*An exercise app that elicits a sense of community for users who are new and used to going to the gym.*

*Team Tigahs*

Project Portfolio

*May 3rd, 202*

Introduction 2

Team Tigahs 3

System Requirements 3

User Stories 3

User Story #1 3

User Story #2 3

Project Management 3

Continuity of Operations Plan (COOP) 3

Project Plan

System Architecture Design and Development

System Implementation <Milestone 2: Architecture & Milestone 3: System Implementation>

Project Postmortem <Postmortem>

Project Wins

Root Cause Analysis

Lessons Learned

System Design <Milestone 2: System Architecture> 1

System Architecture <Milestone 2: System Architecture> 1

Component Design 1

Data Flow 1

System Components <Milestone 3: System Implementation> 1

Component [Component Name 1] 1

Component [Component Name 2] 1

Component [Component Name n] 1

Design Pattern <Milestone 3: System Implementation> 1

Design Pattern <Milestone 3: System Implementation> 2

System Implementation <Milestone 3: System Implementation> 3

# Introduction

One of the biggest challenges people face when trying to start working out is figuring out what exactly they need to do to start out on the right foot. Attempting to learn it all on your own can be very intimidating, and it has led many people to give up on even trying to work out at all. New gym-goers aren’t the only group of people who struggle with going to the gym either. Experienced people also struggle to get to go to the gym consistently, to work out properly, and to find a sense of community that helps them to feel like they belong.

Our goal for this app is to create a social fitness app that matches users based on their fitness goals, workout routines and schedules. The app would allow users to search and connect with workout partners, trainers and like-minded individuals in their local gym or fitness community. In order to make this happen, we may make use of technologies such as as JavaScript, Flutter, Dart, CSS, or HTML to create a website and a mobile apps while also using MySQL for database management. Java or C# will more likely than not be our OOP language of our choice for connecting the website and mobile apps to the database for retrieval and input of user data.

Core Features:

* Profile Creation: Users will be able to create a profile that includes general information such as their age, sex, height, weight, etc. A profile picture and a bio will also be provided, should the users want to include more information. On top of height and weight, users can also supply a series of basic exercises so others have an idea of what their physical build is by analyzing their individual muscle groups and body as a whole.
* Matching Algorithm: The tool would use a matching algorithm to connect users with workout partners who share similar stats. The algorithm will be customizable so that they can choose whether they want to meet people who are more or less similar in physical stats. The algorithm will also provide the user with the ability to match with other users in a specific muscle group instead of only being able to match with someone while considering all muscle groups.
* Messaging: Based off the user matches derived from the algorithm, users will be able to chat with others as well as add them to their friends list to allow them to meet up with each other.

Viable Features:

* Social Feed: The app could have a social feed where users can post about their workouts, share progress, and connect with other users. The feed could also be used by trainers to share tips and insights on fitness and wellness.
* Workout and Training Plans: The app could provide users with personalized workout plans and training programs tailored to their fitness goals and level of experience. Trainers could also create and share custom training plans with their clients.

Stretch Features

* In-app Purchases: The app could offer premium features, such as customized workout and training plans, access to exclusive workout classes, and personalized coaching from trainers, for a fee.
* The app could also have a calendar feature to help users keep track of their scheduled workouts.
* Safety Features: The app would need to have safety features including but not limited to guidelines for meeting up with workout partners, and reporting tools to report any suspicious behavior.

# Team Tigahs

Aidan Eiler – Team Leader, Project Owner, System Architect, Backend Developer, Database Developer

Jonathan Morse – SCRUM Master, Lead Web Developer, Spring Boot Research and Implementation

Tiffany Scroggins – System Architect, Web Developer , Wireframe Designer

Tyler Stephens – Web Developer, Wireframe Designer

Reece Hernandez  - Backend Developer, Spring Boot Research and Implementation

Jack Legnon – Backend Developer, Database Developer, Wireframe Designer

Project GitHub Link: https://github.com/Gr8Potato/CSC-3380-Project

# System Requirements

## User Stories

### User Story #1

As an average gym goer, I want to be able to connect with like-minded individuals of similar physique (stats, build, etc.), so I can get the best results and socialize with other people.

### User Story #2

As a person interested in working out, I want to be able to network with experienced gym enthusiasts, so I can learn from someone and start my workout journey on the right foot.

