0. Table of Contents

1. Introduction

1.1 Overview

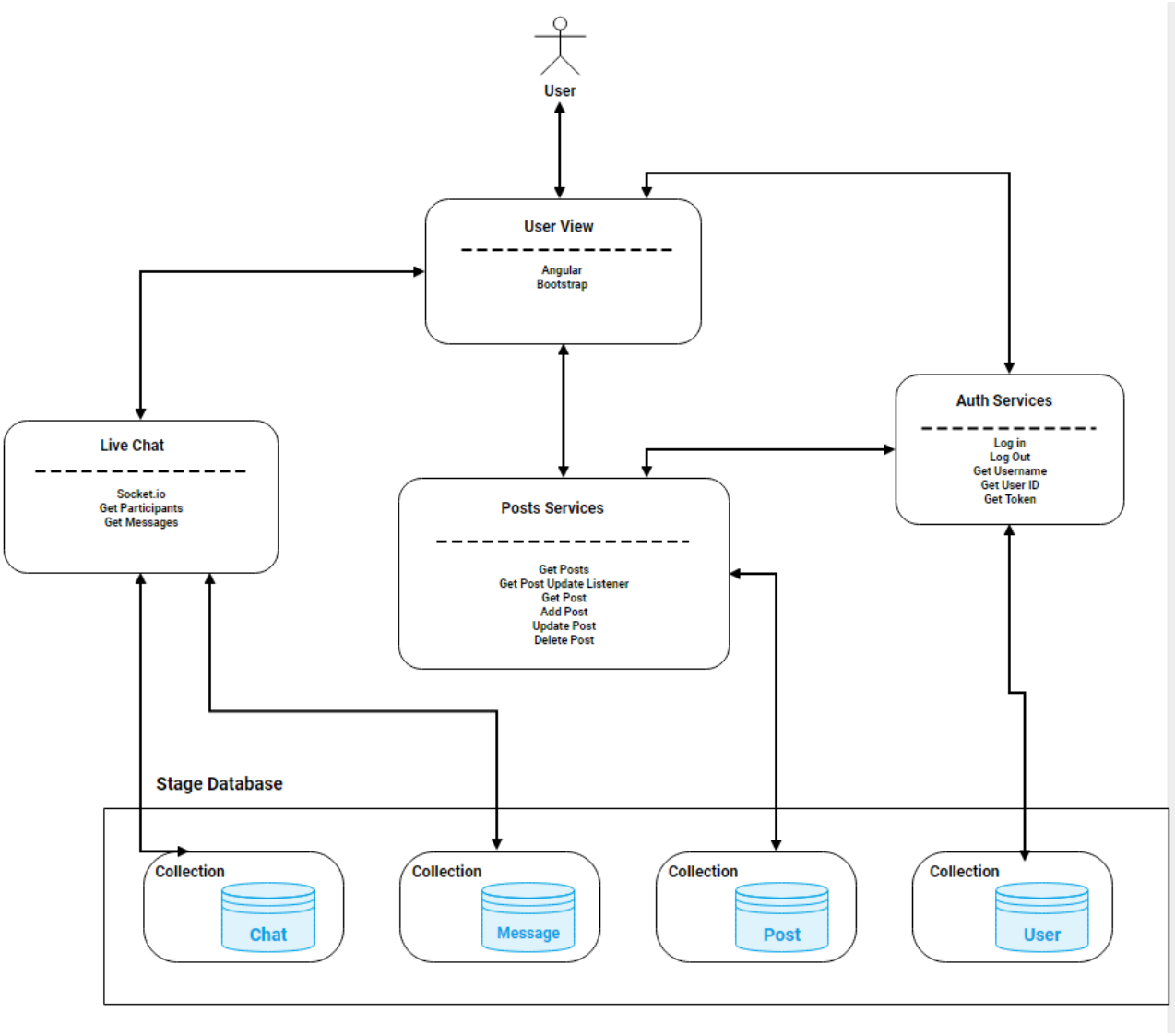
PASII is an online community for the clubs and societies of DCU to communicate with each other. The aim of the project was to create a website for club and society members to organise events, send each other messages, post message board style threads and chat to each other via an instant messaging service.
Our goal was to deliver a vibrant and responsive user interface with a simple and navigable design. From our research we have determined the features we plan on implementing are extremely appealing to our target audience. The system that was developed allows users to register an account, log in, chat in real time to another user, browse registered clubs, create new posts and edit their existing posts.

