5CCS2SEG - Major Group Project

Fitness and Nutrition Aggregator

Report

Client
Danielle Dodoo

Team Codebrew
Selim Alastra - k1897684@kcl.ac.uk
Khalid Alsheeb - k1897497@kcl.ac.uk
Ioana Bottez - k19008196@kcl.ac.uk
Ioana-Alexandra Ghinea - k19006468@kcl.ac.uk
Joshua Harris - k19008090@kcl.ac.uk
Bianca Opariuc - k19002013@kcl.ac.uk
Sergiu-Stefan Tomescu - k19027239@kcl.ac.uk
Tushita Yadav - k19010851@kcl.ac.uk

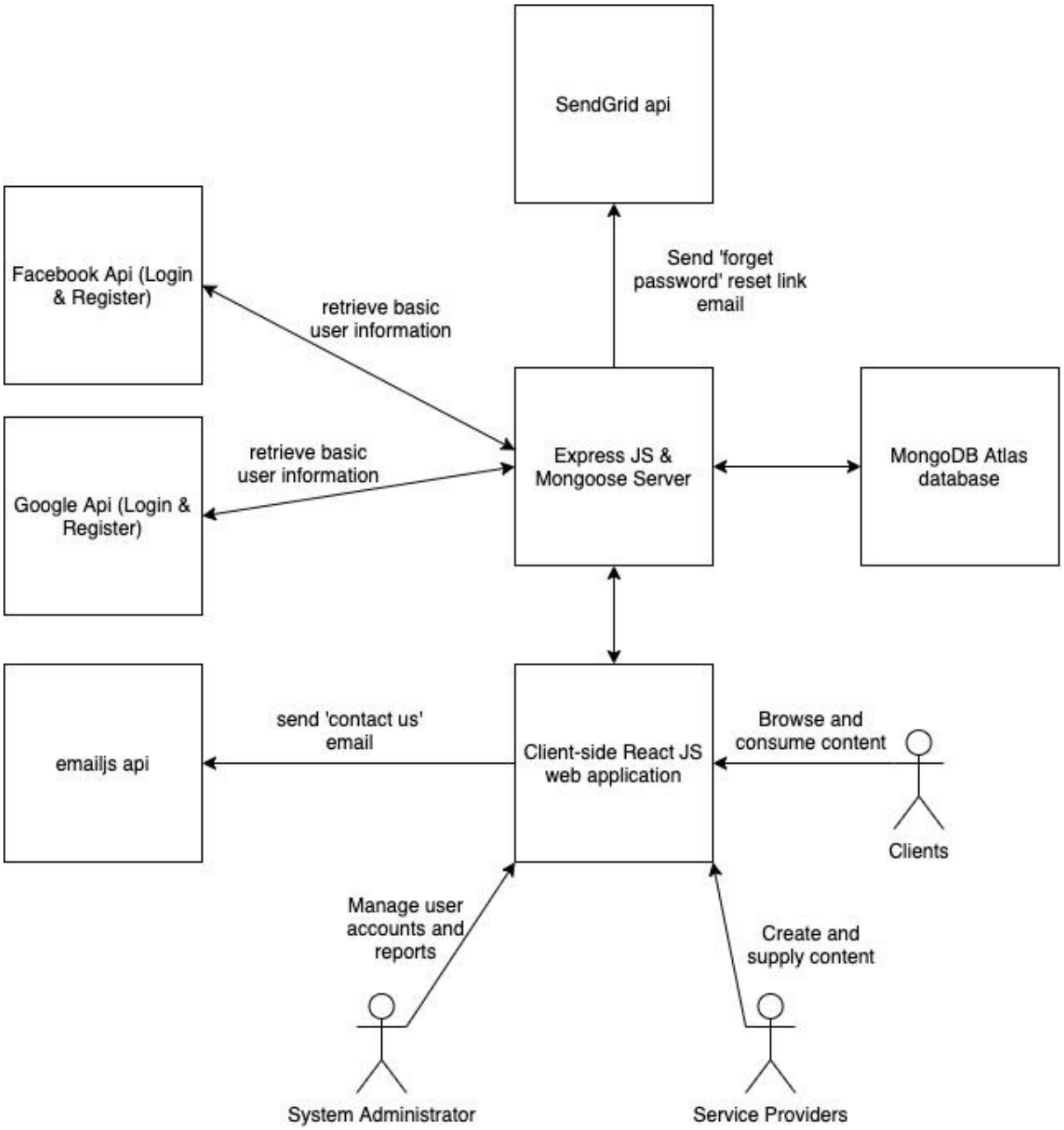
Client’s Objectives

The client’s vision and goals for this platform were of paramount importance to the team and the team tried to use their strengths to best approach development. The platform developed aims to bring social media influencers and online personalities offering holistic health, nutrition and fitness services together, onto one platform. The platform enables service providers to not only create bundles of custom content but also allows them to pull their services from other platforms such as YouTube videos, videos hosted on platforms such as Vimeo, Twitch, FaceBook Posts, Audio links as well as embedded links from their personal websites and blogs. This aggregation of content makes it easier for clients to search and find influencers offering the services they are looking for. It aims to make the matching between service providers and potential clients a smoother process. In the future the platform would provide budding influencers a proper marketplace for their services. The team believes that the accumulation of service providers and clients on one platform is the primary objective of this project.

Index

System Architecture

Displayed below is a basic diagram depicting an abstraction of the system we have developed for the client. As you can see, the main system is split into a client side application and a server side application, which both make use of proprietary services like the Facebook api. There are three types of system users, which are all depicted in the diagram.
The Facebook and Google apis are used to gain access to basic profile information in order to create an account without filling out the register form. The ‘emailjs’ api sends client messages to the application owner. Furthermore, the SendGrid api sends reset password emails to clients and service providers in order to reset their passwords.



Technology Stack