# Project Management

## Continuity of Operations Plan (COOP)

We have created a Discord server which we will use for accessible communication, both text, audio, and video. We also will have weekly or bi-weekly in-person meetings to update one another on progress as well as to discuss any issues we may be having. We're considering practicing paired programming, in order to ensure everyone has the capability to produce quality work, as well as dividing work amongst the team, attempting to cater towards our natural skill sets. Given our team has an odd number of members, it may be the case that a group of three is formed or one person works on a particular component individually.

Using Discord as our primary form of communication throughout the project allows us to be very flexible should unforeseen issues arise. If we can’t meet in person for an extended period of time, we will use our Discord's audio chat feature in place of our in-person meetings. If, for whatever reason, it's the case that one or more of our members is unable to contribute to the project, whether it be temporarily or permanently, we can expect the other in the programming pair to take over the work of whoever can't, provided the work doesn't become arduous, and if that become the case, we will begin adapting to fulfill the project's goals and its trajectory.

## Project Plan

### System Architecture Design and Development

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **#** | **Activity** | **Pre #** | **Estima-ted**  **Effort (hrs)** | **Actual**  **Effort** | **Estimated**  **Start Date**  **(MM/DD/YYYY)** | **Estimated**  **Finish Date**  **(MM/DD/YYYY)** | **Actual**  **Start Date** | **Actual**  **Finish Date** |
| 1.0 | Learn EA. |  |  |  |  |  |  |  |
| 1.1 | Attend Class on Thursday. |  | 1.5 | 1.5 | 2/9/2023 | 2/9/2023 | 2/9/2023 | 2/9/2023 |
| 1.2 | Have all members complete EA assignment(s). |  | 5 | 3 | 2/9/2023 | 2/10/2023 | 2/10/2023 | 2/12/2023 |
| 1.3 | Spend time practicing EA outside of homework hours. |  | 5 |  | 2/10/2023 | 2/11/2023 | 2/10/2023 | 2/11/2023 |
| 2.0 | Determine System Architecture |  |  |  |  |  |  |  |
| 2.1 | Determine which components might be needed |  | 10 | 1.5 | 2/10/2023 | 2/13/2023 | 2/10/2023 | 2/13/2023 |
| 2.2 | Evaluate the benefits and drawbacks of each system architecture | 1.1 | 10 | 0.5 | 2/13/2023 | 2/14/2023 | 2/13/2023 | 2/13/2023 |
| 3.0 | Integrate System Architecture to Project Design |  |  |  |  |  |  |  |
| 3.1 | Determine data flow | 2.1,2.2 | 10 | 2 | 2/14/2023 | 2/17/2023 | 2/13/2023 | 2/16/2023 |
| 3.2 | Generate a rough draft for different machine states | 2.1,2.2 | 15 | 2 | 2/17/2023 | 2/20/2023 | 2/13/2023 | 2/16/2023 |
| 3.3 | Refine components if need be | 3.2 | 3 | 4 | 2/20/2023 | 2/21/2023 | 2/17/2023 | 2/23/2023 |
| 3.4 | Integrate Fully Fleshed System Architecture into EA | 3.3 | 25 | 4 | 2/21/2023 | 2/23/2023 | 2/22/2023 | 2/23/2023 |
| 3.5 | Begin prototyping | 3.4 | 8 | 8 | 2/22/2023 | 2/23/2023 | 2/22/20203 | 2/23/2023 |

