B.Sc. (Hons) in Software Development

Ollscoil Teicneolaíochta an Atlantaigh

Atlantic Technological University

ECHO

By
RONAN SHAUGHNESSY

for
MARTIN HYNES

April 21, 2023

Minor Dissertation

Department of Computer Science & Applied Physics,
School of Science & Computing,
Atlantic Technological University (ATU), Galway.

Contents

1 Introduction **2**
- 1.1 My Idea . . . 2
- 1.2 Technology Stack . . . 3
- 1.3 Objectives . . . 4
- 1.4 Aims . . . 5
- 1.5 Overview . . . 5
  - 1.5.1 Methodology . . . 5
  - 1.5.2 Technology Review . . . 5
  - 1.5.3 System Design . . . 5
  - 1.5.4 System Evaluation . . . 5
  - 1.5.5 Conclusion . . . 6
  - 1.5.6 GitHub Repository . . . 6

2 Methodology **7**
- 2.1 Requirements Gathering . . . 7
- 2.2 Planned Approach to Project . . . 7
- 2.3 Methodology Approach . . . 8
  - 2.3.1 Extreme Programming . . . 8
  - 2.3.2 Behaviour Driven Development . . . 8
- 2.4 Validation . . . 8
  - 2.4.1 White & Black Box Testing . . . 8
  - 2.4.2 Manual Testing . . . 9
  - 2.4.3 Automated Testing . . . 9
- 2.5 Source Control . . . 9
  - 2.5.1 Jira Software . . . 9
  - 2.5.2 GitHub . . . 10

3 Technology Review **11**
- 3.1 React . . . 11
  - 3.1.1 React Redux . . . 11
- 3.2 Node.js . . . 12
  - 3.2.1 Node.js Asynchronous Programming . . . 12
  - 3.2.2 Axios . . . 13
- 3.3 Express . . . 13
  - 3.3.1 Express as middleware . . . 13
  - 3.3.2 RESTful Express . . . 14
  - 3.3.3 Express with HTTP . . . 14
- 3.4 MongoDB . . . 14
  - 3.4.1 MongoDB a NoSQL Database . . . 15
- 3.5 Coding Languages & Markup Languages . . . 15
  - 3.5.1 JavaScript . . . 15

3.5.3 TypeScript . . . 16
3.5.4 TypeScript vs JavaScript . . . 16
3.5.5 JSX . . . 16
3.5.6 HTML . . . 17
3.5.7 CSS & Material-UI . . . 17
3.5.8 JSON . . . 17
3.6 Cypress . . . 18
3.6.1 Cypress Architecture . . . 18
3.6.2 Cypress Features . . . 18
3.6.3 Cypress & Cucumber . . . 19
3.7 Jira Software . . . 20
3.8 Kommunicate . . . 21
3.9 Platform as a Service . . . 21
3.9.1 Heroku . . . 21
3.9.2 Hostinger . . . 21

4 System Design **22**
4.1 Architecture . . . 22
4.2 Front End Component Tree . . . 23
4.3 Front End Home . . . 23
4.3.1 Rendering Home Page . . . 24
4.3.2 Search . . . 24
4.3.3 Viewing Posts . . . 24
4.3.4 Creating Posts . . . 24
4.4 Front End Form . . . 25
4.5 Front End Pagination . . . 25
4.6 Front End Navbar . . . 26
4.7 Front End Auth . . . 26
4.7.1 Front End Sign Up . . . 26
4.7.2 Front End Sign Up Verification Email . . . 27
4.7.3 Front End Sign In . . . 28
4.8 Front End Posts . . . 28
4.8.1 Front End Post . . . 29
4.9 Front End Post Details . . . 29
4.10 Front End Post Details Comment Section . . . 29
4.11 Miscellaneous Front End Post Features . . . 30
4.11.1 Circular Progress Bar . . . 30
4.11.2 Kommunicate Bot . . . 30
4.12 Back End . . . 30
4.13 Back End Model View Controller . . . 30
4.14 Back End Controllers . . . 31

4.14.1 Back End post.js controller . . . 31
4.14.2 Back End user.js controller . . . 32
4.14.3 Back End auth.js middleware . . . 33
4.15 Back End Models . . . 34
4.15.1 User Model . . . 35
4.15.2 Post Message Model . . . 35
4.16 Back End Routes . . . 35
4.16.1 Posts.js . . . 36
4.16.2 Users.js . . . 36
4.17 Deployment . . . 36
4.17.1 Heroku . . . 36
4.17.2 Hostinger . . . 36

5 System Evaluation **38**
5.1 Cypress . . . 38
5.2 Objectives . . . 38
5.2.1 To create a dynamic single page application . . . 39
5.2.2 The user can register their own account with the application 40
5.2.3 The user can sign into their own account once registered . . 41
5.2.4 A user is be able to post pictures, they wish to share with their friends/family . . . 41
5.2.5 Users can search for certain memories using taglines or memory titles . . . 42
5.2.6 Users can send messages between each other . . . 43
5.2.7 A user is allowed to like a picture, and comment underneath it 44
5.2.8 Users can create bio's underneath their posts . . . 46
5.2.9 Users should be able to login to their profile and logout . . . 46
5.2.10 Designing test scenarios to use in conjunction with cypress automated test tool . . . 48
5.2.11 To have a fully automated test features to verify the application for commercial use, using test case scenarios designed 48
5.3 Automation Results In Cypress . . . 50
5.4 Limitations . . . 50

6 Conclusion **51**
6.1 Aims . . . 51
6.1.1 Aim One . . . 51
6.1.2 Aim Two . . . 52
6.1.3 Aim Three . . . 52
6.1.4 Aim Four . . . 52
6.2 Jira Roadmap . . . 53

6.3 Other Learning's . . . 53
6.3.1 Using JSON . . . 53
6.3.2 Creating a dynamic single page application . . . 53
6.3.3 Using REST . . . 53
6.3.4 Model View Controller . . . 53
6.4 Succeeding in Objectives? . . . 54
6.5 The Applications Future . . . 54
6.6 Credit . . . 54

7 Appendix **55**
7.1 GitHub Link . . . 55
7.2 Installation Instructions . . . 55
7.2.1 Prerequisites . . . 55
7.2.2 Installing Test Environment . . . 55
7.2.3 Installing The Application Locally . . . 56

List of Figures

1.1 Application Homepage. 3

2.1 Gantt Chart 7
2.2 White Box vs Black Box Testing 9

3.1 Node server makeup using on thread 13
3.2 A brief look at express as a middleware 14
3.3 JavaScript as a class 16
3.4 TypeScript as a class 16
3.5 Node running the server side for Cypress 18
3.6 Cypress running in the browser supported by node.js 19
3.7 Step Definition 20
3.8 Gherkin Feature 20

4.1 System Architecture 22
4.2 React Component Tree 23
4.3 Home Page 23
4.4 Searching dynamically using tags 24
4.5 Form rendered on home page 25
4.6 Pagination rendered on home page 26
4.7 Sign Up Form 27
4.8 Welcome email user receives 27
4.9 Sign In Form 28
4.10 Page Details 29
4.11 Model View Controller Diagram 31
4.12 Controllers present in file system 31
4.13 Controller, creating a post 32
4.14 Controller, signing in a user 32
4.15 Logic to send an email to signed up user 33
4.16 Logic in the auth.js middleware 34
4.17 Post Message Model 34

4.18 User Model . . . 34
4.19 Posts.js Routes . . . 35
4.20 Users.js Routes . . . 35
4.21 Heroku Deployment . . . 37
4.22 Connection of server to client . . . 37

5.1 Specs for testing the application . . . 38
5.2 A dynamic id assigned to a post . . . 39
5.3 Searching in use Dynamically . . . 40
5.4 Paginate Dynamically through pages . . . 40
5.5 Specification for signing up a user manually passing . . . 40
5.6 Specification for signing in a user manually passing . . . 41
5.7 User then signed in . . . 41
5.8 Visual representation of created post . . . 41
5.9 Automated test for user creating a post . . . 41
5.10 Search bar test . . . 42
5.11 Using search bar test case . . . 42
5.12 Asserting tagline working . . . 43
5.13 Tagline search test case . . . 43
5.14 Asserting tags and search title working together . . . 43
5.15 Tags and search title test case . . . 43
5.16 Liking a post when signed in . . . 44
5.17 Test case for liking . . . 44
5.18 Unliking a post . . . 45
5.19 Cypress automating a comment . . . 45
5.20 Test case for commenting on a post . . . 45
5.21 Cypress automatically creating a bio . . . 46
5.22 Test Feature for the above objective . . . 47
5.23 Cypress asserting a user can log into their profile . . . 47
5.24 Simulating clicking logging out . . . 48
5.25 After click assert sign in form appears . . . 48
5.26 Five E2E Features . . . 50

6.1 Agile Board . . . 53

List of Tables

5.1 Cypress Automated Testing Results 49

Chapter 1

Introduction

During the course of this year, I will be undertaking a project, which to my hope shall be up to standard of a level eight degree in software and development. Initially I was given the option to be assigned a group, or to work on my own. I have chosen to work by myself, as I feel I can assign myself a good level of scope for the project required. Additionally, I did not wish to be assigned into a group as I had done group work in the past, which worked well however, I never had a say in the application my team was making, this made it really hard to gel into the team, as some design choices were not feasible with too much scope taken into the project. I feel if I operate by myself I can make good design choices and which can be reflected in my project. At the start of the year I coined with working on a project in unity, for example a 3d shooter, with a simple AI of enemies to come and find the player around a medium sized map. However after some thought I imagined the scope would have been too big for one person to take on. My next idea was to create a MERN(Mongo, Express, React, Node) [1], CRUD(Create, Read, Update, Delete) [2] application, to upload memories in a social app setting. This application would also showcase new skills I wished to take on such as testing using cypress, which is a JavaScript testing tool to automatically simulate user behaviour such as button clicks. I decided this application would best suits my skills, as I can engage with new technologies in particular, automated testing. I decided to call the application Echo, I felt this was a unique name as it means to be reminiscent of the past, in which users of the application, may post memories or adventures of their past.

1.1 My Idea

My intention is to create a MERN stack application to compliment cypress test automation, with a sign in/register system, sending messages to other users, up-

loading pictures, liking, and commenting on pictures. In conjuncture to have a fully automated test suite set up using cypress, test cases will also be created to accompany and structure the JavaScript code in cypress, test cases are instructions on how the program should be used, simulating behaviour of a user. My hope is to create a friendly environment where users can upload their travels around the world with friends, or cool hobbies. In addition showcasing new technology for example cypress, in which test automation is used to reduce time from manually testing. Below is a wire frame of how I wish the application to look.

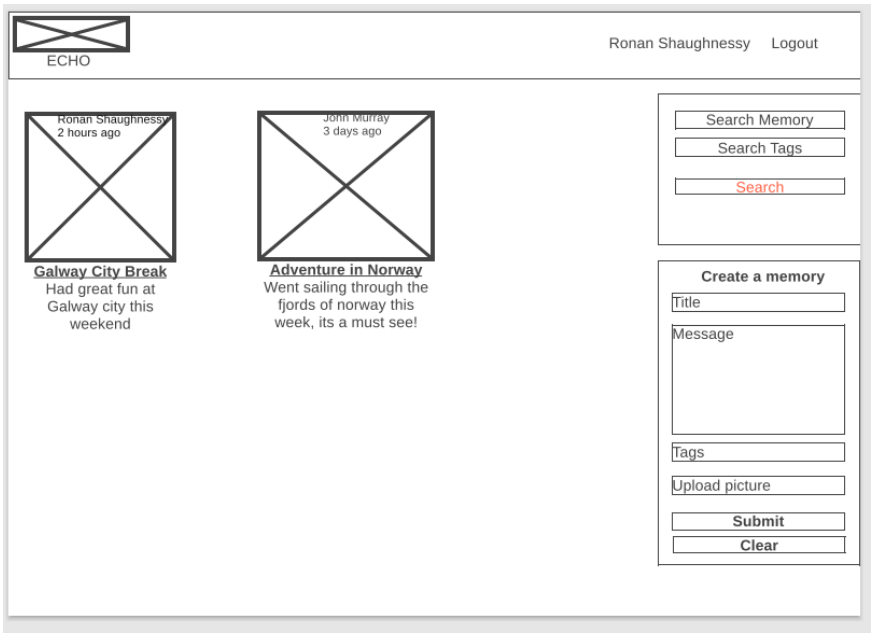

Figure 1.1: Application Homepage.