For the project, we decided to use the ‘MERN’ stack which consists of MongoDB, Express JS, React JS and Node JS.

For the frontend components, we used the React JS framework to design and construct the user interface along with the following libraries: react-modal; formik; react-iframe; react-google-login; react-facebook-login and reactjs-popup. To supply the data to the frontend components, we used Redux to dispatch and fetch data from the main store. Furthermore, we used Axios to make api requests to the backend. For styling the frontend components, we used the Material-UI, Bootstrap and FontAwesome libraries.

For the backend, we used Mongoose to setup the connection to the MongoDB database, to create the database schemas and to fetch the documents from the database. Also, we used SendGrid to send ‘reset password’ emails, bcrypt to encrypt and decrypt passwords, express to setup and manage the server and requests and ‘jsonwebtoken’ to supply web tokens to authenticate users. Furthermore, we decided to use MongoDB Atlas for both the production and test database.

For testing, we mainly used Chai, Mocha and Supertest for testing the server side of the web application. Furthermore, we used Cypress to test the user interface and Chai, Mocha for the reducers on the frontend of the web application.

For deploying the web application, we used the Heroku platform, which hosts both the backend and frontend of the web application.

Finally, we use Node Package Manager to install, manage and remove JavaScript packages for both the backend and frontend of the web application.

Quality Assurance

The team employed automated tests for different parts of the application. It was of vital importance to the team to test the components of the backend and frontend adequately. The backend uses Chai, Mocha and SuperTest to test the models of the database and the routes on the server side. These are automated unit tests that test each component in the application. For the frontend, integration tests have been carried out using Cypress which builds upon the Mocha testing framework. These are automated integration tests used for end-to-end testing of the frontend user interface. The tests for the reducers on the client side have been written in Chai and Mocha as they can then be tested as individual units as opposed to Cypress' usual integration tests.

