Contents

[Introduction 2](#_Toc500786628)

[Initial Proposal 2](#_Toc500786629)

[Parts I wrote: 8](#_Toc500786630)

[API 8](#_Toc500786631)

[Thread List 8](#_Toc500786632)

[Add Thread 9](#_Toc500786633)

[View Thread 10](#_Toc500786634)

[Confirm Modal / Post Modal 12](#_Toc500786635)

[Authentication: 13](#_Toc500786636)

[Auth Guard. 13](#_Toc500786637)

[Login 13](#_Toc500786638)

[Event Binding / Extra Features 13](#_Toc500786639)

[Authentication: 13](#_Toc500786640)

[Buttons 13](#_Toc500786641)

[Post 13](#_Toc500786642)

[Error 14](#_Toc500786643)

[Toast 14](#_Toc500786644)

[Hosting 14](#_Toc500786645)

[Reflection 14](#_Toc500786646)

# Introduction

Student ID: S00165159

User details needed are as follows.

[mcgowan.b@gmail.com](mailto:mcgowan.b@gmail.com) and password is Password1

url <https://secure-cove-75663.herokuapp.com/dashboard>

Angular Repository: <https://github.com/mcgowanb/y3-web-proj>

API Repository: https://github.com/mcgowanb/y3-web-proj-api

If you intend to run this locally, you will need to start the server in the api module using npm run dynamic. Ensure the cors options are set to include localhost connections, or else it will refuse connection requests. The app must make sure in the breadcrumb service, dataService and emailService that they are pointing to the localhost endpoint too.

There are too many commits to snip and screenshot into this doc, so I’ve attached an excel spreadsheet with all the commits filtered to my username. There are two tabs, one for the app and the other for the api.

These repo’s are private since there are auth keys contained within (and also to stop others copying code) so you must be logged in to github with the SBanksITSligo user in order to view / checkout

There are no separate authentication levels on this project, so all users are treated the same. You are also free to register for the app yourself via the register page. There is no two-step authentication in this app, so once registration completes you are immediately logged in.

Note, we decided in the end not to bother with the twitter and facebook plugins as there wasn’t enough time to add them in.

All interaction with the API is managed via an injectable service called dataService

# Initial Proposal

**Web Programming 3 Project Proposal October 17**

**Group12**

**Project Description**

**Idea & Users**

The idea behind our website is that it would be a forum for third year ITSligo Computing students who wanted to discuss work from any of their different classes.

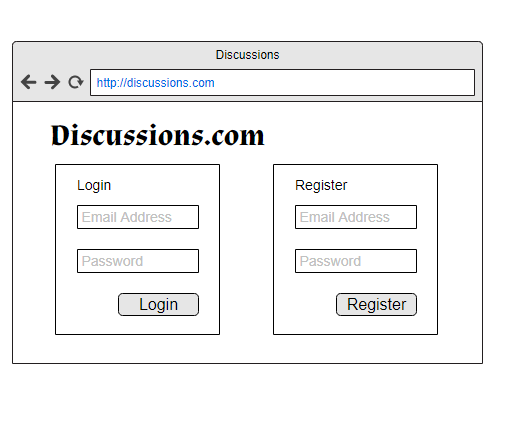
**Features**

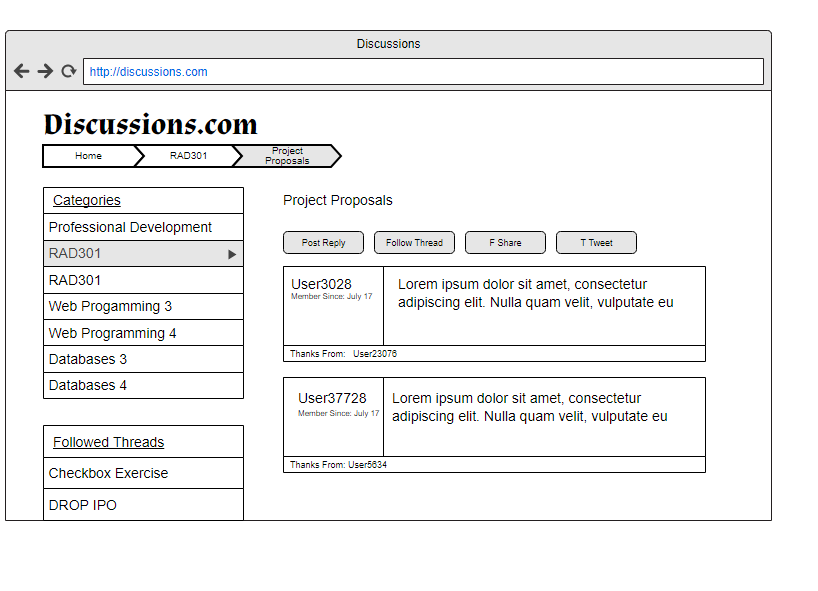
Users would be able to start a new thread, reply to existing comments and thank posts. There would be a Facebook and twitter API where they could share the discussions. You can favourite threads of interest and they will be added to a list where you will be notified by email of any new comments.