### System Implementation

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **#** | **Activity** | **Pre #** | **Estimated**  **Effort** | **Actual**  **Effort** | **Estimated**  **Start Date** | **Estimated**  **Finish Date** | **Actual**  **Start Date** | **Actual**  **Finish Date** |
| 1.0 | Understand class diagrams (in EA) |  |  |  |  |  |  |  |
| 1.1.0 | Review class diagram slides |  | 1 | 1 | 2/27/2023 | 2/27/2023 | 2/27/2023 | 2/27/2023 |
| 1.1.1 | Do EA HW #2 | 1.1.0 | 3 | 2 | 2/27/2023 | 2/27/2023 | 2/27/2023 | 2/27/2023 |
| 1.2 | Spend time outside of class studying EA (for class diagrams for MS #3) |  | 2 | 4 | 2/27/2023 | 2/27/2023 | 2/27/2023 | 2/28/2023 |
| 2.0 | Research Git/GitHub |  | 10 | 15 | 2/27/2023 | 2/29/2023 | 2/28/2023 | 3/6/2023 |
| 3.0 | Implement DBM |  |  |  |  |  |  |  |
| 3.1 | General format queries |  | 10 | 10 | 2/28/2023 | 3/2/2023 | 2/28/2023 | 3/3/2023 |
| 4.0 | Research ReactJS and other web dev tools |  |  |  |  |  |  |  |
| 4.1 | Watch videos |  | 10 | 10 | 2/28/2023 | 3/5/2023 | 3/3/2023 | 3/5/2023 |
| 4.2 | Test in web dev playground | 4.1 | 15 | 20 | 3/1/2023 | 3/8/2023 | 3/5/2023 | 3/10/2023 |
| 5.0 | Implement Auth Manager |  |  |  |  |  |  |  |
| 5.1 | Tailor query results |  | 4 | 3 | 3/10/2023 | 3/11/2023 | 3/10/2023 | 3/11/2023 |
| 5.2 | Conditional branches | 5.1 | 1 | 1 | 3/12/2023 | 3/14/2023 | 3/12/2023 | 3/13/2023 |
| 5.3 | UI output | 5.2 | 10 | 10 | 3/14/2023 | 3/18/2023 | 3/14/2023 | 3/20/2023 |
| 6.0 | Implement account and profile creation |  |  |  |  |  |  |  |
| 6.1 | Register credentials | 5.0 | 5 | 5 | 3/18/2023 | 3/21/2023 | 3/20/2023 | 3/21/2023 |
| 6.2 | Signing with credentials | 6.1 | 5 | 5 | 3/21/2023 | 3/25/2023 | 3/21/2023 | 3/25/2023 |
| 6.3 | Profile page | 6.2 | 10 | 9 | 3/25/2023 | 3/27/2023 | 3/25/2023 | 3/28/2023 |
| 7.0 | Implement messaging module |  |  |  |  |  |  |  |
| 7.1 | Tailor queries | 6.0 | 5 | 4 | 4/1/2023 | 4/5/2023 | 3/23/2023 | 3/24/2023 |
| 7.2 | UI output | 6.1 | 10 | 13 | 4/6/2023 | 4/8/2023 | 3/23/2023 | 3/25/2023 |
| 8.0 | Implement matchmaking algorithm |  |  |  |  |  |  |  |
| 8.1 | Tailor stats | 7.0 | 3 | 15 | 4/10/2023 | 4/10/2023 | 3/27/2023 | 4/4/2023 |
| 9.0 | Testing core features | 2.0-8.0 | 30 | 40 | 3/1/2023 | 4/11/2023 | 3/15/2023 | 4/11/2023 |
| 10.0 | Consider viable/stretch fractures | 9.0 | 10 | N/A | 3/28/2023 | 3/30/2023 | N/A | N/A |
| 11.0 | Implement chosen viable/stretch features | 10.0 | 20 | N/A | 3/31/2023 | 4/7/2023 | N/A | N/A |
| 12.0 | Test viable/stretch features | 11.0 | 20 | N/A | 4/8/2023 | 4/11/2023 | N/A | N/A |
| 13.0 | Do MS #3 per guidelines | 12.0 |  |  |  |  |  |  |
| 13.1 | Finalize GitHub | 12.0 | 5 | 8 | 4/10/2023 | 4/11/2023 | 4/11/2023 | 4/11/2023 |
| 13.2 | Complete Template | 12.0 | 20 | 10 | 4/10/2023 | 4/11/2023 | 4/11/2023 | 4/11/2023 |

## Project Postmortem <Postmortem>

### Project Wins

[Provide a bulleted list of at least 3 positive aspects of the project.]

### Root Cause Analysis

[Provide a bulleted list of at least 3 negative aspects of the project. For each negative, provide the answer to the three successive “Why” questions. ]

### Lessons Learned

[For each negative aspect identified in the Root Cause Analysis, provide a mitigation strategy (i.e., what process should be introduced) to ensure that the problem is not repeated in subsequent projects.]