Server	Client
In the server/tests directory: **Chai & Mocha** models admin.test.js basicUser.test.js buckets.test.js goal.test.js postMessage.test.js professionalUser.test.js report.test.js service.test.js	In the client/src/tests/reducers directory:- **Chai & Mocha** reducers basicUsers.test.js buckets.test.js goals.test.js posts.test.js professionals.test.js reports.test.js services.test.js
In the server/tests directory:- **Chai, Mocha & Supertest** routes admin.test.js basicUsers.test.js buckets.test.js goals.test.js posts.test.js professionalUsers.test.js reports.test.js services.test.js	In the client/cypress/integration directory:- **Cypress** adminlogin.spec.js basicuserregister.spec.js clientDashboard.spec.js prodessionalBundles.spec.js professionalDashboard.spec.js professionaluserregister.spec.js professionalProfile.spec.js In the client/cypress/component directory:- **Cypress** landingpage.spec.js

(Screenshots of Runs can be found in the Appendix)

Note: While team members ran the cypress tests on their own personal machines, the team discovered that some tests often fail for reasons such as poor network connection. The Secure Routing implemented, redirected to the Landing Page causing the Cypress user Interface Tests to fail.

The client side contains adequate integration tests that cover the breadth of the user interface however the team struggled to write unit tests for independent units within the user interface. The landingpage.spec.js test written is the only working component test. The team speculates that this limitation is due to the secure routing implemented across the application, making integration tests more suitable for the frontend. While frameworks exist for writing unit testing the user interface the team was unable to find a workaround for writing them within the deadline.

Project Management

The team discussed project management early on in the project and set guidelines which would help the team work in synchronisation. First, the following communication channels were discussed and agreed upon.

Communication Channels:

- Microsoft Teams
- WhatsApp Group
- Trello Board

The team decided that it would be most effective if all members met at least twice a week to discuss progress of existing tasks and the new tasks to be allocated. The following days and times were discussed and agreed upon.

Weekly Meetings:

- Tuesday: 11:00am - 12:00pm
- Friday: 2:00pm - 3:00pm

Pair Programming Sessions and Bug Fixing Sessions would be held separately and would usually involve those developing the particular components.

Prior to the start of the project the team established guidelines for Conflict Resolution which would help the project progress smoothly and in harmony.

Conflict Resolution:

- Acknowledge the conflict
- Discuss the impact
- Agree to a cooperative process

- Agree to communicate on all occasions

The team decided to use the MERN Stack to develop the application as it seemed to be the most appropriate however, none of the team members had prior experience working with the stack. Thus it was difficult for the team to estimate how long a particular development phase/sprint would be. Hence the team decided to adopt the Waterfall Model Software Process.

Waterfall Model Software Process

Requirements Elicitation & Analysis
The team met with the client on two occasions and discussed the client's visions, goals and key features to be implemented. Following these meetings the team consulted the minutes that were documented and brainstormed further on the kind of prototype that was to be developed, what features could be added and what would make the platform unique. All documentation was maintained in a shared Google Drive Folder so that all team members had access to it. The deliverables initially seemed expansive as the team wanted to implement all the client's deliverables and also wanted to develop features and ideas that originated within the team. This was a very pressing concern during the initial phases of the project due to considerable time limitations. The academic advisory meetings that followed were very helpful as they helped the team to focus on the key deliverables required to build a coherent, working prototype.

System & Software Design
In the weeks following Requirements Elicitation the team divided tasks based upon the key deliverables that had been agreed upon for the initial stage of the project. Since all team members were beginners and had no proper experience working with ReactJS, Express, NodeJS and MongoDB the team decided to allocate considerably larger tasks to two people. For example, creating Login/Register Forms (including google, FaceBook logins) were allocated to a pair of team members. During this stage the team members split up the basic tasks and began searching for suitable resources and tutorials online(included in the Appendix) that could be used to set up the basic structure of the project. As team members started implementing their allocated tasks, the meetings during the week helped track everyone's progress and understanding of the new web stack.

Implementation
The team used the Trello board and Github repository to keep track of everyone's progress. Prior to development the team decided to create separate branches for each task and a development branch that would be updated after team members reviewed the completed branches. Once team members completed their tasks the branch was modified to fit the agreed upon file structure/system design and was merged with development. Progress was discussed at the weekly meetings and bug fixes were resolved within the team. Once the team

had completed the basic deliverables, the code was refactored to fit into the agreed upon software design and new tasks to build upon existing features were allocated. As the project reached the later weeks, the team set internal deadlines to complete the functionality of the project and obtain feedback from the client. Following this the necessary changes and last minute functionality was added before the team moved onto the design and testing phases.