1.2 Glossary

HTML: Short for Hypertext Markup Language. Used for website creation.
MEAN: Short for MongoDB, Express.js, Angular, and Node.js.
MongoDB: A non-relational database.
Mongoose: Object data modelling framework for MongoDB
Node.js: A JavaScript runtime environment.
Express.js: A JavaScript framework used for building APIs
Angular: A front-end web application framework.
JavaScript: A core technology of websites. It is a dynamic, interpreted programming language.
TypeScript: A syntactical subset of JavaScript for large scale web applications.

Socket.**io**: A JavaScript library for real-time web applications.

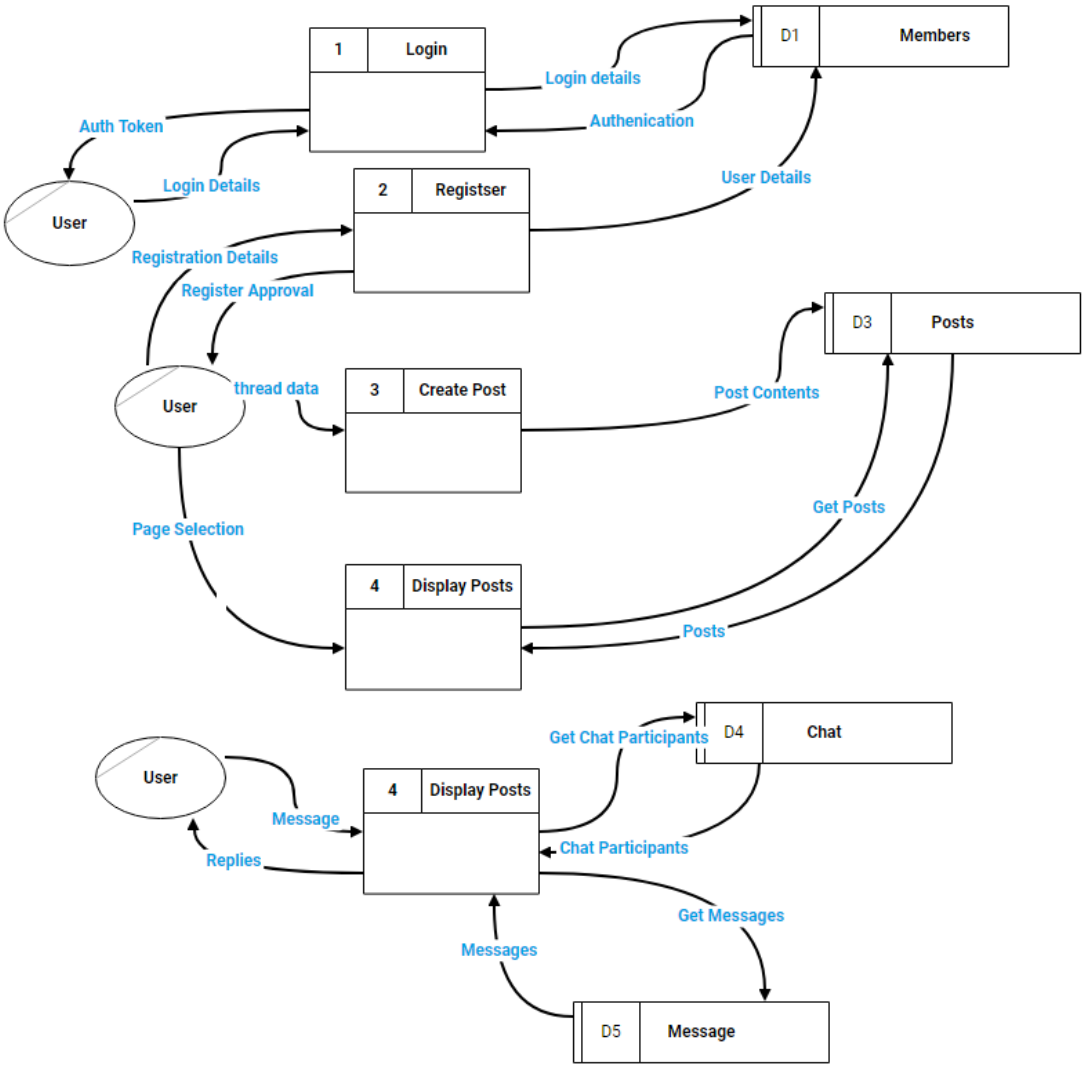
2. System Architecture



This System architecture diagram shows how the user interacts with the system and the steps that the users data takes before it reaches the database. There are 4 separate mongoose schemas. The chat schema holds all the current live chats that are active. The Message schema holds all the messages that have been written with a reference to which chat they a part of. The Post schema holds the data for the message board posts, including the title, the body, and the author. The posts are accessed through the post service, whenever the getPost() method is called. The User schema holds user IDs, usernames, email addresses and passwords. The auth service uses this data to authorise users that are attempting to log in. The auth service generates a token that is appended to the users data when they are navigating the website. This ensures that they remain logged in even after refreshing the page; it is valid for 24 hours.