**External Data**

Threads, posts, user information all stored in an external database.

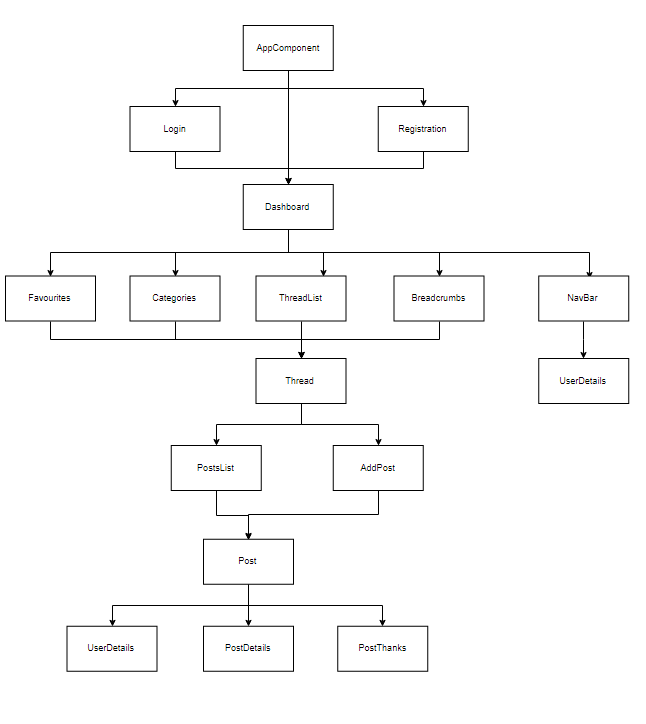
**User Interface Design**

****

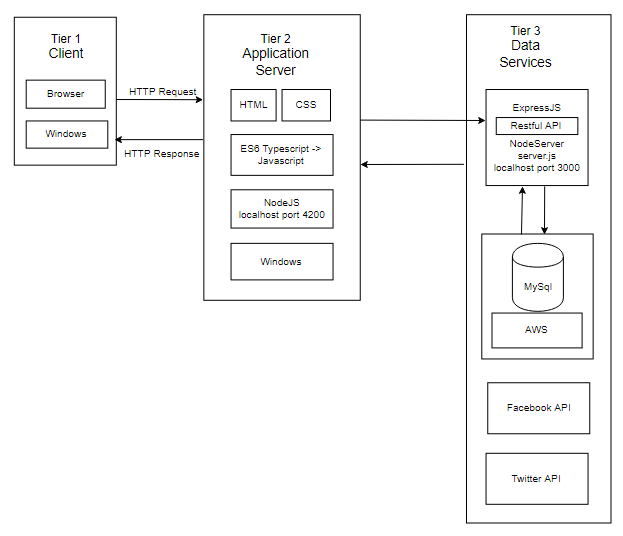
****

****

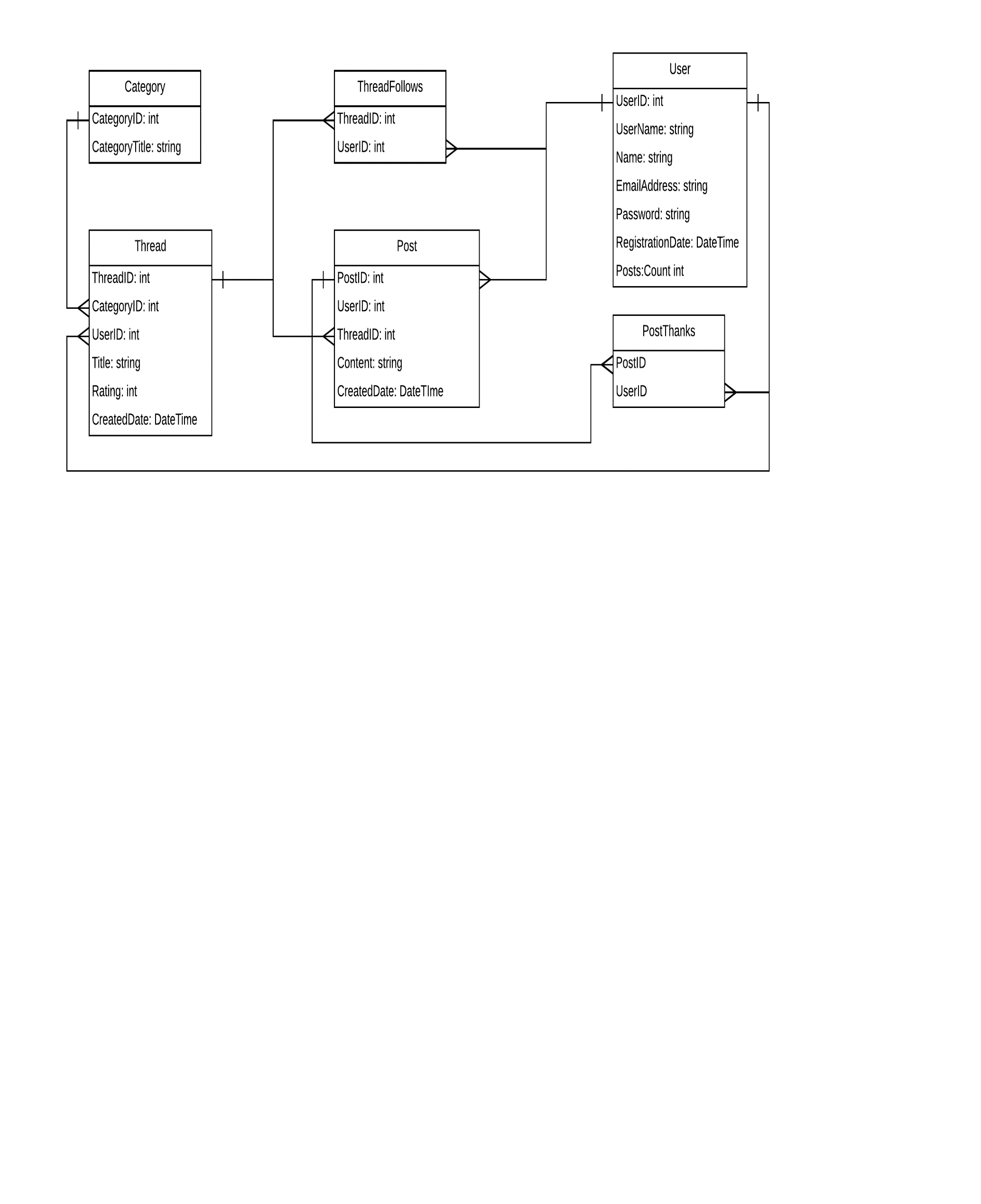
**Component Tree Diagram**

****

**Architecture Diagram**

****

**Database Diagram**

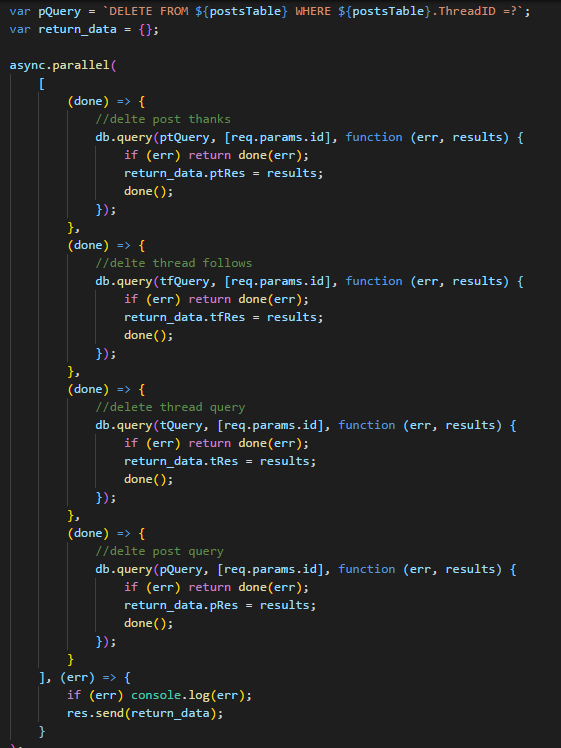
****

# Parts I wrote:

I created the following parts of the application

## API

I created 90% of the api project, including original scaffolding, database creation, end-point