Design
At this stage of the project, all the components were working together however each of them had their own CSS and styling files. As the team had focused on functionality thus far, designing all the components was a considerably large task. However, due to time constraints it was of vital importance to complete not only design but also testing for all the components. The team discussed and agreed that splitting up the team into two would be the best approach to tackling Design and Testing.

Testing
Testing for backend routes and models was carried out during the design phase as there were not many changes to be made to the existing functionality. Subsequently on completion of frontend design, tests were written for the frontend routes and user interface.

Team Organisation

Team Member	***Development***	***Design***	***Testing***
Selim Alastra	Landing Page, Home Page, Database Schema, Contact Us Page	Bundles, HomePage, Edit Goals, Landing Page, Modals	
Khalid Alsheeb	Admin Components, Private Routing, Reports, Backend controllers		Backend Routes, Bug Fixing, Frontend Routes, Code Refactoring, front-end reducers
Ioana Bottez	Register, Login for Users, Forget/Reset Password, Buckets	Profiles, Bundles, Landing Page, Reset Password Page, Admin	

Ioana-Alexandra Ghinea	Register, Login for Users, Forget/Reset Password, Buckets	Buckets, Posts, Dashboards, PopUpPost, MyPosts	
Joshua Harris	Profiles, Bundles, Edit Details		Backend Models, Backend Routes
Bianca Opariuc	Dashboards, Posts, Search		Frontend Integration, Frontend UI
Sergiu-Stefan Tomescu	Quiz for Users, Search, Navbar	Quiz, Navbar	
Tushita Yadav	Dashboards, Posts, Search		Frontend Integration, Frontend UI

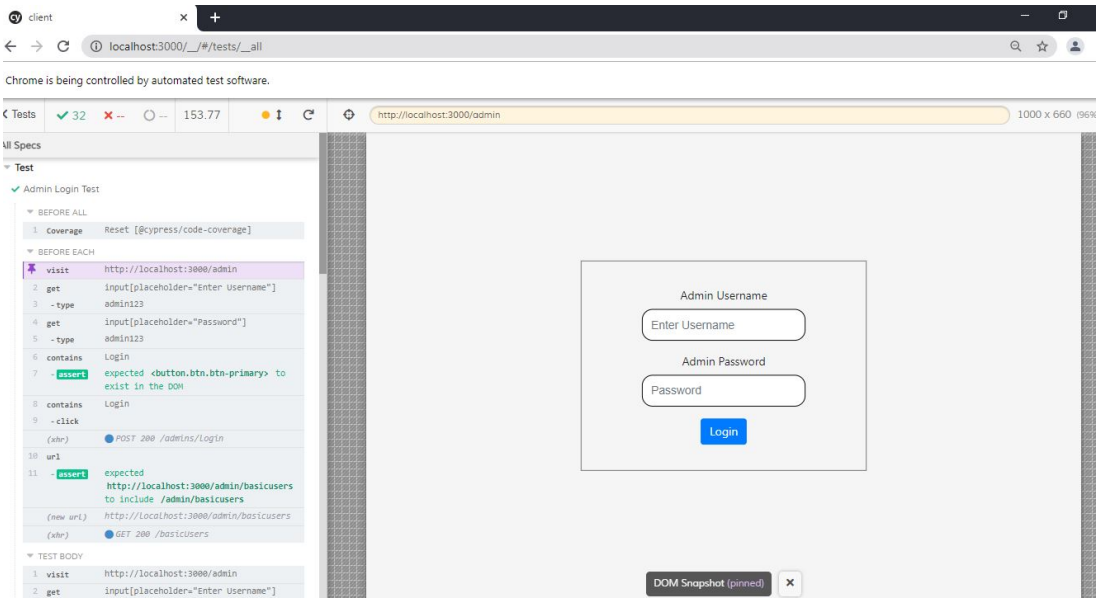
Team Representative: **Joshua Harris**
Meeting Organisation & Minutes: **Bianca Opariuc**
Report: **Joshua Harris, Tushita Yadav**
Developer's Setup: **Joshua Harris, Tushita Yadav, Bianca Opariuc**
User's Manual: **Ioana Bottez, Ioana-Alexandra Ghinea**
Screencast Demonstration: **Joshua Harris**
Note: In the screencast (timestamp - 15:58) the centering of profiles has been modified and fixed.
Backup YouTube Link to Screencast: https://youtu.be/Qely5MiD8i4