3. High Level Design

Document Flow Diagram



This diagram is a lower level of abstraction than the previous one. It shows how the user interacts with each of the main processes that take place in this system.
The user sends login details to the login process which then compares the bearer token to the one found in the Members database, along with. If a match is found, an authentication token is sent back to the user, and stored in localstorage.

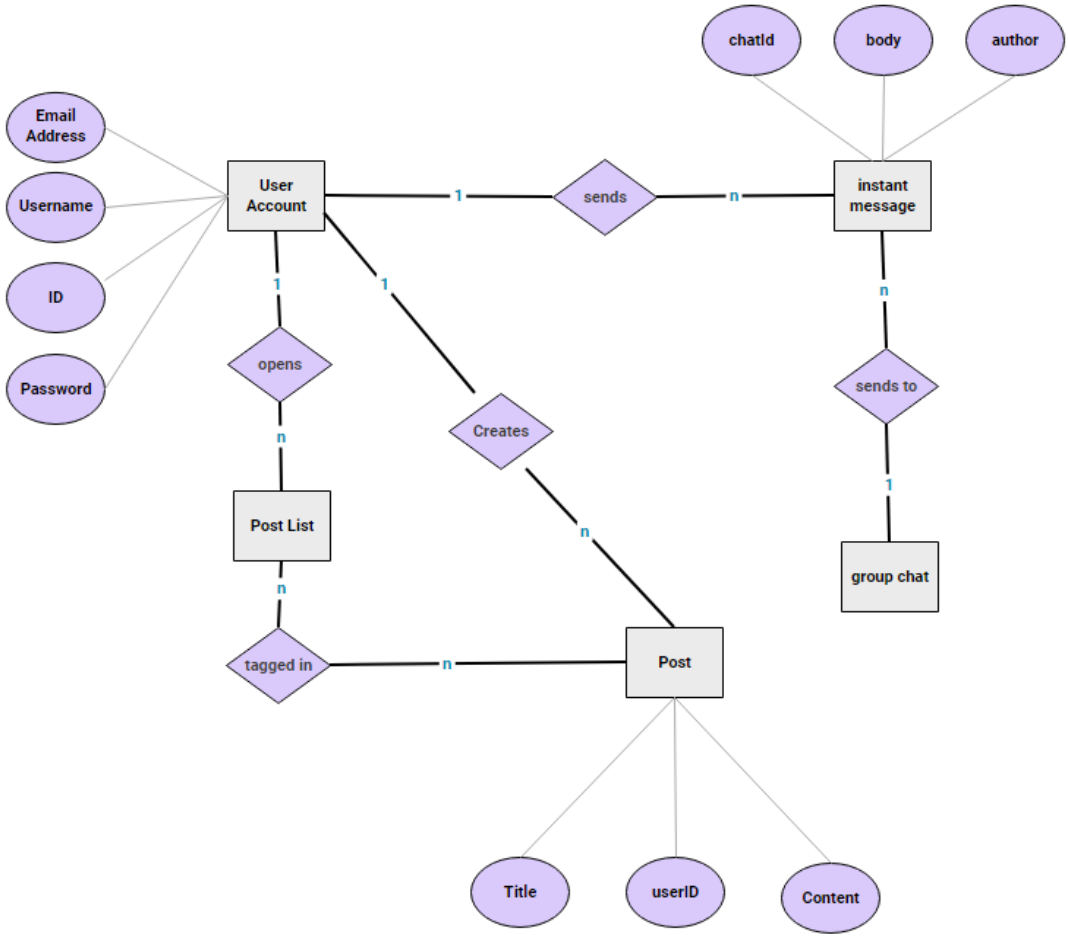
Similarly, the user can send registration data to the registration process in order to register a new account. If the username does not already exist in the User database, the user's registration data will be saved along with all of their entered details. For security reasons, the password is encrypted so that only the user that entered it knows it.