exposure and the mySQL queries used to generate the data for the application. There are chained queries asynchronously running together, from which a single result object is returned to the app. For example, the deleteThreadCascading function runs multiple queries at the same time, namely delete posts, thread, postthanks and threadfollows when a thread is deleted. (see image). Except for authentication, we used no other api’s for interaction bar our own custom service to interact with our database. Given the blog nature of the project there were a considerable number of endpoints to be created, and some of them would need to handle GET, POST and DELETE requests

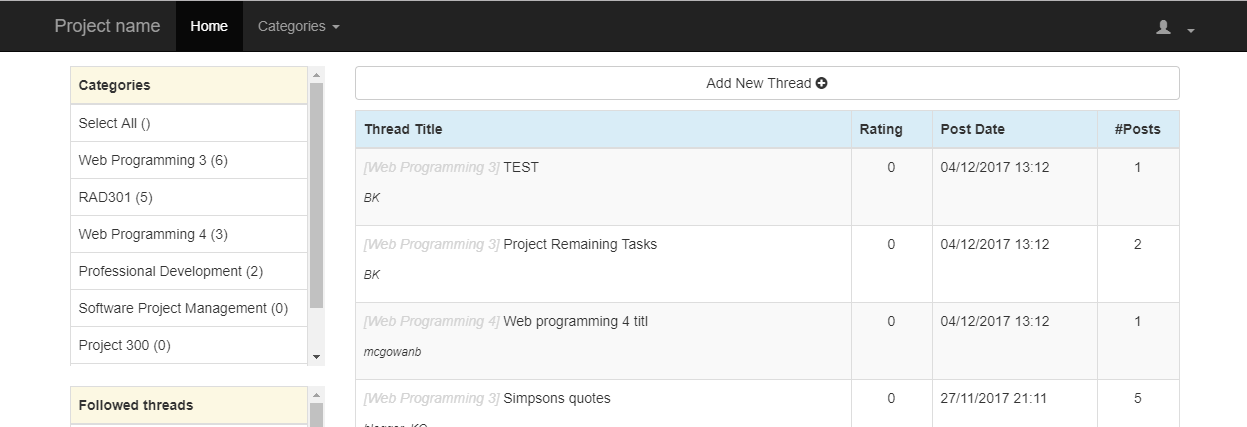


## Thread List

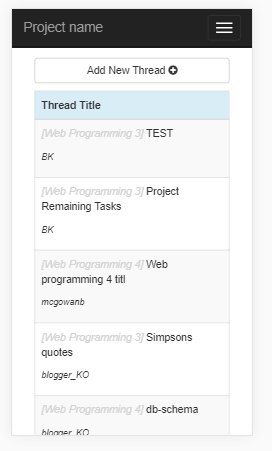
On the dashboard, the list of threads that’s displayed for users to click on to view an additional thread. I also included animations in angular to ease the transition in on the page. A spinner is displayed on the page until the http request to load the threads has completed to give the user an indication that the application is busy.