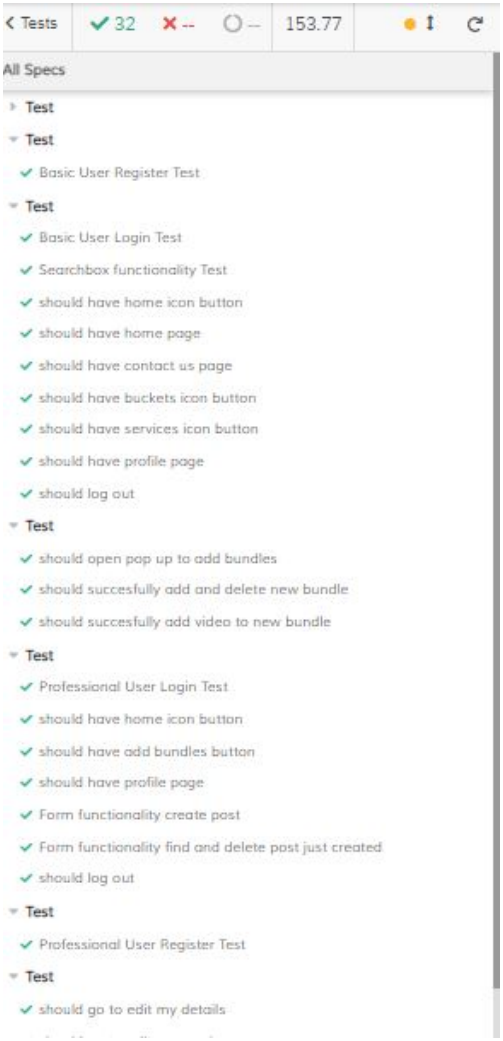
Appendix

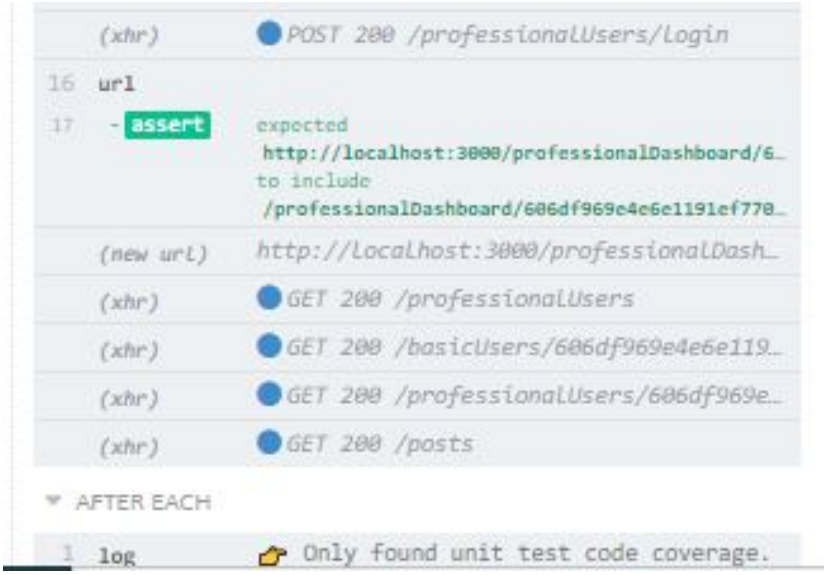
Test Runs:

Frontend

Cypress User Interface: ***npx cypress open***


client
localhost:3000/__/#/tests/__all
Chrome is being controlled by automated test software.
Tests
32
153.77
http://localhost:3000/admin
1000 x 660
All Specs
Test
Admin Login Test
BEFORE ALL
BEFORE EACH
TEST BODY
Admin Username
Enter Username
Admin Password
Password
Login
DOM Snapshot (pinned)


Tests
32
153.77
All Specs
Test
Basic User Register Test
Basic User Login Test
Searchbox functionality Test
should have home icon button
should have home page
should have contact us page
should have buckets icon button
should have services icon button
should have profile page
should log out
should open pop up to add bundles
should succesfully add and delete new bundle
should succesfully add video to new bundle
Professional User Login Test
should have home icon button
should have add bundles button
should have profile page
Form functionality create post
Form functionality find and delete post just created
should log out
Professional User Register Test
should go to edit my details


(xhr) POST 200 /professionalUsers/Login
16 url
17 - assert expected http://localhost:3000/professionalDashboard/6...
to include /professionalDashboard/606df969e4e6e1191ef770...
(new url) http://Localhost:3000/professionalDash...
(xhr) GET 200 /professionalUsers
(xhr) GET 200 /basicUsers/606df969e4e6e119...
(xhr) GET 200 /professionalUsers/606df969e...
(xhr) GET 200 /posts
AFTER EACH
1 log Only found unit test code coverage.

Command Line: ***npx cypress run***

```
(Run Finished)

     Spec                                              Tests  Passing  Failing  Pending  Skipped
┌────────────────────────────────────────────────────────────────────────────────────────────────┐
│ ✓  adminlogin.spec.js                     00:18        9        9        -        -        - │
├────────────────────────────────────────────────────────────────────────────────────────────────┤
│ ✓  basicuserregister.spec.js              00:04        1        1        -        -        - │
├────────────────────────────────────────────────────────────────────────────────────────────────┤
│ ✓  clientDashboard.spec.js                00:20        9        9        -        -        - │
├────────────────────────────────────────────────────────────────────────────────────────────────┤
│ ✓  professionalBundles.spec.js            00:12        3        3        -        -        - │
├────────────────────────────────────────────────────────────────────────────────────────────────┤
│ ✓  professionalDashboard.spec.js          00:17        7        7        -        -        - │
├────────────────────────────────────────────────────────────────────────────────────────────────┤
│ ✓  professionaluserregister.spec.js       00:04        1        1        -        -        - │
├────────────────────────────────────────────────────────────────────────────────────────────────┤
│ ✓  professionaProfile.spec.js             00:05        2        2        -        -        - │
├────────────────────────────────────────────────────────────────────────────────────────────────┤
│ ✓  landingpage.spec.js                    496ms        1        1        -        -        - │
└────────────────────────────────────────────────────────────────────────────────────────────────┘
  ✓  All specs passed!                      01:23       33       33        -        -        -
```

Backend

Models:

```
[nodemon] 2.0.7
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `mocha */**/**.test.js --require ./tests/globalHooks.js --timeout 20000 --exit`
(node:11156) ExperimentalWarning: The ESM module loader is experimental.
(node:11156) DeprecationWarning: collection.ensureIndex is deprecated. Use createIndexes instead.
Server running on port: 5000


  Testing Admin model
    ✓ is valid when all of the fields are valid
    ✓ is invalid when no username is supplied
    ✓ is invalid when no password is supplied
    ✓ is invalid if the username is less than 3 characters
    ✓ trims the username to remove whitespaces

  Testing BasicUser model
    ✓ is valid when all of the mandatory fields are provided
    ✓ is invalid as the madatory fields are null
    ✓ is invalid as the username does not meet the minimum length
    ✓ is invalid as the username is not unique
    ✓ is invalid as the email is not unique
    ✓ is invalid as the gender is not one of the enumerated values
    ✓ has the correct default value for resetPasswordLink
    ✓ trims the username

  Testing Buckets model
    ✓ is valid as all fields are valid
    ✓ is invalid as the title is not supplied
    ✓ is invalid as the userId is not supplied

  Testing Goal Model
    ✓ is valid as all fields are correct
    ✓ is invalid if the userID is not supplied
```

```
Testing Post model
  ✓ is valid as all mandatory fields are supplied
  ✓ is invalid as not all mandatory fields are supplied
  ✓ has a default empty array for likes

Testing ProfessionalUser model
  ✓ is valid, as all of the mandatory fields are supplied
  ✓ is valid, as all fields are supplied
  ✓ is invalid, as madatory fields are not supplied
  ✓ is invalid when the gender is lowercase
  ✓ is invalid if gender is not one of the enumerated values
  ✓ is invalid if the username is shorter than 3 characters

Testing Report model
  ✓ is valid as all fields are valid
  ✓ is invalid as the mandatory fields are not supplied

Testing Service model
  ✓ should be valid, as all fields are supplied correctly
  ✓ should be invalid, as userID is left out
  ✓ should be invalid, as title is left out
  ✓ should be invalid, as description is left out
  ✓ should be invalid, as price is left out
  ✓ should be valid, as price is converted to a string
  ✓ should be valid, as description and title are converted to strings
```