1.2 Technology Stack

I plan to use a variety of programming languages I have acquired during the last three years for instance, HTML, CSS, JavaScript, and TypeScript. I decided on MongoDB for my database as according to research I conducted, it stores records as documents, combined with mongoose it can create documents for storing pictures, or any other storage the user will need. I opted for MongoDB over MySQL as I used MySQL often over the last three years, and I wished for a new challenge. I wanted to design the structure of my documents with MongoDB as I find it easier to change Mongo's schema compared to MySQL. I will connect my front-end and back-end using express.js, node.js. Lastly, I will use React for my front-end. I picked React

over Angular as during my research I found that React is better for larger web applications as it is more light weight, and I can develop quicker. I feel angular's framework is more rigid and strict combined with the fact React is used more in 2022. Additionally, I wish to improve my skills in software testing, to this end I will employ cypress automated testing to complete testing automatically using TypeScript, and cucumber. Cypress with the aid of TypeScript can create scripts which simulate button clicks, routing to pages etc. This will make testing faster, improving my skills in software testing. I have chosen cypress over selenium as I have never covered cypress, to give me a new challenge and to automate my tests. Following research, I found that cypress manages execution time performance more efficiently than selenium, cypress uses less source lines of code(SLOC) and looks neater when reviewing test cases in the browser [3].

1.3 Objectives

The paramount goal is to develop a single-page application, dynamically updating the application, which then allows the user to display photos of their memories and adventures. Moreover to create an application which simulates user behaviour such as button clicks automatically. In order to reach this goal, the subsequent objectives must be present in the application.

- To create a dynamic single page application.
- The user can register their own account with the application.
- The user can sign into their own account once registered.
- A user is able to post pictures, they wish to share with their friends/family.
- Users can search for certain memories using taglines or memory titles.
- Users can send messages between each other.
- A user is allowed to like a picture, and comment underneath it.
- Users can create bio's underneath their posts.
- Users should be able to login to their profile and logout.
- Designing test scenarios to use in conjunction with cypress automated test tool.
- To have a fully automated test features to verify the application for commercial use, using test case scenarios designed.

1.4 Aims

The purpose of creating the application, what I expect to achieve during the development of the website.

- To build a highly pleasing front-end website, which will have a great appearance to please all who visit the application.
- To create multiple test scenarios, and have such scenarios fully automated using cypress.
- To improve my website design skills using React and JavaScript.
- To further my ability using automated testing.

1.5 Overview

A brief description of the chapters in this dissertation which illustrate the rationale behind designing this project.

1.5.1 Methodology

This segment describes the work undertaken during the development stage, with particular emphasis on approach to development, validation, testing and tools used to aid development.

1.5.2 Technology Review

This section outlines the technologies which were used in this project, setting up the aforementioned technologies and how the technologies were applied throughout the project.

1.5.3 System Design

This chapter will detail how the project was assembled, how the back-end interacts with the front-end, and will provide a comprehensive description of the overall system architecture.

1.5.4 System Evaluation

This segment will evaluate if the project met the objectives listed in the introduction, and discuss how effectively the system was implemented.

1.5.5 Conclusion

The last chapter examines if the objectives arrayed in the introduction chapter were met, trials which occurred during the development of the project, and how such difficulties were overcome.

1.5.6 GitHub Repository

The GitHub repository for ECHO, my final year project can be found by clicking here.

Chapter 2

Methodology

The current section concerns the approaches which were made in undertaking this project, what validation was used, and how units of work were tracked during the course of the project.

2.1 Requirements Gathering

Requirements for this project were gathering in chapter 1, the specific sections where the requirements for the application can be found in these sections, 1.3, 1.2 and 1.1

2.2 Planned Approach to Project

The original planned approach to the application can be seen below in the form of a gantt chart.

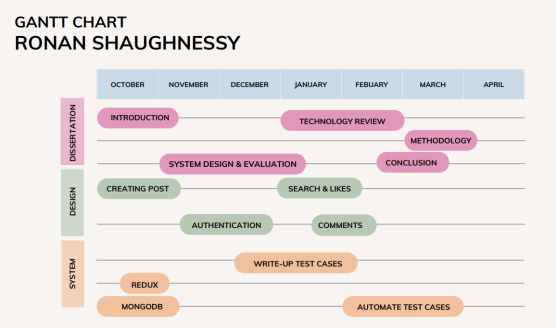

Figure 2.1: Gantt Chart

2.3 Methodology Approach

Rather than one single approach, multiple methodologies were used in the creation of the application, Extreme Programming for development, Behaviour Driven Development for testing the application, Jira Software agile tool was also used to track progress alongside GitHub.

2.3.1 Extreme Programming

The approach which was undertaken in order to complete tasks done each week was extreme programming(XP) [4]. XP is a agile framework, which takes the agile methodology to the extreme, in short this means shorter sprints combined with more tasks in each sprint. The aims of using XP was to make the most use out of the limited amount of time that was set for the creation of the project, with this in mind maximising all the time during the college year was crucial to the applications success. Lastly XP can handle changes to the requirements smoothly, if the need was deemed high enough for the application.

2.3.2 Behaviour Driven Development

Behaviour Driven Development(BDD) is developing the application, to behave as expected if the application does not function as thought, then work is needed to improve to make the system in order to make it compliant with user behaviour and allow the application to work with ease [5]. BDD was employed to great extent during the testing phases of the application, where test cases needed to be designed based on how the user would interact with the program, in order to assert if it works as planned.

2.4 Validation

Instruments which were used in order to test the application ensuring its appropriate use.

2.4.1 White & Black Box Testing

Black box methodology was employed using BDD, as BDD takes the role of a user who doesn't care how the applications code base works, only that the application does what is expected. This was used in automated testing of the application, as the Cypress testing tool has no prior knowledge of the internal code base, only simulating user behaviour. White box methodology was employed at a manual

test level, as I had a clear idea of the code base which I was working on in order to test the application.

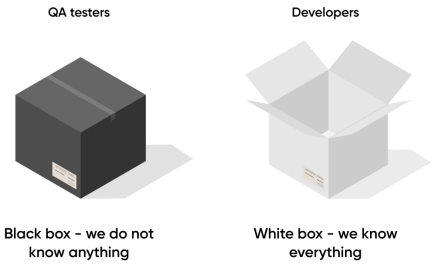

Figure 2.2: White Box vs Black Box Testing

2.4.2 Manual Testing

As previously stated, white box methodology was used during manual testing of the application. When certain restful APIs were called using the application's internal code base, a console.log would be applied to the controller at the back end to determine whether or not there was an error. The aforementioned testing was performed on the application several times throughout the project, using Google's console in the browser to assert if there were errors in the code, this was also accomplished quite successfully with the use of status codes such as the 400, 500 and 200.

2.4.3 Automated Testing

Automated validation was conducted with black box methodology in mind, in which the application would simulate user behaviour for example liking a post once signed in, or creating a post. Automated testing was employed to ensure the front end of the application was working correctly as a user would intend for it to work.

2.5 Source Control

Tools which were used in order to track and manage the software programming.

2.5.1 Jira Software

Jira was used in conjunction with extreme programming to track tickets/tasks. Atlassian stories were created under epics and then brought from the backlog into

a new sprint. When a previous sprint finished, tasks were taken into a sprint from the backlog then a new sprint was started, since it was a one man team, each sprint varied in time in custom with XP. At the end of each sprint, the tasks done were evaluated as to how well the aforementioned tasks were completed, after which tickets would be updated if they needed to be brought into the next sprint for improvement, if bugs were found through the use of testing preformed such bugs were logged to be solved in the next sprint, any enhancements which could be made to the application would also be taken into the next sprint.

2.5.2 GitHub

Once a number of tickets were completed during a sprint for example two, then the programming workload would be committed to GitHub, when used in conjecture with Jira it aided to track the projects progress. This was to ensure that the application was always in a stable state, for example if work proceeded after the latest commit, and the code that was being worked on during the latest ticket introduced a breaking bug that halted the application, GitHub could be used to revert the code base, thereby getting the application back to its previous working state.

Chapter 3

Technology Review

This chapter, will discuss the technology used in the projects stack on a theoretical level, detailing features of each piece of technology used in this project.

3.1 React

According to stackoverflow [6], react.js is the second most common technology for making web based applications at 42.62%. React was originally launched in May 29 2013, a common misconception is that react is a framework, this is not true, as react is a massive library in which a programmer can make interfaces that revolve around react components [7] this along with other features aids to rapid development of a application. React uses the command, create react app to make the application into a framework. React has a seamless development time, making it easier to get applications running using this technology aiding to its popularity over the years [8].

3.1.1 React Redux

The project itself lends a great deal of logic from the redux library of react. These two technologies work very well together and normally go hand in hand [9]. Redux is widely used in this application when state must be changed dynamically without the user noting the change, this greatly supports making a single page application [10]. Redux is best summarized in its key concepts which are employed in the application [11].

Store

Store in redux is where the state is stored, this state is then obtained from the store using the getState method, after which the state can be use with the useDisptach

method to forward actions.

Actions

Actions are responsible for sending the data into the Redux store, actions are the data which alert the store that the state has changed in the application. Redux actions must be a type object, and have other key data so as they are distinguishable to the reducers.

Reducers

Redux reducers are viewed as clear functions, such functions take the preceding state and action, then making them inputs thereby returning a new state. A redux reducer script can also be seen as finite state machine, operating on switch statements to switch the state depending on the redux action [12].

Asynchronous Actions

Lastly redux employs asynchronous actions which can fetch the data from the back-end using redux thunk, after which a response can be handled, and a action can be dispatched so as to update the state of the application. Thunk itself is a middleware, that greatly aids in processing the necessary requests.

3.2 Node.js

Node is a JavaScript run time environment, some of its biggest features include allowing programmers to code in JavaScript on the server side of their application, node can be employed across platforms regardless of OS, and enables programmers to make use of asynchronous programming, so as developers can handle multiple requests in their applications [13][8]. Moreover applications can be launched readily and quickly due to node.js package manager (NPM), greatly aiding a developer to create a application in quick schedule [14].

3.2.1 Node.js Asynchronous Programming

As eluded to above, Node.js offers asynchronous programming as one of its mainstay features. Such a feature enables a node server to be event driven, and lock free I/O, aiding node in being able to deal with concurrency effectively. Node is able to parse all requests made to it, thereby handling requests in a concurrent fashion using the space execution memory available to it in its one threaded makeup [15].

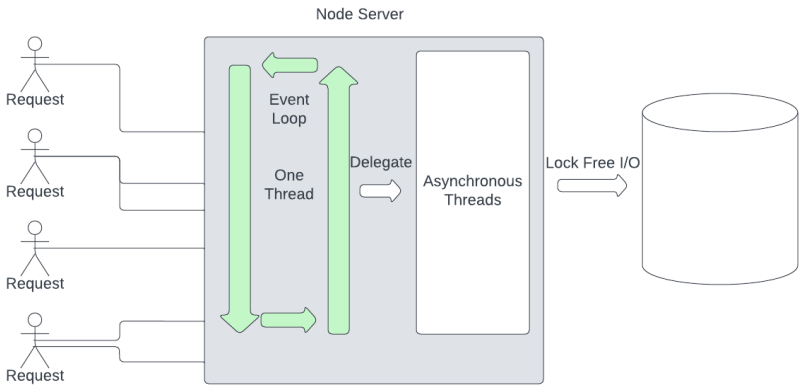

Figure 3.1: Node server makeup using on thread

3.2.2 Axios

Node interacts with react at the front end via HTTP request response, axios helps to streamline this relationship by acting as middleware between node and react, allowing both to send HTTP requests or responses between the server and client sides [8][14].