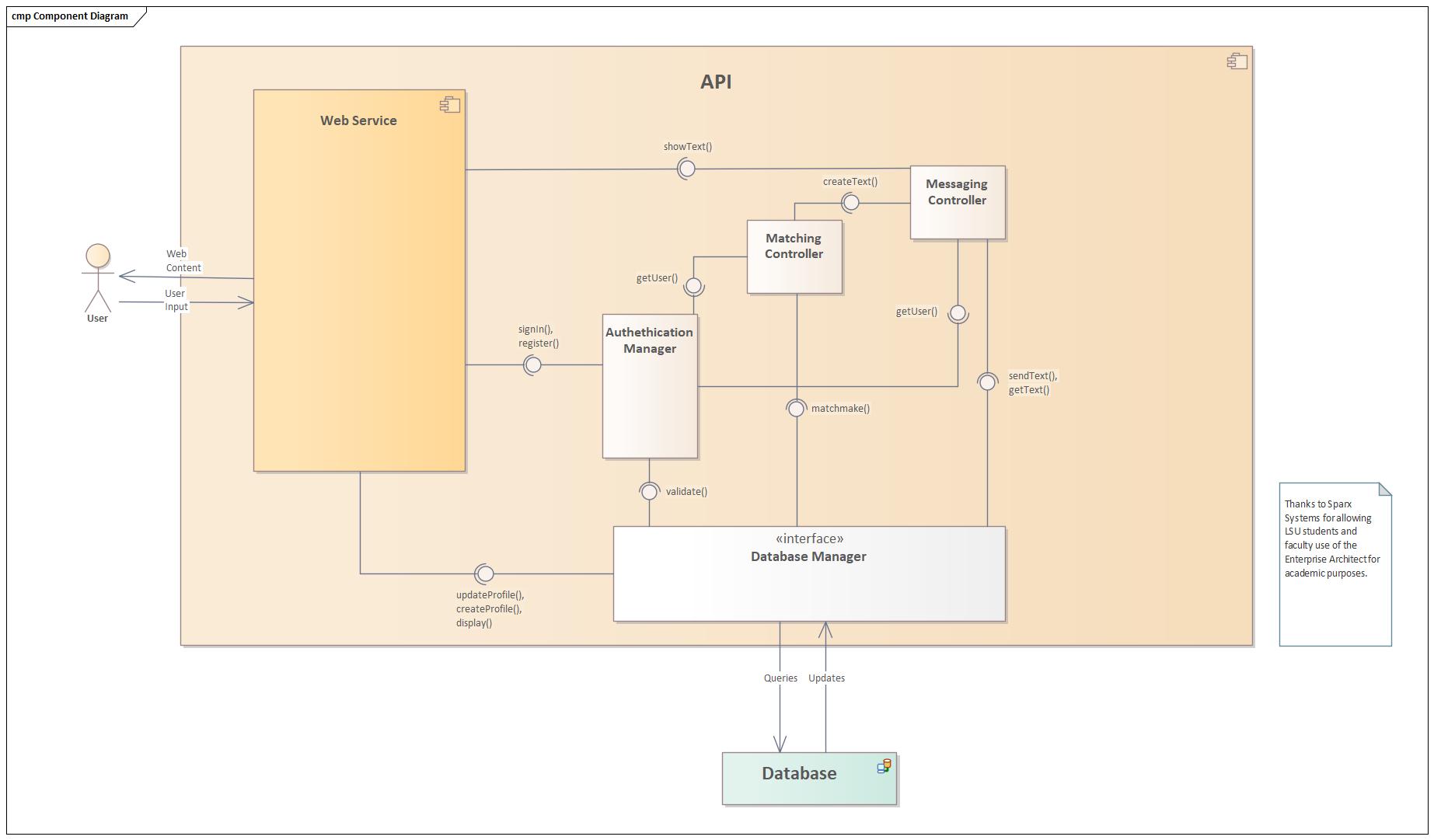
# System Design

The Authentication Manager provides the user full access to the Matchmaking Controller and the Messaging Controller as well as their profile, all of which is done through the Database Manager, which is responsible for interacting the database directly through the repositories. The Matchmaking Controller provides the results of the match while also allowing the user to start a chat with matches, and the Web Service component provides the UI for all of these interactions to allow the user to easily see their matches, add friends, and start conversations.

## System Architecture

We are using a three-tier server/client architecture, where the user acts as a client through the website, which interacts with the API to provide information and algorithms using the MySQL server as well as the multiple repositories that are responsible for interacting with that server. The API retrieves database information, acting as a client through the repositories, which acts as a server to the user by sending the retrieved information back to the user.