Routes:

```
admins routes
  post /admins/login
    ✓ should login an admin (128ms)
    ✓ should not login an admin, as the username is wrong (91ms)
    ✓ should not login an admin, as the password is wrong (90ms)

Basic users routes
  post /basicUsers
    ✓ should make a new basic user (102ms)
    ✓ should return a 400 if the basic user could not be added
  get /basicUsers/:id
    ✓ should retrieve a specific basic user (94ms)
    ✓ should return a 404 status code as an invalid id is supplied
  get /basicUsers
    ✓ should get all basic users and return a 200 status code (95ms)
  patch /basicUsers/update/:id
    ✓ should update the basic user (99ms)
    ✓ should return a 404 status code as the id does not link to a basic user
  delete /basicUsers/:id
    ✓ should delete the basic user associated with the uri (100ms)
Error: Cannot delete this basicUserCastError: Cast to ObjectId failed for value "1234" at path "_id" for model "BasicUser"
    ✓ should return 404 status code as the uri is not associated with a basic user
  post /basicUsers/register
    ✓ should register a new basic user and returns a web token and user (460ms)
    ✓ should return error as email is already in use (105ms)
  post /basicUsers/login
    ✓ should login a basic user and returns a web token and user (97ms)
    ✓ should return error as no user with this email exist (89ms)
    ✓ should return error as the password does not match the email (94ms)
    ✓ should return error as the user is banned (92ms)
  put /basicUsers/forgotpassword
    - should return a sent message on successful delivery on reset email
    - should return an error message as no user with the email address supplied
```

```
  buckets routes
started
    post /buckets
      ✓ should make a new bucket (199ms)
      ✓ should return a 400 if the bucket could not be added
    get /buckets
      ✓ should get all buckets and return a 200 status code (95ms)
    get /buckets/:id
      ✓ should get the buckets object that have userId equal to basicUserId (91ms)
      ✓ should return a 404 status code as an invalid id is supplied
    patch /buckets/:id
      ✓ should update the bucket (101ms)
      ✓ should return a 404 status code as the id does not link to a bucket
    delete /buckets
      ✓ should delete the bucket associated with the uri (99ms)
      ✓ should return 400 status code as the uri is not associated with a bucket

  goals routes
    post /goals
      ✓ should make a new goal (102ms)
      ✓ should return a 400 if the goal could not be added
    get /goals
      ✓ should get all goals and return a 200 status code (94ms)
    get /goals/:id
      ✓ should get the goals object that have UserId equal to basicUserId (106ms)
      ✓ should return a 400 status code as an invalid id is supplied
    patch /goals/:id
      ✓ should update the goal (201ms)
      ✓ should return a 404 status code as the id does not link to a goal
    delete /goals/:id
      ✓ should delete the goal associated with the uri (96ms)
      ✓ should return 400 status code as the uri is not associated with a goal
```

```
sts routes
post /posts
  ✓ should make a new post (97ms)
  ✓ should return a 400 if the post could not be added
get /posts/:id
  ✓ should retrieve a specific post (91ms)
  ✓ should return a 404 status code as an invalid id is supplied
get /posts
  ✓ should get all posts and return a 200 status code (90ms)
get /posts/:id/bucket
  ✓ should get all posts froma bucket, and return a 200 status code (184ms)
patch /posts/:id
  ✓ should update the post (102ms)
  ✓ should return a 404 status code as the id does not link to a post
patch /posts/:userId/likePost
  ✓ should add a user id to the like array of the post (198ms)
  ✓ should return a 404 status code as the id does not link to a post