3.3 Express

Express is a node framework which lends in creating web application, express is normally used in conjecture with node, and can be installed in a application using NPM. Express itself provides an abundance of features for example routing middleware or HTTP capabilities [16].

3.3.1 Express as middleware

Express as a middleware sits between the client and the server, in order to handle request and responses. Express uses functions to take in a request, thereafter such request functions will send back a response, errors when unsuccessful, or relevant data if successful [17]. In the code snippet below, express is used to log a request and the url to the console. The app.use() method is called, indexing the middleware function, logrequest() is indexed to app.use(), meaning this method will be executed at every incoming request [18].
In the project application of this dissertation an example of how express is used, is to create a request response for users signing into the application.

Figure 3.2: A brief look at express as a middleware

3.3.2 RESTful Express

Express supports building RESTful APIs for web applications, ways in which Express promotes RESTful usage are, keeping the client side and the server side separate, implementing design patterns for developing web APIs such as aiding in identifying functions to be called for usage, for example get, post, or put. Express can then manipulate such resources over HTTP [19], helping to develop an application to be REST compliant by providing robust APIs for handling HTTP requests and responses. Furthermore express promotes REST by handling errors when they occur, or by stating how a user will receive data.

3.3.3 Express with HTTP

Express is built on node.js, using http to access all http functionality, for example get, post being the most common. Express uses http to listen then respond to front end requests [20]. The above-mentioned features allows developers to handle http request through express in the way they see fit, based on the URL and method call, these points make express ideal for making web applications quickly and efficiently.

3.4 MongoDB

Originally developed in 2009, MongoDB is a open source document orientated, NoSQL database which stores, retrieves and manages data for applications [21]. MongoDB is adeptly suited for numerous applications, for example blog sites which tend to produce a lot of unstructured data, in this case MongoDB is better equipped in dealing with such data in comparison to a RDBMS [22]. It can be seen as human readable as it stores documents in a JSON like format [23]. MongoDB prides itself on having easy to use interfaces, and being able to bring hugely adaptable databases to an application. Beside the features mentioned above, MongoDB ensures acid compliance in their database systems guarantying the validity of data, and uses map reduce as a feature [22], which is a data processing model that lends itself to executing actions on sizeable data collections and then assisting in the aggregation of results.

3.4.1 MongoDB a NoSQL Database

There are three types of NoSQL databases, one of which is a document store, MongoDB falls into this category. Document-stored databases are classified as a collection of key-value stores that are then converted into documents and stored in the database [23]. One of NoSQL's key features is being able to handle unstructured data to great effect, this is given its rise to popularity in recent years [22]. Where NoSQL differs from the conventional RDBMS, is that MongoDB's schema is not established in place when compared to a RDBMS, MongoDB will use ID keys in order to retrieve data, as each piece of data is assigned a unique ID key. In reference to the aforementioned, a read/write can only be preformed using the ID key assigned [24].

3.5 Coding Languages & Markup Languages

The below technologies represent the coding languages used in the creation and development of the application.

3.5.1 JavaScript

Developed in 1995, JavaScript or sometimes known as just js, originally created for Netscape 2, although after some time js was preferentially used by Mozilla where it was used to develop web capabilities for Firefox [25]. JavaScript is a interpreted language i.e for the computer to understand js, it needs to be complied into something it can understand, normally executable byte code [26]. With the aforementioned in mind, JavaScript code is interpreted at run time, it is not complied before run time, like for example Java. Attributing to the above, JavaScript is dynamically typed, variables will be assigned at run time, this differs from static typed languages where variables are known at compile time [27]. As such for creating web systems, when a application is running locally, changes can be made to the JavaScript code, then the changes will be shown immediately on the application, provided there are no errors in the code.

3.5.2 JavaScript an object orientated language?

JavaScript is a object orientated language however, js is used more as object based, rather than upholding the principles of object orientated programming. JavaScript as a language promotes the use of being able to change or being agile, as the above mentions js is weakly typed, interpreted, and variables could be removed or added at run time [28]. While js is a object orientated language it is normally not used

in such a way, TypeScript a offshoot of JavaScript, is popularly used to adhere with oop principles, which in of itself is a superset of JavaScript.

3.5.3 TypeScript

TypeScript is a superset of JavaScript, this means that TypeScript can understand all JavaScripts syntax and capabilities [29]. TypeScript does more to uphold the principles of object orientated programming, by bringing in classes, interfaces, and is statically typed [30] catching errors before run time, in many ways TypeScript looks at what JavaScript doesn't do well, then adding features to JavaScript to aid developers. TypeScript does not have a interpreter, instead it compiles to JavaScript as browsers cannot understand TypeScript [31]. With the above facts in mind, TypeScript is normally used now in favour of JavaScript, as if a person learning web development for the first time, TypeScript uses classes more frequently, and it statically typed language making it more familiar to new comers when compared to JavaScript. Moreover it can be used quite avidly by programmers who use JavaScript as TypeScript understands JavaScript, as it only adds to JavaScript features.

3.5.4 TypeScript vs JavaScript

Below, is a code snippet of TypeScript as a class vs JavaScript as a class, as shown below TypeScript has variables brought in before the constructor i.e variables are known before run time, when compared to JavaScript the variables let person = new Person("Alice", 30); are only assigned during run time.

```
class Person {
    constructor(name, age) {
        this.name = name;
        this.age = age;
    }

    sayHello() {
        console.log(`Hello, my name is ${this.name} and I am ${this.age} years old.`);
    }
}

let person = new Person("Alice", 30);
person.sayHello();
```

Figure 3.3: JavaScript as a class

```
class hello {
    private name: string;
    private age: number;

    constructor(name: string, age: number) {
        this.name = name;
        this.age = age;
    }

    sayHello() {
        console.log(`Hello, my name is ${this.name} and I am ${this.age} years old.`);
    }
}
```

Figure 3.4: TypeScript as a class

3.5.5 JSX

JavaScript syntax extension or JSX, which like TypeScript is a addendum to the already established JavaScript. JSX allows developers to outline Reacts object structure into syntax that is comparable to HTML. Simply put JSX, is an XML

related addition, allowing programmers to write JavaScript that can appear as a HTML like, and then have it returned as a component in React [32].

3.5.6 HTML

Hyper Text Markup Language (HTML) is a markup language which is used in representation of web applications. HTML is crucial in any web development as it details the architecture of a web application i.e it explains how the application should be rendered, however the application need not be written in HTML, as it is unable to write functions or carry out complex tasks when compared to JavaScript [33].

3.5.7 CSS & Material-UI

Cascading Style Sheets (CSS) can be closely aligned with the use of HTML or other markup languages, CSS like HTML is also used in changing how a web application is presented, although unlike HTML, CSS tells the application what color, layout or general style elements of the web page should be using [34]. The creation of pleasing designs can be employed quickly using a CSS framework Material-UI, which follows googles material design patterns, Material-UI comes with existing React components, for example SVG icons, uniformity in displaying forms or navigation bars, and other features. Material-UI can help developers worry less about the look of a application when compared to using normal CSS, as Material-UI will handle most of the aesthetic of the application [35].

3.5.8 JSON

JavaScript Object Notation, allows programs to represent JavaScript objects in plain text, JSON is used to serialize and parse objects over the network, normally between the front end and the back end. However this comes at a price as the plain text used, are much larger in size when compared to their binary equivalent, therefore parsing must be preformed on both sides for example in client side and server side [36].

JSON Web Token

Web tokens are used to identify if users are verified on certain systems for instance, if a user signed into Facebook, a web token would be assigned to them, asserting a authenticated user [37]. A JSON web token uses a JSON object to assert safety between two parties normally the user and the client or application the user wishes to use [38]. Authorization is the most regular use of a JSON web token, when a

user is logged into Facebook for example, they will then be able to avail of the activities which Facebook advertises, using their JSON web token which is stored in local storage, all of which is hidden from the users view [37].

3.6 Cypress

Cypress automation test tool is a relatively new web automation tool only being released in 2017. Cypress is a client side testing tool created for web applications, Cypress greatly encourages Behaviour Driven Development, building tests side by side while creating a web application, then asserting such tests pass as development advances [39].

3.6.1 Cypress Architecture

Cypress is often contrasted with Selenium, although the two testing tools are architecturally different. Cypress is built using Node.js, and comes in its own npm module to be installed locally, then running npm to open cypress in a browser window in order to open the application the developer wishes to test [40]. Altering web traffic at will, is how Cypress performs operations at the network layer. This gives Cypress the opportunity to change what goes into and out of the browser, allowing Cypress to also adjust code that could cause glitch's or errors in the browser's automation [39], resulting in a smother testing experience, enabling the engineer to obtain accordant outcomes [41]. As Cypress is installed into a developers local machine using node, it allows the automation to use the local operating system in order to make use of the camera for taking screenshots of failing or passing tests, and recording videos of the automated test suite.

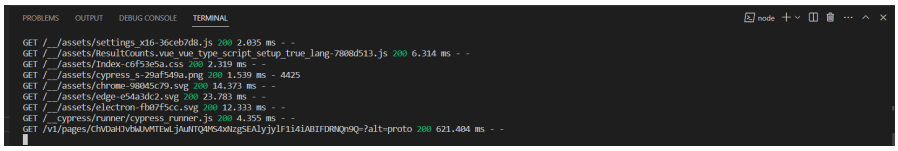

Figure 3.5: Node running the server side for Cypress

3.6.2 Cypress Features

Cypress contains a number of capabilities for example Cypress uses the wait command at will in order to wait for the DOM to load in correctly, this is needed greatly as Cypress tests sometimes preform blindingly fast, although if a test engineer wishes for extra time to be applied the cy.wait command can be hard coded

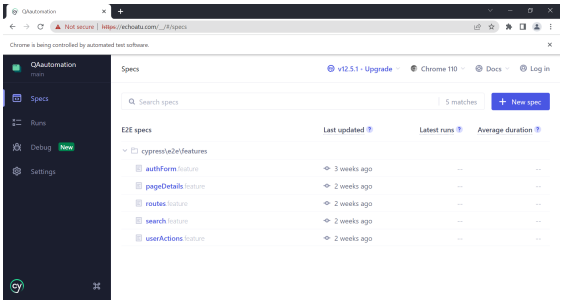

Figure 3.6: Cypress running in the browser supported by node.js

in [42]. JavaScript is the predominate coding language used in Cypresses testing framework, and in turn this means TypeScript too is greatly used in Cypress, considering many web applications are made in the aforementioned languages [43], this makes for a easy learning curve from Front End developer to test engineer and vice versa. Further traits that Cypress contains as testing capabilities are, the ability to use multiple browsers other than chrome for example Microsoft edge, Firefox, or electron, using Cypress dashboard to detect flaky tests i.e tests which passed but then began to fail, the tester can begin to pinpoint a flaky test and update the test in order to make it more reliable [39].

3.6.3 Cypress & Cucumber

Cypress can be further integrated, to use behavior driven development (BDD) with the aid of a tool called Cucumber.

Cucumber

Cucumber as mentioned above is a framework which buttresses BDD, Cucumber enables the user to create tests in gherkin syntax of which gherkin syntax is normal text which make up features [41]. The aforementioned features consists of various steps which Cucumber must work through in order to assert if the test is passing, or failing. Then the plain text is linked to step definitions, which is the coding section of Cucumber, thereafter the written code carries out the task assigned using the gherkin syntax e.g Then The echo should be deleted [44].

```
Given("A the relevant user is signed in", () => {
    cy.visit("/");
    cy.get('span.MuiButton-startIcon').first().click();

    cy.get('input[type="email"]')
    .type('CyAuto1@email.com')

    cy.get('input[name="password"]')
    .type('12345')

    cy.get('button.MuiButton-containedPrimary').eq(0).click();
    cy.wait(3000);

    cy.get('h6.MuiTypography-root.jss5.MuiTypography-h6').should('exist');
    cy.get('.jss7').should('exist');
});

When("The user clicks the delete button on their echo", () => {
    cy.get('button.MuiButton-textSecondary[type="button"]')
    .eq(0)
    .click();
});

Then("The echo should be deleted", () => {
    cy.get('button.MuiButton-textSecondary[type="button"]')
    .should('not.exist');
});
```

Figure 3.7: Step Definition

```
Scenario: User deletes their echo

Given A the relevant user is signed in
When The user clicks the delete button on their echo
Then The echo should be deleted
```

Figure 3.8: Gherkin Feature

Cucumber integration with Cypress

Cucumber is not installed with Cypress, thus it has to be installed in order to access gherkin syntax, and have it incorporated with Cypress. After Cypress is installed, Cucumber can further be added to the testing file by using these commands in the CLI, npm i @badeball/cypress-cucumber-preprocessor, this package allows programmers to define Gherkin syntax in running their Cypress tests [45][46]. Now Cucumber should is installed, Cypress should work using Gherkin syntax after changes to the configuration file [47].