### Component Design



### Data Flow

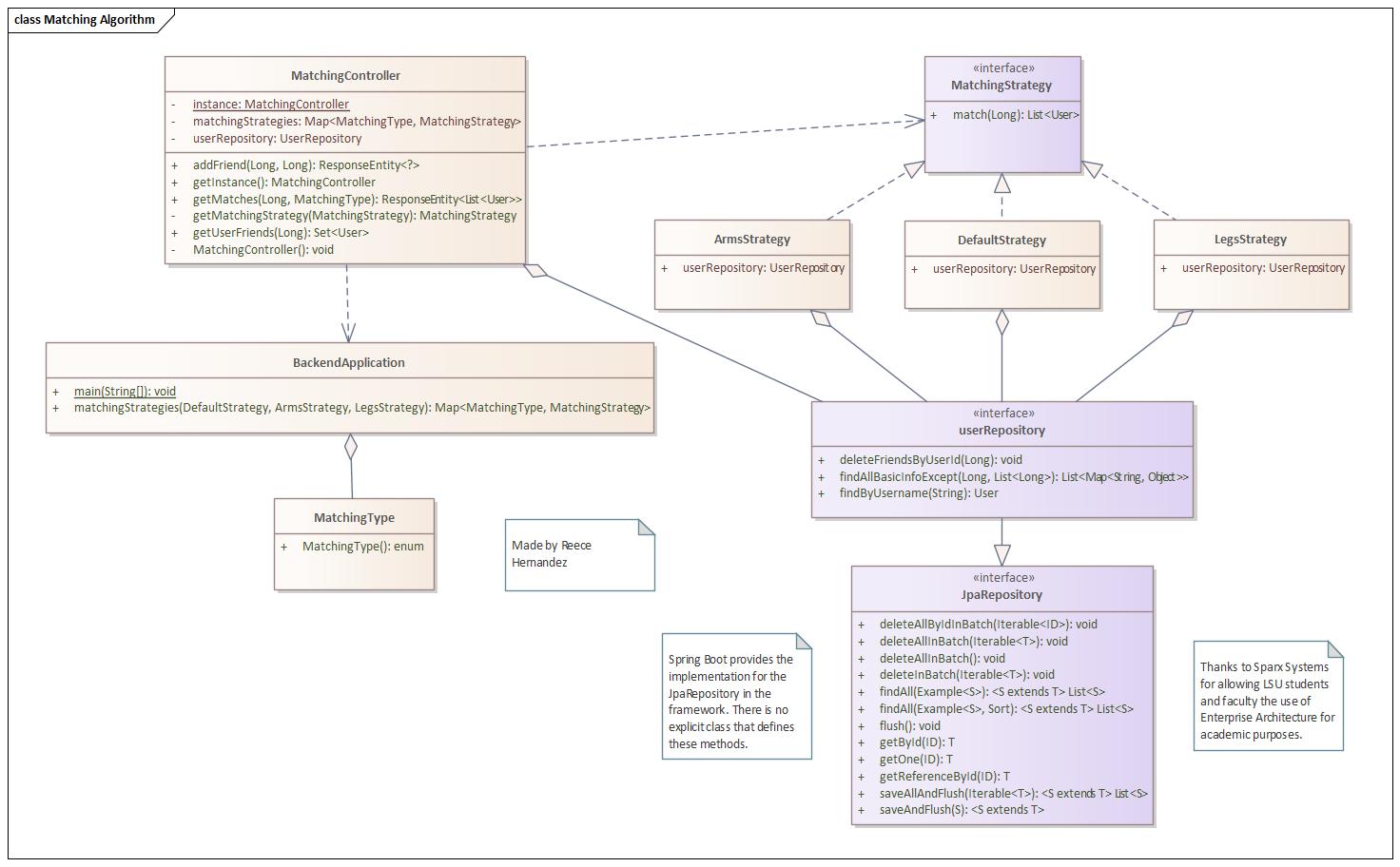
Diagram

Description automatically generated

The Database Manager is responsible for sending and receiving raw SQL data. The Authentication manager, Matching Controller, and Messaging Controller components take semi-refined data from the Database Manager and tailors it depending on why the components make the update/query requests. The Web Service component sends credential information to the Authentication Manager and receives a rejection/accept as a result. The Authentication Manager informs multiple components of the active user, as that will reflects on what the user sees and the outputs of the Matchmaking Controller and Messaging Controller among others.