delete /posts/:id
  ✓ should delete the post associated with the uri (104ms)
  ✓ should return 404 status code as the uri is not associated with a post

fessional user routes
post /professionalUsers
  ✓ should make a new professional user (103ms)
  ✓ should return a 400 if the professional user could not be added
get /professionalUsers
  ✓ should get all professional users and return a 200 status code (101ms)
get /professionalUsers/:id
  ✓ should get the professional user object that have UserId equal to professionalUserId (100ms)
  ✓ should return a 404 status code as an invalid id is supplied
patch /professionalUsers/update/:id
  ✓ should update the professional user (107ms)
  ✓ should return a 404 status code as the id does not link to a professional
delete /professionalUsers/:id
  ✓ should delete the professional associated with the uri
  ✓ should return 404 status code as the uri is not associated with a professional
post /professionalUsers/register
  ✓ should register a new professional and returns a web token and user (455ms)
  ✓ should return error as username is already in use (95ms)
post /login
  ✓ should return a token and user object on successful login (93ms)
  ✓ should return an error if no user exists with the email provided (94ms)
  ✓ should return an error if the email and password do not match (89ms)
  ✓ should return an error if the professional is banned (88ms)
put /forgotpassword
  - should return a sent message on successful delivery on reset email
  - should return an error message as no user with the email address supplied
```

```
  report routes
    post /reports
      ✓ should make a new report (100ms)
      ✓ should return a 409 if the report could not be added
    get /reports/:id
      ✓ should retrieve a specific report (95ms)
      ✓ should return a 404 status code as an invalid id is supplied
    get /reports
      ✓ should get all reports and return a 200 status code (87ms)
    delete /reports/:id
      ✓ should delete the report associated with the uri (103ms)
      ✓ should return 404 status code as the uri is not associated with a report

  services routes
    post /services
      ✓ should make a new service (119ms)
      ✓ should return a 400 if the service could not be added
    get /services
      ✓ should get all services and return a 200 status code (97ms)
    patch /services/update/:id
      ✓ should update the service (188ms)
      ✓ should return a 404 status code as the id does not link to a service
    delete /service/:id
      ✓ should delete the service associated with the uri (155ms)
      ✓ should return 404 status code as the uri is not associated with a service


  113 passing (7s)
  4 pending

server stopped!
[nodemon] clean exit - waiting for changes before restart
```

References

Development:
https://www.youtube.com/watch?v=ngc9gnGgUdA&t=2112s
http://react.tips/radio-buttons-in-react-16/
https://blog.hubspot.com/marketing/embed-social-media-posts-guide
https://www.youtube.com/watch?v=wxz5vJ1BWrc
https://css-tricks.com/exposing-form-fields-radio-button-css/
https://www.youtube.com/watch?v=y7yFXKsMD_U&t=4398s
https://github.com/jquense/yup
https://formik.org/docs/guides/validation
https://www.youtube.com/watch?v=NgWGllOjkbs&ab_channel=RemyFamily

Testing:
https://blog.sapegin.me/all/react-testing-4-cypress/
https://testersdock.com/cypress-fixtures/#:~:text=Fixtures%20are%20used%20to%20store,be%20used%20throughout%20your%20tests.
https://docs.cypress.io/guides/references/best-practices#Having-tests-rely-on-the-state-of-previous-tests
https://www.youtube.com/watch?v=N_9LLQ9B5Fo&ab_channel=TheTestingAcademy
https://www.youtube.com/watch?v=dVRivkL5eGc&ab_channel=BasaratAli
https://github.com/herrkraatz/react-unit-testing