3.7 Jira Software

Jira Software was originally created in 2002, it is a tool used by software development teams in order to track the amount of work done. Jira supports many agile methodologies which attribute as to the why it is used so widely in industry, it promotes change and ability to alter the software project [48]. As mentioned above Jira uses the agile perspective to software development, for example test driven development(TDD) in which the software requirements are made into cases before code is developed in the application, thereafter the tests will assert that the application fulfils the users expectation [49]. Jira makes the above possible with, team planning where tasks are created outlining the objectives to be done during the sprint, a sprint is a duration of work normally consisting of one or two week iteration, inside the sprint, tasks called stories which are assigned to team members to be completed before the end of the sprint, during the sprint daily stand up meetings are arranged to scale how well each team member are doing with their

tasks, after the sprint ends, their is a team review, and a new sprint starts again repeating the process [50].

3.8 Kommunicate

Kommunicate is a chat bot AI which can help a user if they need to ask questions about the application. A developer can make their own AI bot through kommunicate [51], customize it to suit the needs of their web application, and optimize the AI to answer an questions a user might conceive while visiting the application.

3.9 Platform as a Service

Platform as a service (PaaS) refers to services that enable developers to launch their applications into a cloud-based development environment. PaaS enables an application deployment without the need to worry for operating systems or other infrastructure [52].

3.9.1 Heroku

Heroku is a PaaS that allows developers to deliver applications online without going through the infrastructure set up [53]. Heroku is able to run programmes across the network in virtual repositories, after which are then executed in the browser during run time [54]. Such repositories can be scaled up or down based on the developers needs, for most one person projects or simple business applications only one container would be needed.

3.9.2 Hostinger

Hostinger, another form of PaaS which allows programmers to host sites on a domain. In turn this allows the developer to choose the name of their domain after their application is built. When a developers application is built, simply drag and drop the application from the local drive into the file manager hostinger provides. After which hostinger deals with deployment of the application, then being available on the browser using the domain name that was picked upon registering the application. Features hostinger provides includes, allowing web traffic into the associated domain to exceed bandwidth where hostinger will handle it, domain name can be provided after sign up, HTTP/3 usage, and GIT support are among the most useful [55].

Chapter 4

System Design

This section is concerned with how the application was made, with in-depth detail on the FrontEnd, BackEnd, architecture, deployment, how these features were made, and designed.

4.1 Architecture

The below figure 4.1 shows a system diagram overview architecture of the application

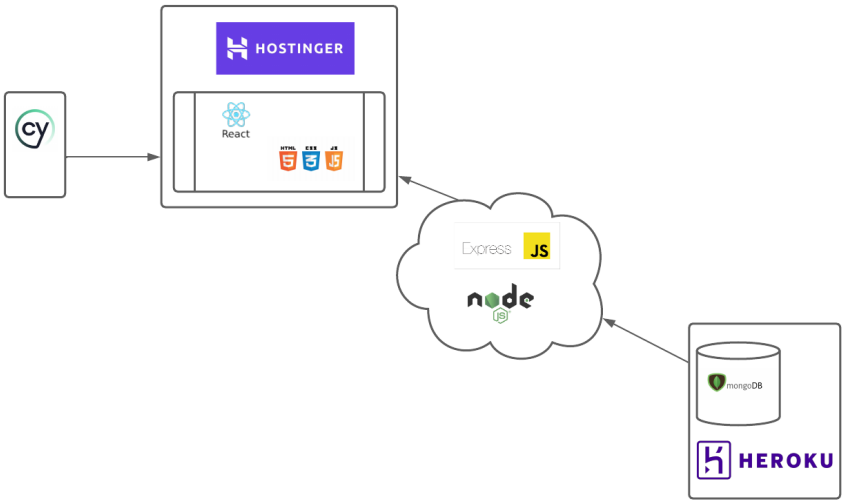

Figure 4.1: System Architecture

4.2 Front End Component Tree

The below figure 4.2 shows a high level tree of the various components in the front end of the application.

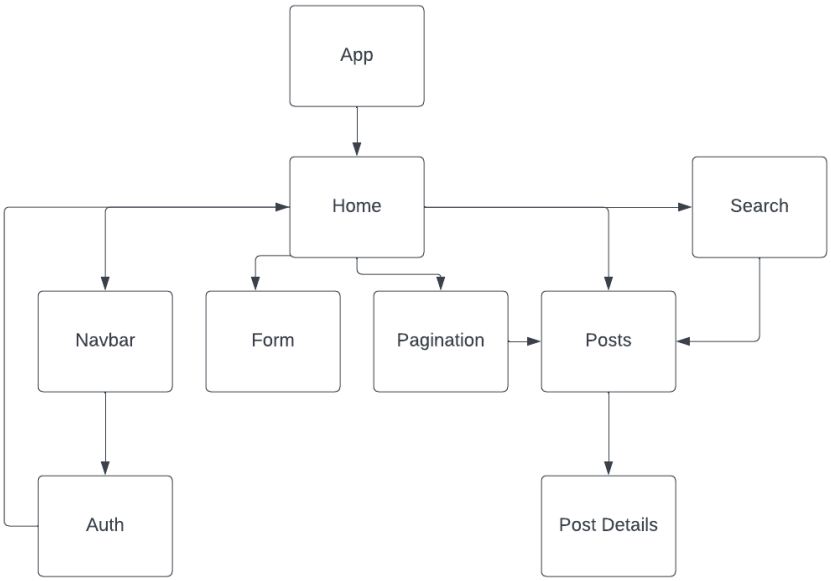

Figure 4.2: React Component Tree

4.3 Front End Home

The home page react component of the application encompasses rendering the home page, along with creating and viewing posts, lastly the home page is concerned with searching for a particular post.

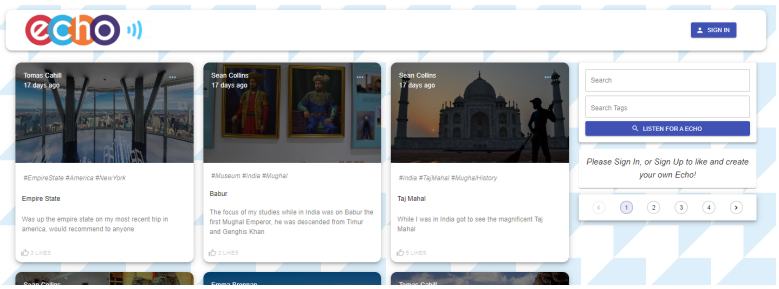

Figure 4.3: Home Page

4.3.1 Rendering Home Page

The home page renders various UI components such as Posts, Form, TextField, ChipInput, and Pagination using JSX syntax. In addition it applies Material UI styles to these components using the useStyles hook.

4.3.2 Search

Search uses the useState hook from react to store the search term, or if the user uses tags instead these are stored as an array of tags using useState, subsequently useDispatch is used when the user clicks on the search button updating the URL using history.push method then displaying the desired echo. The above mentioned, dynamically search's for specific tags or search bar term based on user inputs. The tags will only be inputted into the application when the user presses the enter button.

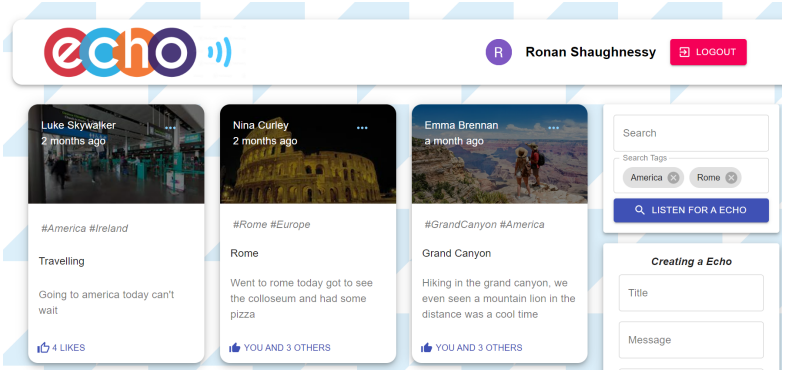

Figure 4.4: Searching dynamically using tags

4.3.3 Viewing Posts

The posts component is imported in order to view stored posts on the home page, setCurrentId is passed from posts allowing signed in users to be able to preform CRUD operations on a certain post. The setCurrentId function is defined using the useState hook. It updates the state of the currentId when the user selects a post to like or delete if applicable.

4.3.4 Creating Posts

Creating a post can be done on the home page using the form component in which the react useState is called. This hook is used to maintain its own local state for

the form fields, including postData, which contains the data for the post being created. Then when the handleSubmit function is called by clicking the submit button, a createPost action uploads the new post with the data inputted in the form and adds it to the application state.

4.4 Front End Form

The form react component is responsible for creating a post once the user is signed in, the form component is imported into the home component to be rendered on the homepage.

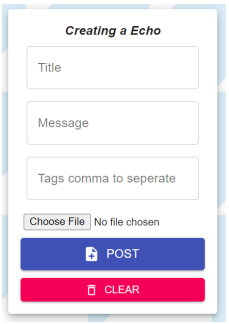

Figure 4.5: Form rendered on home page

The form only appears for a signed in user using JSON to parse local storage to see if a 'profile' is present, otherwise a message appears telling the user to sign in to create a echo. The currentID prop is used to see if the currentID is zero, if it is, then use dispatch to post the data through to the backend, when the relevant post details are filled out. Then the currentID prop is used to assign id's to the created posts, in which the id is assigned to the made post, and then is used when a user wishes to see a post in more detail under the post details component.

4.5 Front End Pagination

The Pagination component is a simple component which allows to user to view posts on different pages by accepting a page parameter, then using the useSelector hook to access the number of pages of which this value is stored in redux. Next the useEffect hook dispatches the action getPosts, in order to recover the number of posts for the page, of which six are displayed on each page. The useEffect hook contains a page parameter to dynamically change every time the getPosts action is called, this is to facilitate posts being deleted or created.

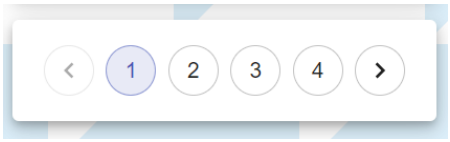

Figure 4.6: Pagination rendered on home page

4.6 Front End Navbar

The navbar component is responsible for rendering the navbar and displaying it at the top of the application 4.3. This component contains a button to sign in the user bringing the user to the auth route. When the user state is initialized using the useState hook, this parses a profile to assert if a user is signed in, then the navbar will display the signed in user and their avatar. Then when a user is signed in, the logout function can be used to dispatch a redux action to logout the user when the logout button is clicked, therein logging out the user. Finally when the user is signed in a JWT token is created, this token then expires after a specified amount of time automatically signing out the user with the useEffect hook.

4.7 Front End Auth

The auth component which can be sub typed into two forms which change if the user clicks the bottom button, launching the switchMode function to change the form to one of these options, sign in or sign up. Other functions such as the showPassword method is included, allowing the user to be able to click the eye button toggling the view of their password.

4.7.1 Front End Sign Up

Manually signing into the application can be done using the aforementioned bottom button and clicking it to switch the form, allowing the user to manually fill out their details. Error checking is preformed on this form to assert that the user cannot use an email that has already been picked, or if their passwords do not match.