## System Components

### Component [Matching Algorithm]

The Matching Algorithm component takes Users and compares their stats to check if they are a valid match or not using Euclidean Distance. It queries the database for a user to compare to the logged in user and offers the user the option to message their valid matches. We implemented the strategy design pattern for this component to handle requests to match users based on stats for full-body, arm, and leg workouts. This component implements the singleton design pattern to ensure that only one instance of the manager is present during runtime, which is beneficial in terms of memory and readability.

### Component [Authentication Manager]

The Authentication Manager compares login information inputted by the user and compares it to users in the database to enforce one of the three factors of authentication: what the user knows. The user’s information will be handled by other parts of the program. It also handles the creation of new accounts by updating the database with use information when a new user is made. This component implements the singleton design pattern to ensure that there is only one instance of the manager present at runtime.

### 

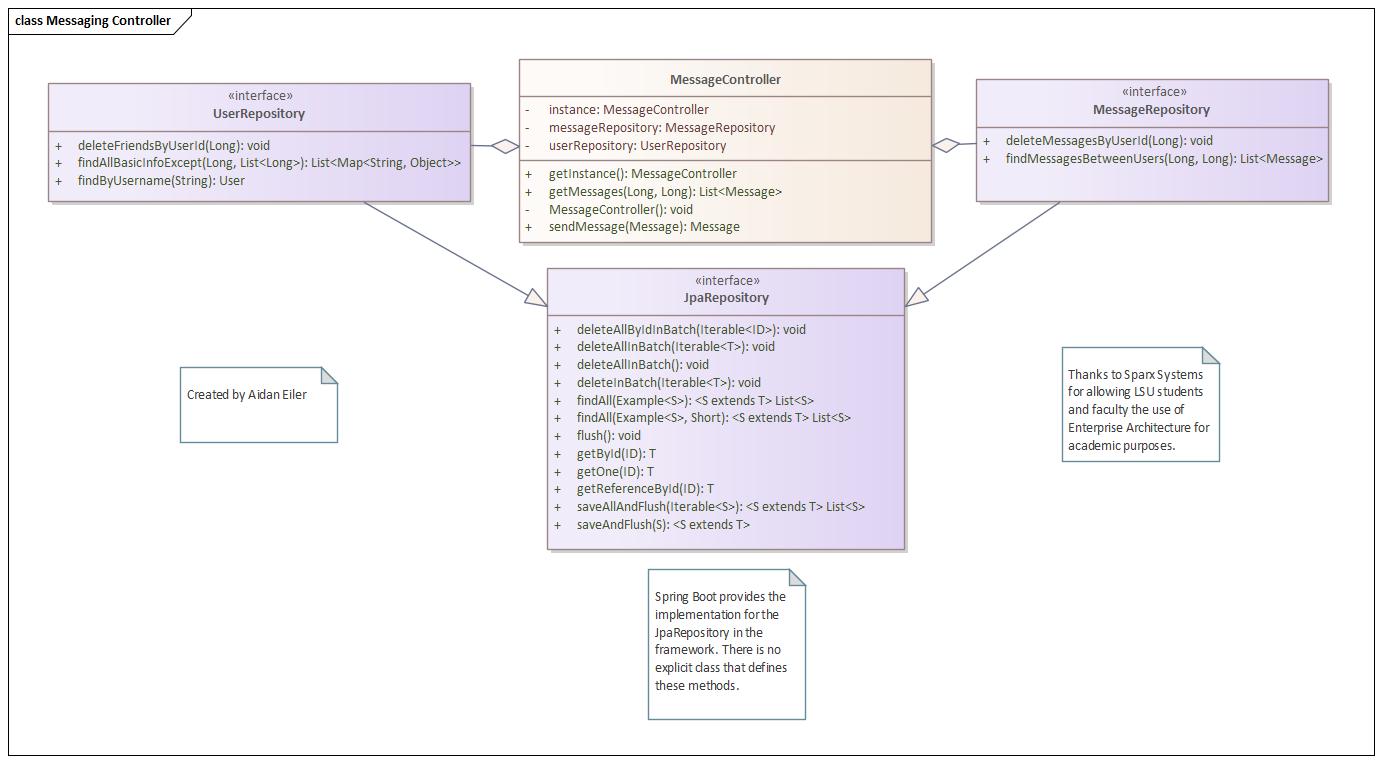
### Component [Database Manager]

The database manager interacts with the MySQL database whenever other classes or the Spring Boot framework call methods that send queries and updates to the database. This component implements the singleton design pattern to ensure that there is only one instance of the manager at runtime.