If the user is authenticated, they can create posts on the message board, and access the greater parts of the website. The user's bearer token is authorized and then they may interact with the system. Creating posts is achieved by filling out a form and submitting this.

The API receives this data from the front end and fits it into the database. The Display Posts process gets a list of posts from the Posts database and displays it to the user upon refresh.

The chat API works similarly to the forum API, in that it creates now chats based upon a request from the user with the route being the id of the recipient. Messages are made similar to forum post requests. The difference being that they have a reference to the chat they they are a part of.

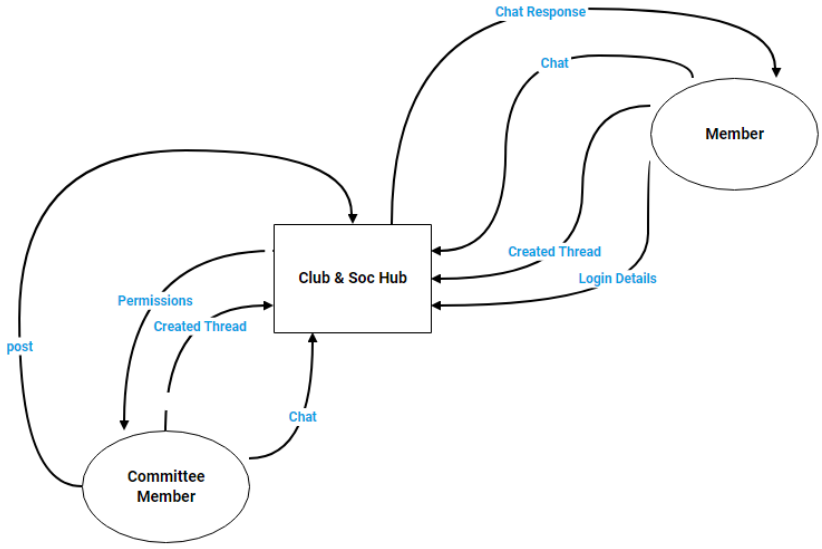
Entity Relationship Diagram



The entity relationship diagram above describes the different entity types that are a part of a system, their attributes and their relationship to other entities.

The User Account entity has 4 attributes: Email address, Username, ID and Password. One user account can open many post lists. One user can create many Posts. This is all handles through the forums route on the API and passed there with the dashboard component in the Angular front-end. One user can send many Instant Messages which is handled with the chats route in the API. The Post entity has three attributes: Title, UserID and Content. Each instant message has three attributes: ChatID, Body and Author. These models are defined in the models section of the express API. The instant message entity has a many to one relationship with the entity Group Chat since many instant messages will be sent to a single group chat and these messages are not accessible by any other group chat entity.

Context Diagram



This diagram shows a high level depiction of the data that is passed in and out of the system. Members can create threads through the forums route, create chat messages through the chats route, and send login details or registration details through the users route. The system will return auth tokens once a user has logged in. It will return chat responses upon request to view them. It will also show forum posts on the dashboard upon initialization. This is where the members may create new threads and delete old ones.

4. Problems and Resolution

During the development of this web application many issues and errors were encountered. One example of an error we encountered was that none of the styling from the Angular framework, Angular Material, was being loaded into our programme. At first, we thought it may have been a semantical error in the code of the component that was being developed at the time. We checked the google chrome developer console to see if any errors had been passed. The error log was empty so we knew that we would need to analyse the code ourselves. In order to do this we installed an extension in our IDE called Angular Essentials. This package instantly identified our problem. The incorrect file path had been specified in our module and components imports file. We were then able to resolve this issue and continue development.

Another issue we encountered was the passing of data between the frontend and backend. When registering a new user we noticed that our new profile was not being saved to our database. In addition we were unable to log in with our newly created account. We added the console.log() command to our typescript file that was sending the data to the backend. We logged the object that was being passed in order to see the structure of the object. After reviewing our backend code that was receiving this data we realised that the wrong parameters were being sent. We were then able to correct this issue and to avoid future problems we specified the expected type.

We encountered a number of small problems, a lot of this was due to us being unfamiliar with the latest version of some of the software, what worked previously in older versions did not work in the version we were using. This was apparent throughout our workload, from back-end to front-end. We have compiled a list of references that we had used for some of the errors we had encountered. Resources such as stack overflow were very useful in resolving issues that we shared with other programmers from around the world.