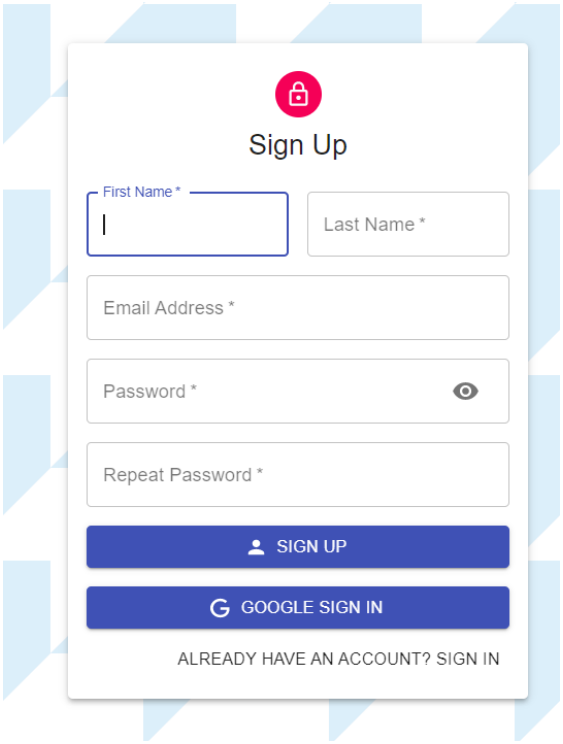

Figure 4.7: Sign Up Form

4.7.2 Front End Sign Up Verification Email

Additional verification for when a user signs up to the application is used, as a welcome email appears when a user has signed up correctly. This email is sent using a BackEnd controller which can be seen here 4.14.2. Below is the welcome the user receives following completion of signing up.

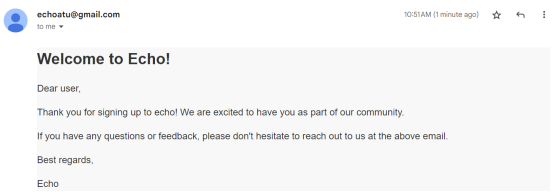

Figure 4.8: Welcome email user receives

4.7.3 Front End Sign In

A user can sign into the application in two ways, one is done manually when the user signs up, filling out their details. The users data is then stored into a object called formData, this data is relayed to the backend where it is stored in the database allowing the user to sign in later. The Second option for a user to sign in is using the GoogleLogin component from react-google-login, wherein a useEffect hook is called to load the Google API client library and initialize it with the client ID, thereby signing in the user without having to manually fill out their details, this is assuming the user has a google account.

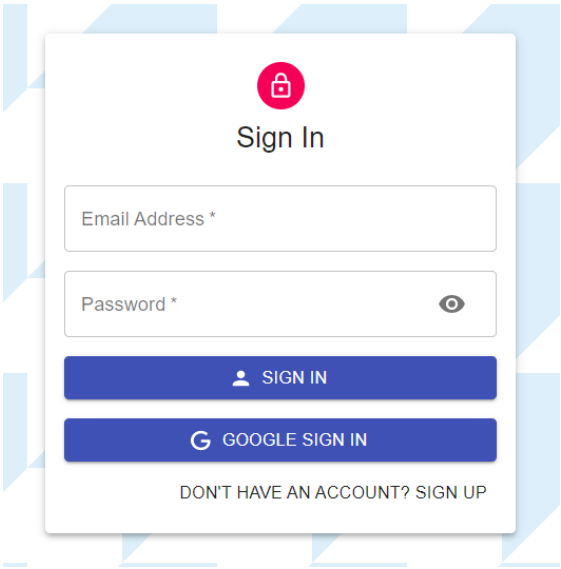

Figure 4.9: Sign In Form

4.8 Front End Posts

This react component manages rendering posts that the user makes in the form component as can be seen from this figure 4.3. Posts lends to the search logic as there is a if statement, stating if the user has searched and no such post exists then a warning message appears. If there are posts to display to the user, a map function is called to iterate over each post and displays each of the Post components which meet the search parameters.

4.8.1 Front End Post

Inside the posts component is a sub component called post that is in control of rendering each single post and their associated content which is pulled from the database. Assuming the user is signed in, functions such as liking and deleting posts will appear for the user. These actions are controlled by using, useDispatch to make a call to redux actions to handle the like, and deleting functions. The user can also click the ellipses on the top right to view a single post in more detail.

4.9 Front End Post Details

Post details provides an in depth look at a single post, using the useSelector hook provided by react-redux library to get the state data, then rendering the aforementioned data using conditional statements. Dispatch is used to call the getPost method that gets an id parameter to fetch the particular post the user has selected from the server. Posts with similar tags will be displayed on the bottom of the page, these posts are stored as a array of recommendedPosts, which are found by filtering posts in the redux store to exclude unrelated posts, based on their tags. These recommended posts can be clicked, then using the history object from react-router-dom to route to the selected post.

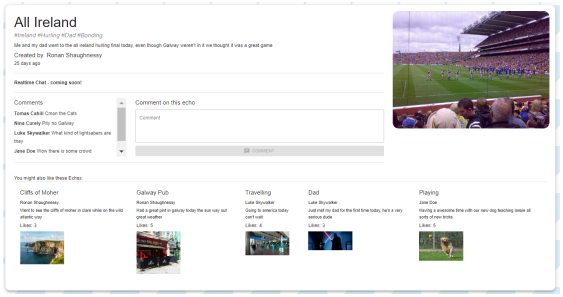

Figure 4.10: Page Details

4.10 Front End Post Details Comment Section

The comment section of the page details component is a sub component, the page details imports the comments section and renders it on the left side of the page details as seen in figure 4.10. React hooks such as useState are used to mange the comment state wherein comments are initially set to false provided there is no comments. A function called handleComment is called when the user types in a comment into the input box and clicking the comment button. This function gets

the user name from local storage so their name will appear next to the comment, then using redux to dispatch an action adding the new comment to the database, finally updating the comment state with the new comment clearing the old input.

4.11 Miscellaneous Front End Post Features

Feature which don't fall into any particular category in terms of high or low level design.

4.11.1 Circular Progress Bar

If internet access is not very fast where a user is accessing the application, a circular progress bar will appear so as the user is aware that the application is loading, and that the application has not crashed.

4.11.2 Kommunicate Bot

If a user is having difficulty using the program, an AI chat bot system has been set up to answer various inquiries if the user is lost or unclear how to perform something on the site. This was added into the indes.html file, with code provided by kommunicate to call an API which I set up containing the specifically configured bot for this application.

4.12 Back End

The back end of the application is a node.js server using express as a framework, then accessing the mongoDB database using mongoose to listen on the specified port.

4.13 Back End Model View Controller

The Model View Controller below 4.11, shows a high-level diagram of how the back end interacts with the application in general, where it relays information the user passes into the application, either submitting a response or adding information to the application.

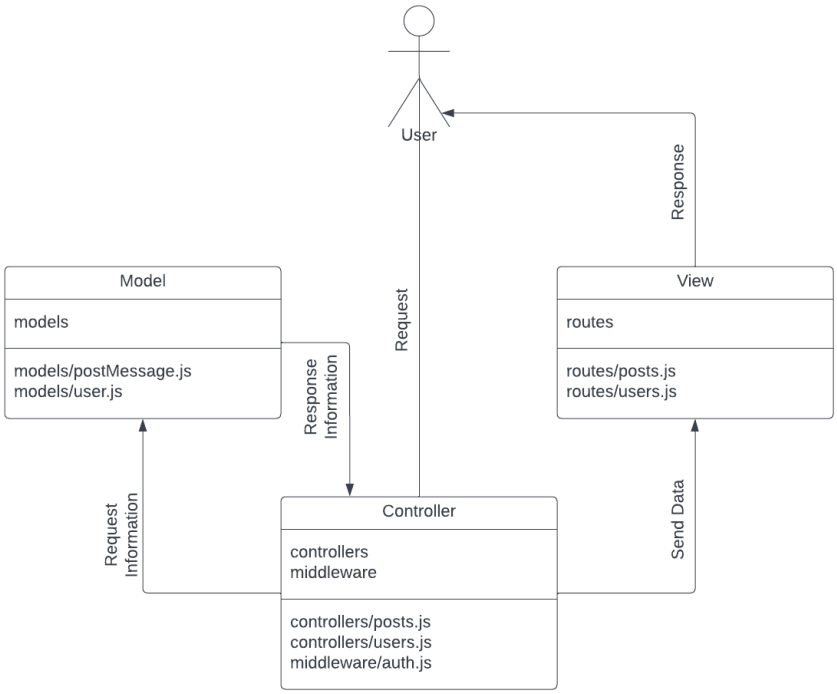

Figure 4.11: Model View Controller Diagram

4.14 Back End Controllers

Two modules which interact as controllers for the application are the middleware, and the controllers folder modules.

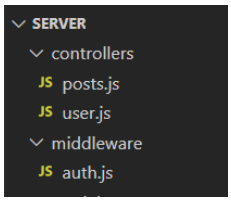

Figure 4.12: Controllers present in file system

4.14.1 Back End post.js controller

The post.js controller handles all routes except for the authorization routes. A router object is created at the top of the script, this allows the routes to be used in the front end of the application. A number of functions are declared in the

script, for example the createPost function in which will briefly be described. The aforementioned function creates a new post with the data the user sends to the back end in the request body, wherein the function then adds the users ID, time the post was committed, and saves it to the database. A status code will then be sent to the front end of 201 for a successful request response, or 409 if there is an error. The above-stated code can be view below 4.13.

```
export const createPost = async(req, res) => {
    const post = req.body;

    const newPostMessage = new PostMessage( {...post, creator: req.userId, createdAt: new Date().toISOString() });
    try
    {
        await newPostMessage.save();

        res.status(201).json(newPostMessage);
    }
    catch (error)
    {
        res.status(409).json({message: error.message});
    }
}
```

Figure 4.13: Controller, creating a post

4.14.2 Back End user.js controller

The user controller interacts in a similar way to the post controller, except in that the user controller is used as part of the authorization level of the application. Imported for this script is jsonwebtoken which is used to generate a JSON web token for authentication.

```
export const signin = async (req, res) => {
    const { email, password } = req.body;

    try{
        const existingUser = await User.findOne({ email });

        // if there is no user with that email in database
        if(!existingUser) return res.status(404).json({ message: "User doesn't exist"});

        // to compare the password normal string compare not good.
        // use bcrypt to compare the hash password to add level of security to passwords
        const isPasswordCorrect = await User.findOne({ password });

        // if the hashed passwords don't match user should not be able to sign in
        if(!isPasswordCorrect) return res.status(400).json({ message: "Invalid password"});

        const token = jwt.sign({ email: existingUser.email, id: existingUser._id}, secret, {expiresIn: "1h"});

        res.status(200).json({ result: existingUser, token});
    } catch(error){
        res.status(500).json({ message: 'Unidentified error'})
    }
}
```

Figure 4.14: Controller, signing in a user

The above figure 4.14, illustrates from a internal view, the programming involved

in creating authorization manually. In the above function signin handles logic related to logging in. Firstly, the function pulls out the email and password the user has submitted from the request body, checking against the database using the User.findOne method to assert the email exists. If no email matches, a 404 error is returned. Secondly using the same logic as before to assert if the passwords match, if not then an error status of 400 is returned. If the password and email match, a JSON web token is created which contains the users ID and email thereby signing the user in, sending a status code of 200. The JSON web token is then configured to expire in one hour, after which the user will have to sign in again.

Back End sendEmail

The logic for sending an email from Echo's admin email to a newly registered user is handled in the function block below. The aforementioned is accomplished through the use of nodemailer library methods such as createTransport, which creates an object for sending the email i.e who it is sent by, mailOptions, which defines the details of the email to be sent, and finally sendMail, which sends the email with the prior methods filling out the details. This function is then called in the signup function block, passing in the email the user has signed up with providing it is a actual email.