### 

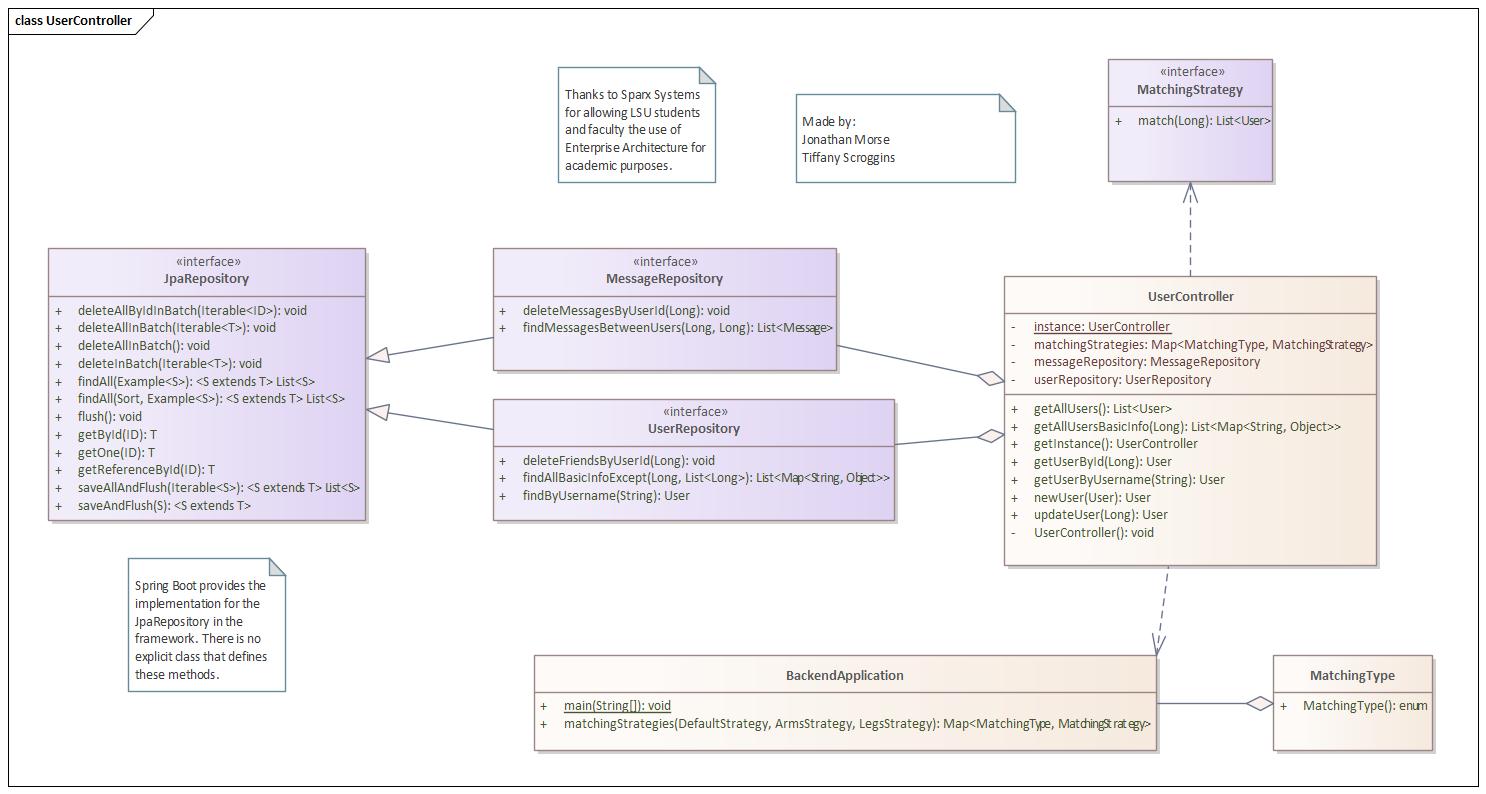
### Component [Messaging]

The Messaging component allows users to send and receive messages from one another while also saving and creating conversations in the database. It receives a request to create a conversation when a match is found and the user wants to add the match to their friends list. This component implements the singleton design pattern so that there is only on instance of the messaging manager at runtime.