The page is 100% mobile friendly and reacts to different screen sizes

FullScreen



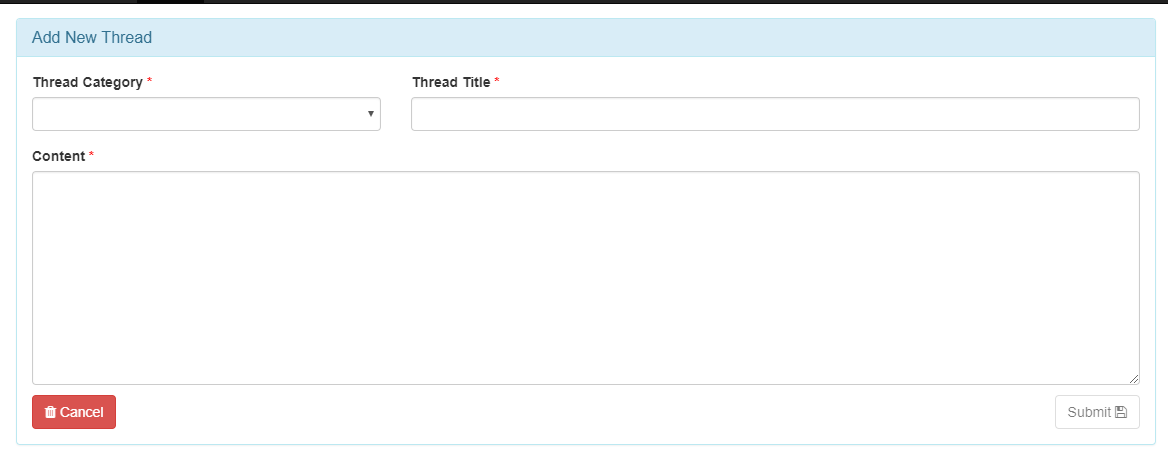
Mobile



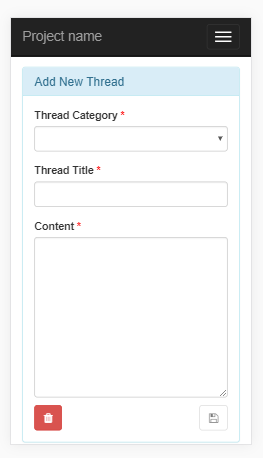
## Add Thread

Add thread page is a simple component with a form that allows a user to create a new thread. The API takes this data, creates a thread record, then a post record with the thread id to create the relationship between the two. The user’s post count is also updated

FullScreen



Mobile



## View Thread

The thread component renders the thread with whatever posts are associated to the thread. There is the following functionality on the thread page:

If you are the thread owner / creator you can delete the thread via modal confirmation

You can reply & add a post to the thread via modal popup box

You can follow a thread. (I did not complete this functionality)

Post Component

The Post component contains the body of the individual posts for the thread, and renders sub components such as the buttons and the user info.

Post Buttons

These buttons are the actions that a user can perform. If a user has not thanked a post, they can do so and it will add to the list of thanks. The thanks button then changes and becomes a remove thanks button which does the opposite.

If you are the owner of the post, you can edit or delete the post via buttons if you wish

Post Thanks

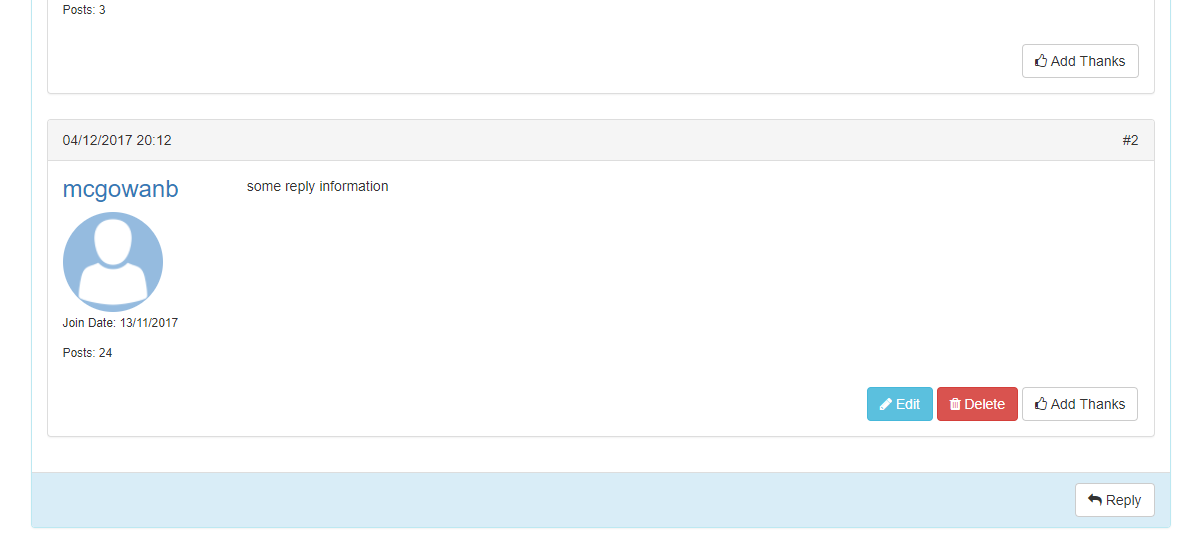
This shows the usernames of the people that have thanked the post