Figure 4.15: Logic to send an email to signed up user

4.14.3 Back End auth.js middleware

The middleware interacts with the controllers to provide an extra layer of authorization to the application, thereby passing in the middleware to the user controller, then certain functions for example liking are only available to those who sign in or up to the application. The middleware uses the jwt.verify method to assert the custom token then setting the userID property of the req object to the id property of the decoded token. If it is not a custom token the function decodes the token

using the decode method and sets the userId property of the req object to the sub property of the decoded token. Lastly next is called to pass control to the user.js file in the controllers folder.

Figure 4.16: Logic in the auth.js middleware

4.15 Back End Models

As shown below, Mongoose is used to implement two schema that are used in the application. The schema is used when performing various CRUD operations. First and foremost, the schema function accepts various objects with a schema definition in order to establish the schema. Second, mongoose.model is used to create the collection in the database, with the collection name as the first argument and the schema as the second.

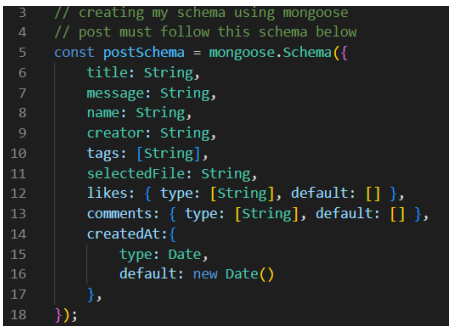

Figure 4.17: Post Message Model

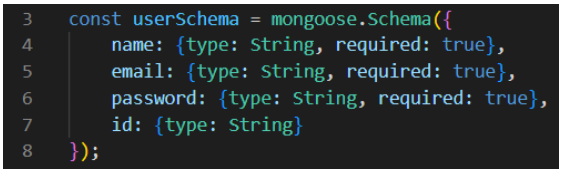

Figure 4.18: User Model

4.15.1 User Model

As previously stated, mongoose is imported into the script, and then the schema is initialized. The user must then follow the predefined schema in order to be saved properly in the database. This model in particular, is only concerned with the user signing in or up, as such the fields in this model are id, name, email, password, all of which a required fields except for id.

4.15.2 Post Message Model

Similarly to the previous model with some noticeable changes, the postMessage schema has more objects such as title, message, creator, tags, name, and selectedFile. Likes, comments, and createdAt are three distinct objects in the schema. Likes are an array of strings that are set to default, so that no post will have any likes when it is posted until a user actively likes it, at which point the like will be saved in the database. Comments work similarly to likes in that none appear in the database until a user leaves a comment on a post. Finally, the other distinct object createdAt has a field type, and date. When a user posts their echo, the default: new Date() function is used to give the exact date the post was made at, e.g. if the was made an hour ago, it will say thus, and so on.

4.16 Back End Routes

This section is concerned with the application's routes and the view section of the model view controller. The implemented scripts make use of express to handle the application's HTTP requests. The routes receive data from the controllers, then sending it to the user in a response. The majority of HTTP requests sent to the front end are get, post, patch, and delete.

```
7   const router = express.Router();
8
9   router.get('/', getPosts);
10  router.get('/search', getPostsBySearch);
11  router.post('/', auth, createPost);
12  router.get('/:id', getPost);
13  router.patch('/:id', auth);
14  router.delete('/:id', auth, deletePost);
15  router.patch('/:id/likePost', auth, likePost);
16  router.post('/:id/commentPost', auth, commentPost);
17
18
19  export default router;
```

Figure 4.19: Posts.js Routes

```
4
5   const router = express.Router();
6
7   router.post('/signin', signin);
8   router.post('/signup', signup);
9
10  export default router;
```

Figure 4.20: Users.js Routes

4.16.1 Posts.js

The applications most common response from the routing is the router.get('/', getPosts), this route handles GET requests, calls the getPosts function to retrieve a list of all posts as seen in 4.3. Second, router.post('/', auth, createPost) handles a post request to the URL, requiring middleware authentication, if the authentication is successful, the user can call the createPost function from the controller. Finally, delete('/:id', auth, deletePost) handles a delete request on a specific post, asserting that the specific user has the correct id to be able to delete the post they wish to remove, this route also requires authentication.

4.16.2 Users.js

Two routing methods are present in this script strictly relating to authentication, both being HTTP post methods. Firstly router.post('/signin', signin) handles the user signing into the application, then calling the signin function from the controller, this route is used to assert the credentials through the controller. Next router.post('/signup', signup) handles the user signing up to the application, then using the signup function in the controllers to create a new user with the provided credentials

4.17 Deployment

The final phase of the the design was to deploy the application onto the internet, this was done with two hosting sites, hostinger and heroku.

4.17.1 Heroku

The server side was deployed to heroku using the heroku CLI, then the package.json script for the server was changed from start: nodemon to start:"node index.js" which makes it easier to run as npm run start is no longer required every time, and the server does not need to be run locally anymore, now there can exist two environments, so as there is always a stable version of the server online. Secondly the index script was changed to give an alert message to assert the server side is running.

4.17.2 Hostinger

In order to link the server to the client, the below code was changed from localhost:5000, to the deployed heroku url. After that, using the node command npm run build to build the client side. SSL [56] must be installed on hostinger, which

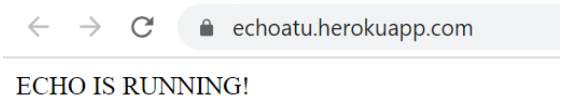

Figure 4.21: Heroku Deployment

```
const API = axios.create({ baseURL: 'https://echoatu.herokuapp.com/' });
```

Figure 4.22: Connection of server to client

provides the application with https security. Finally, once the application has completed the npm run build command, the build files must be dragged into the file manager on hostinger, and hostinger will build the clientside of the application, so that it can be accessed using the domain name specified when creating a website on hostinger. The application can then be seen on echoatu.com 4.3.

Chapter 5

System Evaluation

This section of the dissertation concerns evaluating the projects objectives set out in the introduction chapter 1, alongside the aforementioned, any limitations with the project, manual testing, and automated testing will also be discussed.

5.1 Cypress

The system uses end to end testing [57] in the cypress automated testing environment, which tests the application under real time user experiences. The below are the specs which were created for the application.

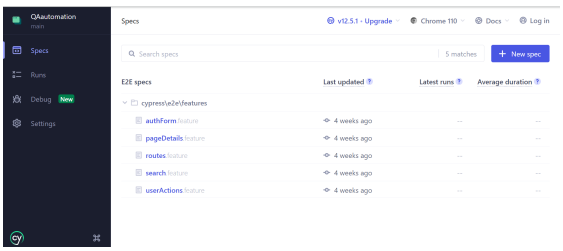

Figure 5.1: Specs for testing the application

5.2 Objectives

The following are the project's objectives, which are listed in the introduction:

- To create a dynamic single page application.
- The user can register their own account with the application.

- The user can sign into their own account once registered.
- A user is able to post pictures, they wish to share with their friends/family.
- Users can search for certain memories using taglines or memory titles.
- Users can send messages between each other.
- A user is allowed to like a picture, and comment underneath it.
- Users can create bio's underneath their posts.
- Users should be able to login to their profile and logout.
- Designing test scenarios to use in conjunction with cypress automated test tool.
- To have a fully automated test features to verify the application for commercial use, using test case scenarios designed.

The above will be analyzed below as to how well these objectives were completed.

5.2.1 To create a dynamic single page application

Manual testing revealed that the application is a single page application, only manual testing could be used at this level because it is difficult to determine unique ids that appear in urls such as the one below. The key advantage of creating a

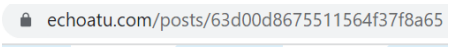

Figure 5.2: A dynamic id assigned to a post

single page application being the user does not have to re route multiple times to get to the page they wish to be on, also quicker loading times aiding the user experience [10].

Other Examples of Dynamic Routing

Other areas of the project where dynamic routing is used to make a single page application is through the below, the search bar, tags, and pagination.

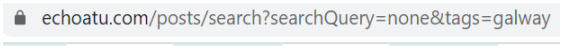

Figure 5.3: Searching in use Dynamically

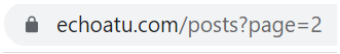

Figure 5.4: Paginate Dynamically through pages

5.2.2 The user can register their own account with the application

A user can manually create their own account by filling out the relevant form, or they can use Google's API for authorization. Manual and automated testing was recorded for manually signing up a user, but due to privacy restrictions, Google's API for signing up could only be tested manually.

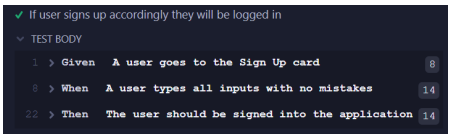

Figure 5.5: Specification for signing up a user manually passing

As shown above 5.5, cypress is used to assert that when a user fills out the relevant fields in the authorization form, that user will be able to sign into the application, and thus a user can register to the application achieving this goal. Google's API is known to pass, but the user must have a Google account in order to use the aforementioned API, so automated testing would be ineffective. After completing this aim, testing discovered that not only was the original goal met, but it was also expanded to include two ways for a user to register with the application.

Email Confirmation of sign in

As an addition to signing up with the application, a user will be notified following successful sign up with a welcome message from the admin email. Testing of this feature was only conducted manually, setting up a mock email, after which checking their emails to see if the sendEmail api was called, which sends an email from the admin email address following sign up. The aforementioned test was successful, automating this case would not be possible due to security issues.

5.2.3 The user can sign into their own account once registered

The preceding section discusses signing up for the application, similarly signing into the application can be accomplished by using Google's API or manually entering information into the sign in form.

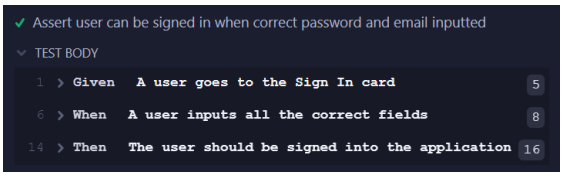

Figure 5.6: Specification for signing in a user manually passing

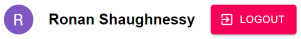

Figure 5.7: User then signed in

Manual and automated testing has shown, that the user is able to sign into their account once it is created. Manual testing was preformed on Googles API, and on the sign in form. Automated testing was only preformed on the sign in form, not on Googles API. This objective similar to the above was also expanded upon to included Googles API for multiple ways to sign in to the application.

5.2.4 A user is be able to post pictures, they wish to share with their friends/family

If the authorization conditions are met, the user should be able to create a post and share it with all application users. In the terms of this objective, the user may also post a picture they wish to share, though this is optional, as the user may prefer not to share a picture and instead express their thoughts on current events.

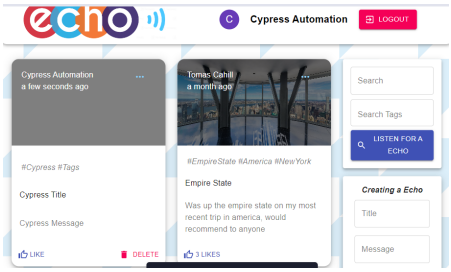

Figure 5.8: Visual representation of created post

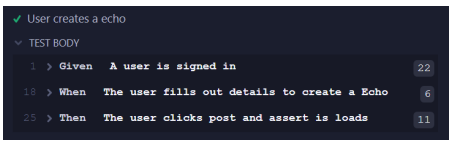

Figure 5.9: Automated test for user creating a post

From the above figure 5.8, it is shown the automated snapshot of running the specification from this figure 5.9. Due to limitations with the cypress testing tool, a picture could not be added to the post, however through manual behaviour testing preformed, it was found that a picture could be added when a file was chosen at the bottom of the creating a echo 4.5.

5.2.5 Users can search for certain memories using taglines or memory titles

Searching in the application has been shown to be done in two ways as the requirements laid out, searching for a particular title, which is shown in this figure using a dynamic search system 5.3. Moreover searching can be preformed through the use of tags 4.4.

Search Testing using Title

Carrying out automated testing on the search bar using the title of a post consisted of, inputting into the search bar what a user might search and then asserting that the right posts displayed on the screen, of which this test was found to be successful both manually and automated using cypress as can be seen below.