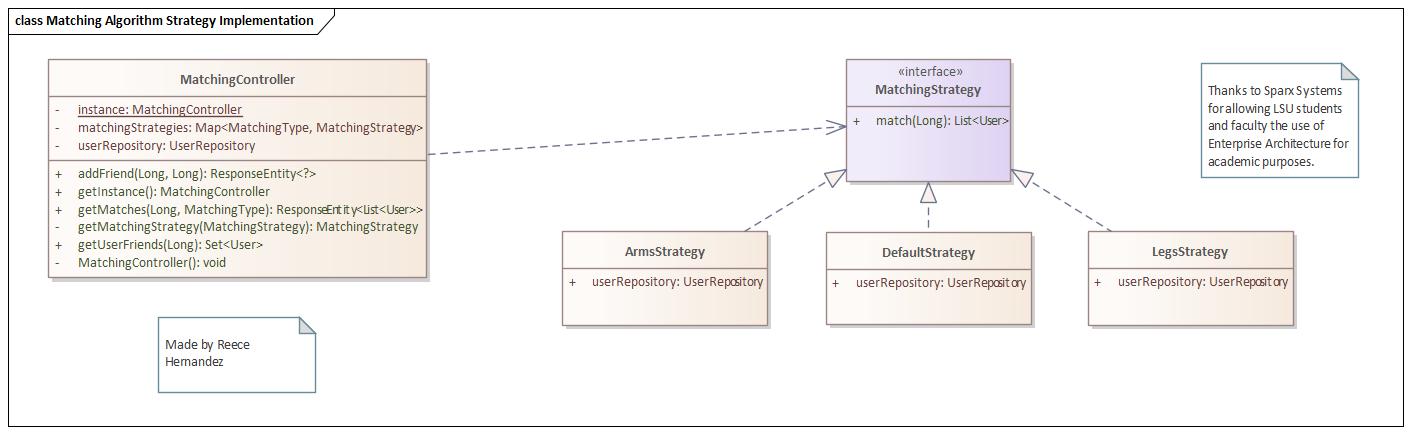


### Component [User Controller]

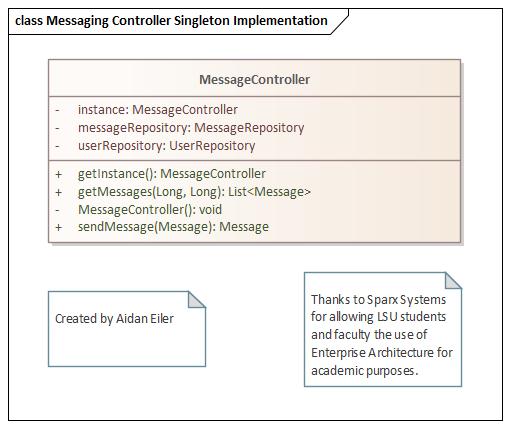
The User Controller handles all queries that have to do with user profile information, including friends. It updates and retrieves user information through the user repository and does the same with messages through the message repository. It also handles requests to changes to the messages and friends of the user.



## Design Pattern (Strategy)



## Design Pattern (Singleton)



# System Implementation

|  |  |  |
| --- | --- | --- |
| **Architectural Component** | **Programming Language(s) %** | **Team Member(s) %** |
| Web Service | Javascript (90%) (Javascript objects are used for frontend, and Java components are invoked for logic)  HTML (7%)  CSS (3%) | Jonathan Morse (60%)  Tiffany Scroggins (15%)  Tyler Stephens (12.5%)  Jack Legnon (12.5%) |
| Matching Algorithm | Java (100%) | Reece Hernandez (25%)  Aidan Eiler (30%)  Jonathan Morse (30%)  Jack Legnon (15%) |
| Messaging | Java (100%) | Jonathan Morse (40%)  Tyler Stephens (30%)  Tiffany Scroggins (30%) |
| Database Manager | Java (100%) | Aidan Eiler (45%)  Jack Legnon (45%)  Jonathan Morse (10%) |
| Authentication Manager | Java (100%) | Reece Hernandez (25%)  Jonathan Morse (25%)  Tyler Stephens (25%)  Tiffany Scroggins (25%) |