Post User Details

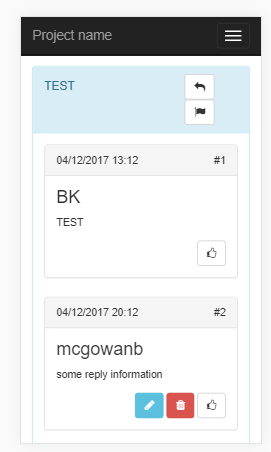
This displays user information for the person who created the post.

Note: On a mobile device, text in buttons is not displayed – only icons for actions. This takes up less real-estate on screen when its limited.

FullScreen



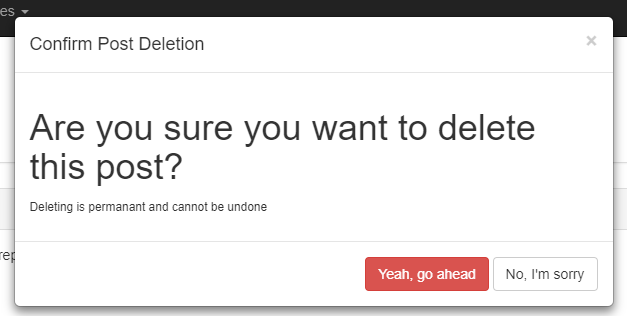
Mobile



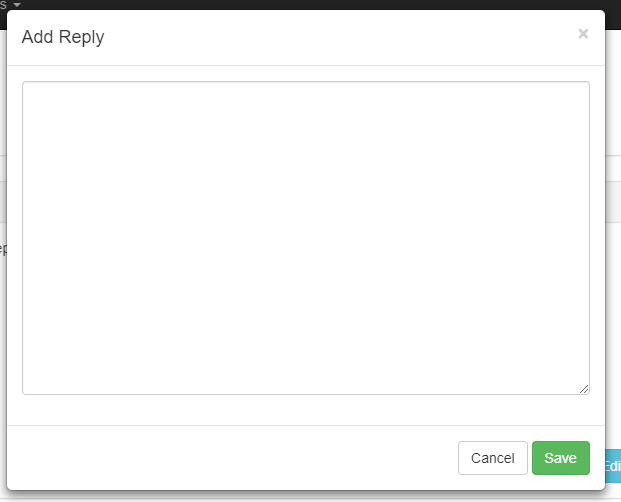
## Confirm Modal / Post Modal

I used components as modals for confirmation and post actions, since the actions required were small enough not to warrant an entire page. There is a generic modal component which, depending on the context, it will display different messages and different action buttons. This was done by extending the DialogComponent class from ng2-bootstrap-modal plugin

Confirmation:



Post



## Authentication:

I implemented the firebase authentication service for the app. When a user registers, they are created as an auth account in firebase, and returns the UID. We take that UID and create a user record in our own database, and so now we have link between our database and the firebase authentication.

## Auth Guard.

The auth guard guards against all un-authenticated access to all urls except for the login and register urls.

## Login

Created the login page and added form validation for valid email address and password.

# Event Binding / Extra Features

I implemented event binding in the following areas:

## Authentication:

I’ve subscribed to the authentication object, and if a users authentication state changes, they are redirected to the login page automatically.

## Buttons

All buttons in the thread component emit events up the stack when they are clicked. They emit custom post action objects, with action events bound to defined enums, in order to make it generic, extensible and more maintainable in the future.

## Post

In the same way, the Post component emits events up to the parent component for actions that are taken from there. Again, there are custom event objects implemented with actions bound to an enum type.

## Error

There’s also an error emitter if there’s a http error on any of the thread http requests, an error is displayed to the user in an alert modal.

## Toast

Implemented bootstrap toasts to display notifications to users.

## Hosting

The app and the api are both hosted on heroku

# Reflection

On balance, the group project worked reasonably well. I was happy with the group that I was in and the collaboration that took place amongst us all, and there were many learning experiences as a group. The incomplete authentication tutorial we were given posed significant issues when trying to implement it. However, once I got a good understanding of how it actually works it was straight forward to implement it as we needed it.