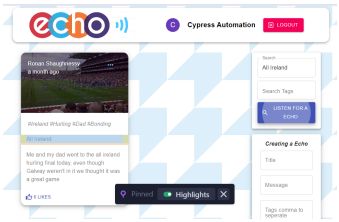

Figure 5.10: Search bar test

Figure 5.11: Using search bar test case

Search Testing using Tags

Manual and automated testing was also preformed on tagline searches, of which searching using tags manually can be seen here 4.4. The automated version of testing can be seen below both types of testing were successful, although an element of flakiness can be seen in an automated test using tagline, which was found be attributed to the page being dynamically loaded.

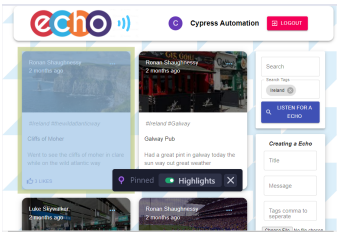

Figure 5.12: Asserting tagline working

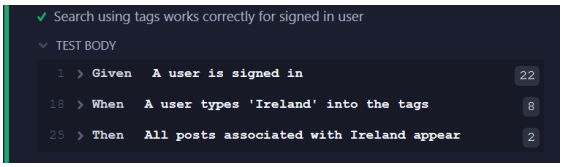

Figure 5.13: Tagline search test case

Search Testing using Tags and Title

When the searching feature of the application was used to search for a title, and tags, an assertion had to be made to ensure both related posts appeared for the user. This was evaluated through the use of manual, and automated testing. Manual testing revealed that this feature was complete, and that the associated posts appears when the title and tagline inputs were filled. Automated testing was also conducted, and the results of the automated test can be seen below.

Figure 5.14: Asserting tags and search title working together

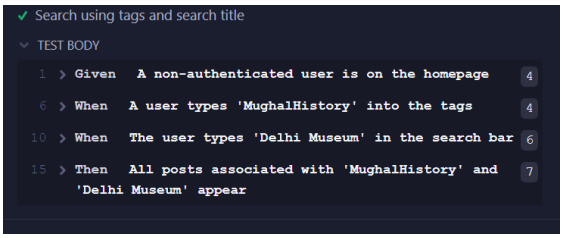

Figure 5.15: Tags and search title test case

5.2.6 Users can send messages between each other

This objective of the project was not reached therefore, this aim is a failed component of the application. Due to time constraints this is why this part of the system was not implemented however, future iterations of the application may include messaging between users.

5.2.7 A user is allowed to like a picture, and comment underneath it

This objective passed, and has seen extensive testing, both automated and manually. This objective assumes the user is signed into the application.

Liking a post

Cypress has shown through the use of behaviour test development, that a user can like a post. In the below figure it is shown using a snapshot Cypress has taken of the application. As a result of the user clicking the like button, Cypress asserts that the application has updated the text to say "You and " then the number of likes present on the post previously, the text "You and " will only appear once a user has clicked like if they have not done so already . Manual testing was conducted prior to automated testing to ensure that the application dynamically updates to say "You and", to assert that the signed in user has liked the post.

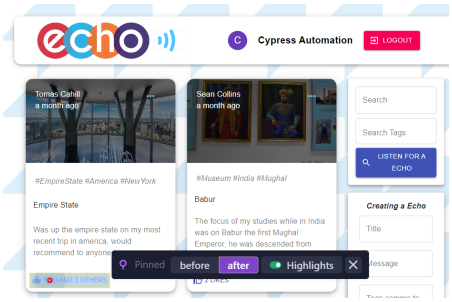

Figure 5.16: Liking a post when signed in

Figure 5.17: Test case for liking

Unliking a post

Succeeding the above test, Cypress continues to follow user behaviour as seen in this figure 5.17, by disliking the post after liking said post. This assertion checks that when a user clicks like again, the post will then revert to its previous from e.g 3 LIKES as shown below, and will no longer have the test "You and ".

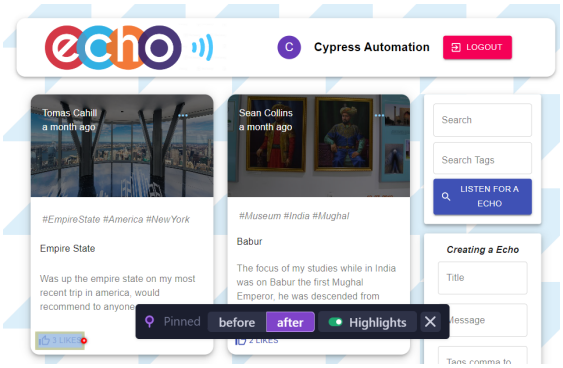

Figure 5.18: Unliking a post

Testing Comments Manually

The user is able to comment in a post provided they are signed in. Testing was carried out manually by clicking the ellipses in the top right, which brings the user to the details of the post, of which this post screen can be seen from this figure 4.10. After which to carry out the test the comment box was clicked and a comment was inputted then the comment button was clicked, following this the comment appeared on the post thereby achieving this objective.

Testing Comments using Cypress

Cypress was employed to automatically simulate this behaviour, therein asserting that the test case passed automatically and that the objective that a user can comment on a post was met. The pretense was to type the content in the comment input box, click comment, and then assert that the username displays on the left under the comments area next to the comment which the user made.

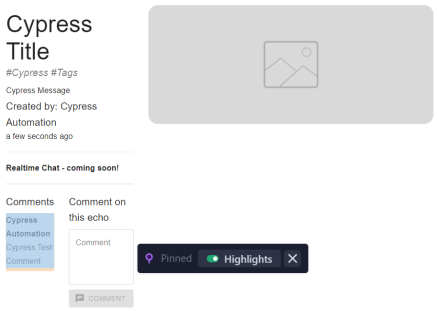

Figure 5.19: Cypress automating a comment

Figure 5.20: Test case for commenting on a post

5.2.8 Users can create bio's underneath their posts

Users can when creating their post as seen in this figure 4.5, make a bio or message to go with their post. While this objective has a limitation in that when the user is writing their message or bio, the message input box does not expand with the number of words the user is using, instead the input box remains the same length and width.

Bio Testing

Testing was conducted manually and automatically, only through manual testing was the above limitation found. However as seen below in cypress the user can create a bio under their post then create the aforementioned post.

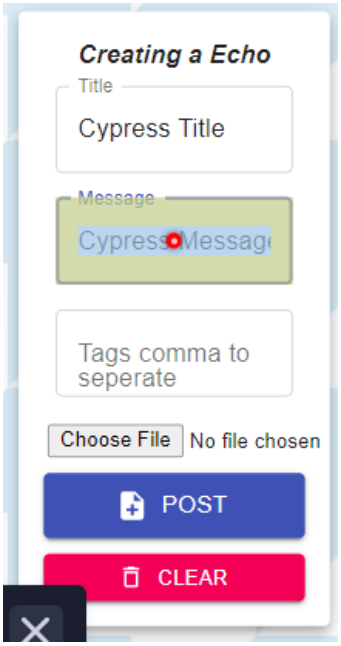

Figure 5.21: Cypress automatically creating a bio

5.2.9 Users should be able to login to their profile and logout

This objective tests the authorization of the application by asserting that a user can sign into the application, thereafter signing out if they wish.

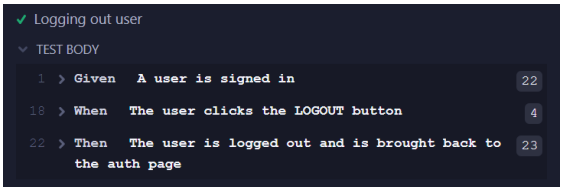

Figure 5.22: Test Feature for the above objective

Logging into a profile

Assertions were made to test the functionality of logging in manually and automatically. First manually testing, which was done by entering in the correct inputs, then asserting that the correct user was then signed into the application, which was found to have passed as the username appears in the navbar after signing in 5.23. Secondly the automation suite asserted that a user could be signed in, Cypress did this by asserting the correct DOM element for a username appears on screen as shown below.

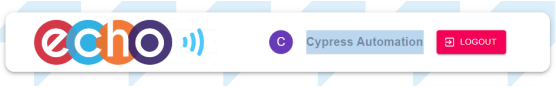

Figure 5.23: Cypress asserting a user can log into their profile

Logging out of a profile

The last two steps from the feature above are done to assert that a user can log out of their profile, thereby completing this objective 5.22. Firstly manual testing was preformed after coding was done, the finding was the logout button works, the user could log out of the application after they have signed in and have clicked the logout button. Finally, Cypress was used to take control of this test feature, automatically simulating the user's behavior. It was found to have passed as when Cypress clicks logout, the user is signed out and returned to the sign in form.

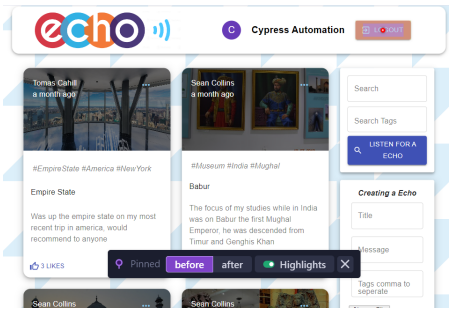

Figure 5.24: Simulating clicking logging out

Figure 5.25: After click assert sign in form appears

5.2.10 Designing test scenarios to use in conjunction with cypress automated test tool

To achieve this goal, research was conducted in order to verify the best way of implementing appropriate language for use in the test scenarios. To that end gherkin syntax was employed, in order for anyone to understand the test cases that were to be written, see this section on gherkin syntax, and how it can be used to work with cypress 3.6.3. Numerous test scenarios was created for the application, some of which can be seen in the above sections of this chapter, the core concept for designing these test scenarios is that anyone could understand what the test is doing, even those who do not have a software development background. These referrals are the gherkin syntax in visual studio code, 3.7 the first reference is the logic code which is used to assert the authentication system works appropriately, 3.8 the second is the plain English employed in the steps for a given scenario, the last reference is to how the gherkin format looks when it is being executed through the cypress environment 5.22. A full list of all test scenarios can be found on GitHub here.

5.2.11 To have a fully automated test features to verify the application for commercial use, using test case scenarios designed

Employing the above objective to support this aim, using gherkin syntax in cypress in order to format the test scenarios. Following completion of the above goal, TypeScript was used to develop the test assertions, e.g. assert that a users name appears when a user is signed in, this maintains that a user name will appear once a user is signed in, asserting the appropriate user is logged into the application.

Automated Testing Results				
Feature Name	Number of Test Cases	Pass	Failed	Flaky
authForm	9	6	0	3
pageDetails	4	4	0	0
routes	9	9	0	0
search	7	6	0	2
userActions	6	6	0	0

Table 5.1: Cypress Automated Testing Results

This objective was therein achieved, as the application has a fully automated test suite, allowing for changes to be made in the development of the application, then the automated tests can run against the changes made in development, in order to assert changes to the code base has not altered any other feature of the application. The below table is a result of the automated tests carried out on the application, a flaky test is a test which passes sometimes, then fails other times.

Flaky Tests

Three flaky tests were observed in the authorization feature authForm. Two of which were asserting a pop up appears on screen when the user has not filled out a relevant part of the sign in or up form. While both these tests do pass technically, it can only be seen to pass visually when the programmer visually sees the pop up appear, as it was hard to be able to assert a pop up using cypress. Secondly a flaky test was seen in signing a user up, the user signs up correctly when the spec is run for the first time, however if the spec is re run, the scenario will fail as the user is already created, this prompts the admin of the database to manually delete the user for MongoDB.

Solving Flaky Tests in the Future

Solving the pop up issue so the test does not have to rely the developer visually seeing the pop up, further code must be written and documentation studied on how to assert pop up boxes. Lastly, instead of the admin removing the user from the MongoDB database, there could be a way in cypress to use a different new email when the email is already in use thereby passing the test consistently, however this could crowd the database with null users if the spec is run often.