List of references below:

https://github.com/auth0/angular2-jwt - JWT tokens

https://stackoverflow.com/questions/47236963/no-provider-for-httpclient - Http message passing

https://blog.ninja-squad.com/2017/07/17/http-client-module/ - Learning HttpClient Module

https://stackoverflow.com/questions/37208801/property-map-does-not-exist-on-type-observableresponse - ".pipe(...)" was new to me

https://stackoverflow.com/questions/53824063/angular-7-httpclient-property-success-does-not-exist-on-type-object - found as a result of solution above

https://stackoverflow.com/questions/43453790/property-error-does-not-exist-on-type-promiseany - again, due to above problem

https://angular.io/api/common/http - useful information on http in angular

https://www.npmjs.com/package/ng-flash-messages
- for implemeting flash messages
https://stackoverflow.com/questions/46155/how-to-validate-an-email-address-in-javascript
- to validate an email
https://github.com/auth0/angular2-jwt
- for selective navbar
https://github.com/auth0/angular2-jwt/issues/554
- null token would crash site
https://stackoverflow.com/questions/24474386/pass-mongoose-connection-to-module
- mongoose vs mongodbClient
https://github.com/Automattic/mongoose/issues/6890
- deprecation warnings

5. Installation Guide

This is a 1 to 2 page section which contains a step by step software installation guide. It should include a detailed description of the steps necessary to install the software, a list of all required software, components, versions, hardware, etc.

Got to https://nodejs.org/en/download/ and download the latest version of Node for your system.

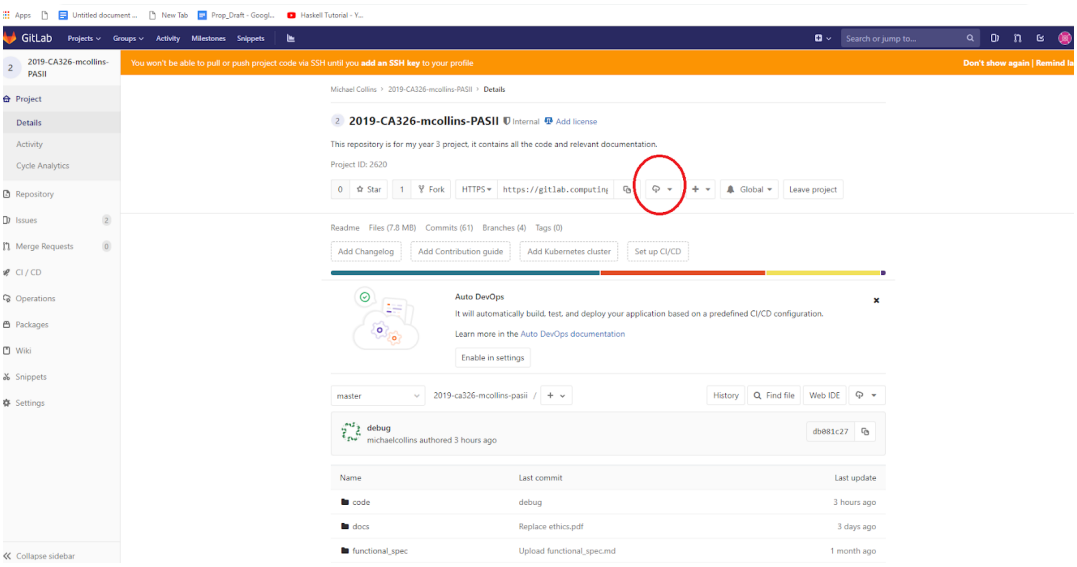
Likewise, go to https://docs.mongodb.com/v3.2/tutorial/install-mongodb-on-windows/ and follow the download, install and run instructions for mongodb

**note: when running the following command

"C:\Program Files\MongoDB\Server\3.2\bin\mongod.exe"

You may need to replace “3.2” with whatever the lastest version of MongoDB is e.g “4.0”

Once these programs are installed, go to
https://gitlab.computing.dcu.ie/collim38/2019-ca326-mcollins-pasii
and download by clicking the button circled in the



Extract the contents of the downloaded zip

Using Command Prompt or any other terminal, navigate to the directory in which you extracted the zip. Navigate to the directory /code. Once you are in that directory run the command “npm install”.

After the install is finished, navigate to the subdirectory ./angular-src. Now run the same command “npm install”

Once the install process is complete, run the command “ng build”.

In your preferred browser, type the following into the search bar and enter “http:localhost:3000”

At this point you will be presented with the web application.