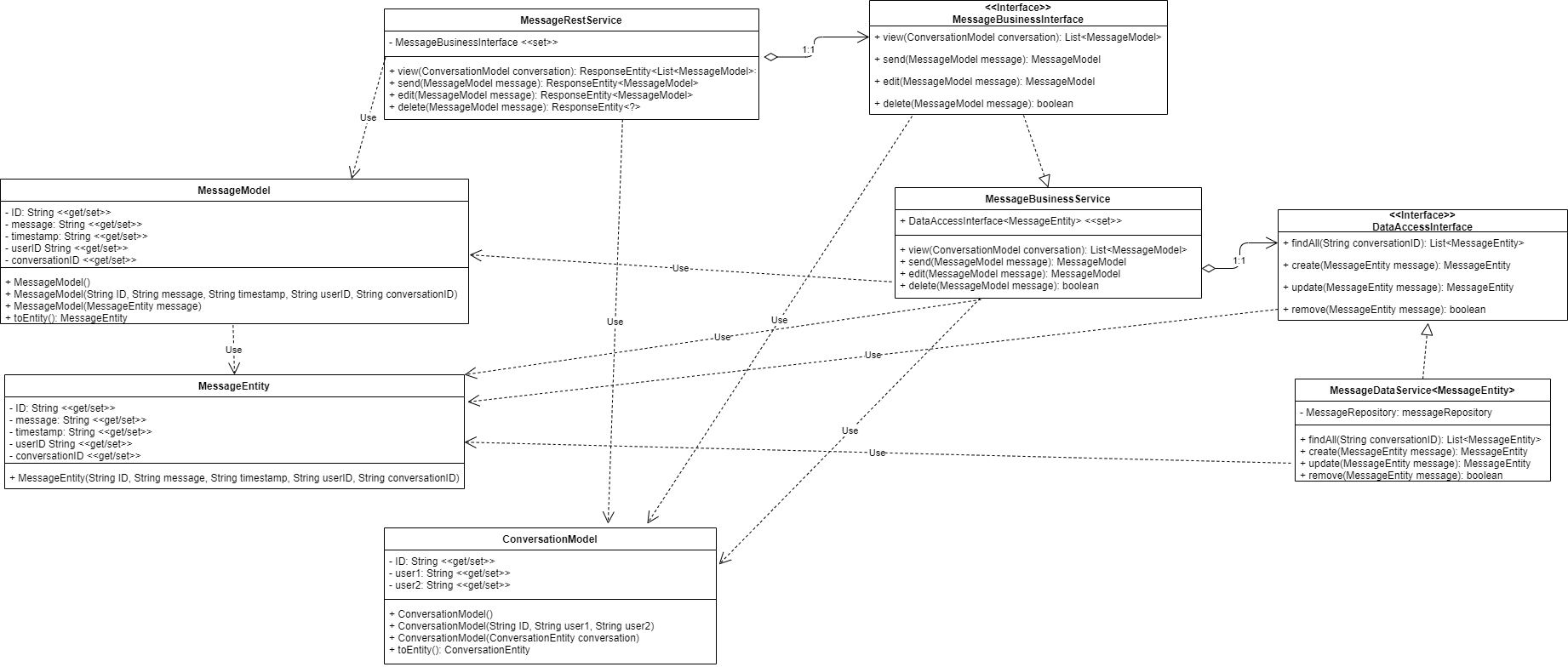
User Service:



Conversation Service:



Message Service:



**NFR’s:**

This project’s non-functional requirement of maintaining a 90% uptime will be supported by the projects design in two primary ways. First the projects physical design with its services hosted in Kubernetes containers via AWS allows for a certain degree of uptime sustainability as if there is ever a serious issue and a container becomes unavailable another container can be started in its place. Logging is the other key factor to supporting the project’s non-functional requirement as the external logging will allow for issue to be noticed and resolved quicker and may even allow for automation as an out-of-scope possibility allowing for new containers to be spun up automatically or issues to resolve without any real consequence.

**Operational Support Design:**

The applications only method of monitoring aside from diagnostics on the container supplied by AWS is logging. The application will be logged using log4j2 instead of Spring Boot’s default starter logging. In order to facilitate the use of log4j2 Spring Boot’s logger has to be excluded in the maven dependencies file and “spring-boot-starter-log4j2” will be included in its place. Log4j2 will then be configured to write logs to an external location using log4j tools.

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? |  |  |  |  |
| I | Service Discovery | Without implementing service discovery, the application’s services would be unable to interact with one another crippling the majority of the application’s functionality | Research a new service discovery solution such as Eureka | Brady Berner | High | 11/25/20 | 12/7/20 |  |
| R | CQRS Solution | Without implementing a CQRS or API composition solution it would not be possible to query from multiple databases at a time decreasing the functionality of the application or increasing the complexity. | Spend as much time as necessary to find, learn, and implement a CQRS or API composition solution to enable cross database queries | Switch to a relation database like MariaDB or MySQL | Medium | 9/29/20 | n/a | 11/5/20 |
| R | Saga Pattern | Without implementing the saga pattern into the application there is a good chance that discrepancies could occur between databases | Spend as much time as possible attempting to research and implement a saga pattern solution or spend time regularly checking for and fixing database discrepancies | Regularly check the databases for discrepancies and fix them manual. | Medium | 9/29/20 | 1/5/21 |  |
| R | WebSocket | If I am unable to learn/implement websocket protocol into my design, then I will need to use an alternative such as a REST API or event subscriber | Spend time researching and potentially working on proofs of concept to allow for implementation | Fall back to a known technology such as REST APis | Low | 9/29/20 | n/a | 10/16/20 |

Appendix B – References

Richardson, C. (2020). Microservice Architecture Diagram. https://microservices.io/i/Microservice\_Architecture.png

Appendix C – External Resources

|  |  |
| --- | --- |
| **GIT URL:** | *Not yet Applicable* |
| **Hosting URL:** | *Not yet Applicable* |