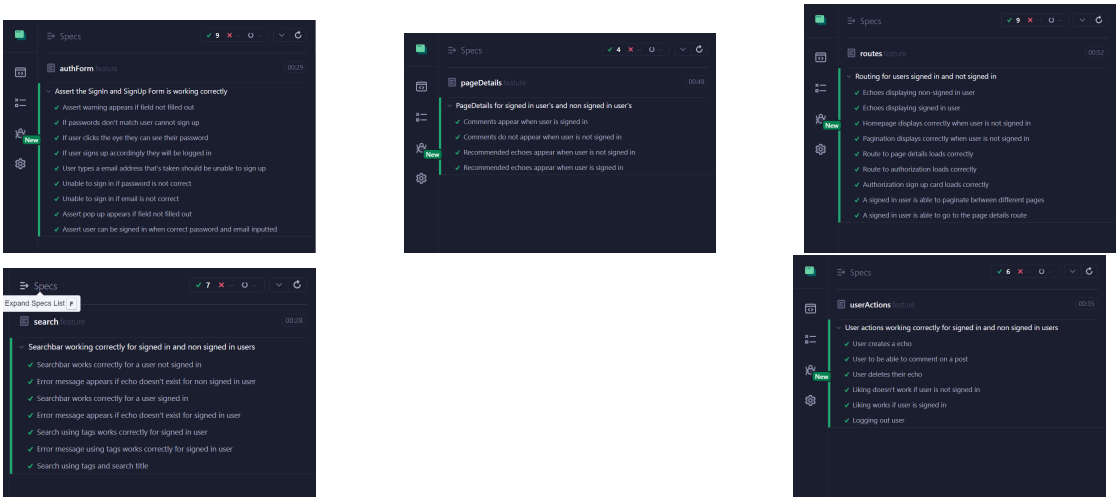

Figure 5.26: Five E2E Features

5.3 Automation Results In Cypress

The below are the visual results of the table above 5.1.

5.4 Limitations

While most objectives were met, creating a messaging system still eludes the application, perhaps in the future using front end technology a real time chat box will be implemented into the section donated for it in the page details component, this chat box would be linked into the page details, of who is active on a certain page detail e.g users on the New York page details post will be able to chat to other users, assuming they are present on the page at the same time, after which the conversation is over the application will automatically delete it. Other limitations include the section of flaky tests covered above 5.2.11. Additional limitations noted while testing the application are, to automated a spike test on the application in order to ascertain how many users the application can handle, develop a forgotten password system when the user forgets their password, and deploying the system in the app store for phone users rather than having it in the browser on a mobile.

Chapter 6

Conclusion

The main idea behind developing this application was to create a software environment similar to what I might encounter after finishing my degree, such as using agile technologies to build a full stack social media application, testing the application, and creating behavior driven environment so that anyone could understand how the application should work. The application is then deployed online, and finally, a highly appealing front end look and pleasing UI were created for the application.

6.1 Aims

The purpose of creating the application, what I expected to achieve during the development of the website, and did I met my personal aims.

- To build a highly pleasing front-end website, which will have a great appearance to please all who visit the application.
- To create multiple test scenarios, and have such scenarios fully automated using cypress.
- To improve my website design skills using React and JavaScript.
- To further my ability using automated testing.

6.1.1 Aim One

I felt I had created a stylish dynamic front end application that promotes ease of use and isn't overly cluttered, with what I consider to be a simple but elegant feel to the CSS and HTML placements of the cards, navigation bar, and form,

combined with a small amount of color in the logo and background, alluding to the application not being overly cluttered.

6.1.2 Aim Two

This is my first time using behavior driven development, and I believe I designed and created test scenarios that promote ease of reading and use. I believe this is how I will develop applications from now on, writing gherkin scenarios, and incorporating such features with the development of the application, allowing testing and development to occur concurrently. When combined with cypress, it helps me achieve my fourth aim.

6.1.3 Aim Three

Throughout the course of creating this application, I have gained a much greater understanding of using JavaScript and integrating functions in JS to be used between the front end and the back end. Additionally, I am much more comfortable using React as a front end architecture, especially using this technology in conjuncture with JSX, which provides the programmer ample more control over the UI.

6.1.4 Aim Four

Using automated testing is a real eye opener, such testing is superior, in terms of tracking metrics in which I can see what parts of the application are working as expected, and to view which sections of the application aren't working as expected. It was a great skill to pick up throughout the building of this application and I hope to carry it onward into a workplace environment using behaviour driven development.

6.2 Jira Roadmap

A comparison of my agile roadmap, how the project occurred during the course of the year versus how I thought the project would go in my Gantt chart 2.1.

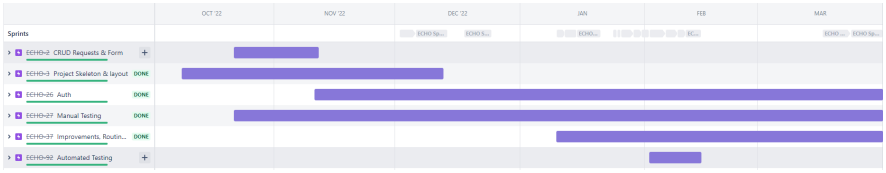

Figure 6.1: Agile Board

6.3 Other Learning's

Supplementary education I received during the course of creating the application.

6.3.1 Using JSON

I had already previously used JSON, however during the time spent on this project I used JSON Web Token(JWT) in order create a unique signature for a signed in user thereby allowing authorization to function properly.

6.3.2 Creating a dynamic single page application

Creating an SPA, which is a popular industry method for building applications that are quick and efficient.

6.3.3 Using REST

Learning more about creating a RESTful application, this would be used in conjuncture with JSON from above, and HTTP methods for declaring how the application should present the data.

6.3.4 Model View Controller

After studying the theory of MVC during my tenure at ATU, I was proud to be able to put it into practise in the back end of my project, it gave the back end substantial organisation when an addition to the application was made.

6.4 Succeeding in Objectives?

How well objectives were met was discussed in chapter 5. However, the findings from the evaluation chapter are as follows, the system works like a dynamic single page application, authorization, search, comments and creating posts were all achieved, however creating a chat environment was not accomplished, all of the above was validated through Cypress automated test suite, and manual testing, using behaviour driven test scenarios. Limitations of the application can be seen here 5.4, below is the possible future of the application.

6.5 The Applications Future

In the future I hope to add features such as having a forgotten password feature, machine learning to predict what the user wishes to search, having a real time chat in the post details section of the application, and putting the cypress tests into a pipeline like for instance Microsoft azure.

6.6 Credit

I would like to take this moment to thank my supervisor Martin Hynes who was a tremendous help throughout the project.

Chapter 7

Appendix

7.1 GitHub Link

The GitHub repository for ECHO, my final year project can be found by clicking here.

7.2 Installation Instructions

7.2.1 Prerequisites

Node.js and Visual Studio Code will have to be installed for interaction. The deployed application can be interacted online by clicking here, the testing suite must be used locally.

7.2.2 Installing Test Environment

1. Clone the GitHub repository to your local environment.

2. Go to the QAautomation folder, then start up a terminal environment.

3. Run npm install in the terminal.

4. After the above is finished, run npm run cypress:open, a localhost window will open, then click E2E Testing and choose a browser.

5. Click any feature you wish to test while in Spec.

7.2.3 Installing The Application Locally

1. Clone the GitHub repository to your local environment.
2. Go to the client and server folders, and run a terminal environment in both.
3. Run npm install in both terminals.
4. In each terminal run npm start, at which point the application will launch on your local environment.

Bibliography

[1] What is the mern stack? https://www.mongodb.com/mern-stack#:~:text=MERN. Accessed: 06-10-2022.

[2] What is crud? https://www.codecademy.com/article/what-is-crud. Accessed: 06-10-2022.

[3] Fatini Mobaraya and Shahid Ali. Technical analysis of selenium and cypress. 2019. https://www.researchgate.net/profile/Shahid-Ali-12/publication/338167875.

[4] Asma Akhtar, Birra Bakhtawar, and Samia Akhtar. Extreme programming vs scrum: A comparison of agile models. *International Journal of Technology, Innovation and Management (IJTIM)*, 2(2), 2022.

[5] Hisham M Abushama, Hanaa Altigani Alassam, and Fatin A Elhaj. The effect of test-driven development and behavior-driven development on project success factors: A systematic literature review based study. In *2020 International Conference on Computer, Control, Electrical, and Electronics Engineering (ICCCEEE)*, pages 1–9. IEEE, 2020. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9429593.

[6] 2022 stackoverflow developer survey. https://survey.stackoverflow.co/2022/#technology. Accessed: 23-02-2023.

[7] Adam Boduch and Roy Derks. *React and React Native: A complete hands-on guide to modern web and mobile development with React. js.* Packt Publishing Ltd, 2020. https://books.google.ie/books?hl=en&lr=&id=XCLhDwAAQBAJ&oi=fnd&pg=PP1&dq=explain+react.

[8] K Saundariya, M Abirami, Kumaran R Senthil, D Prabakaran, B Srimathi, and Govidan Nagarajan. Webapp service for booking handyman using mongodb, express js, react js, node js. In *2021 3rd International Conference on Signal Processing and Communication (ICPSC)*, pages 180–183. IEEE,

2021. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9451783.

[9] Jeremy Nelson. Developing sinopia's linked-data editor with react and redux. *Code4Lib Journal*, (45), 2019. https://journal.code4lib.org/articles/14598.

[10] Madhuri A Jadhav, Balkrishna R Sawant, and Anushree Deshmukh. Single page application using angularjs. *International Journal of Computer Science and Information Technologies*, 6(3):2876–2879, 2015. https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=4c272a54fad90a38e2a218c54653303f1203438d.

[11] Matthias Kevin Caspers. React and redux. *Rich Internet Applications w/HTML and Javascript*, 11, 2017. https://uol.de/f/2/dept/informatik/ag/svs/download/reader/reader-seminar-ws2016.pdf#page=14.

[12] David Holmstedt. Analyzing and implementing a third-party state machine library for friendlyreader and tecst, 2019. https://www.diva-portal.org/smash/get/diva2:1358096/FULLTEXT01.pdf.

[13] About node.js. https://nodejs.org/en/about/. Accessed: 24-02-2023.

[14] Prateek Rawat and Archana N Mahajan. Reactjs: A modern web development framework. *International Journal of Innovative Science and Research Technology*, 5(11):698–702, 2020. https://ijisrt.com/assets/upload/files/IJISRT20NOV485.pdf.

[15] Nimesh Chhetri. A comparative analysis of node. js (server-side javascript). 2016. https://repository.stcloudstate.edu/cgi/viewcontent.cgi?article=1004&context=csit_etds.

[16] Azat Mardan and Azat Mardan. Using express. js to create node. js web apps. *Practical Node. js: Building Real-World Scalable Web Apps*, pages 51–87, 2018. https://link.springer.com/chapter/10.1007/978-1-4842-3039-8_2.

[17] Chat Room. Express. js. https://devopedia.org/express-js.

[18] Writing middleware for use in express apps. https://expressjs.com/en/guide/writing-middleware.html. Accessed: 25-02-2023.

[19] What is rest. https://www.codecademy.com/article/what-is-rest. Accessed: 03-03-2023.

[20] Routing in express apps. https://expressjs.com/en/guide/routing.html. Accessed: 25-02-2023.

[21] Veronika Abramova and Jorge Bernardino. Nosql databases: Mongodb vs cassandra. In *Proceedings of the international C* conference on computer science and software engineering*, pages 14–22, 2013. http://web.cs.wpi.edu/~cs585/s17/StudentsPresentations/This%20Year/Week14/mongodb_vs_cassandra.pdf.

[22] Smita Agrawal, Jai Prakash Verma, Brijesh Mahidhariya, Nimesh Patel, and Atul Patel. Survey on mongodb: an open-source document database. *Database*, 1(2):4, 2015. https://d1wqtxts1xzle7.cloudfront.net/47783657/IJARET_06_12_001-libre.pdf?1470309477=&response-content-disposition.

[23] André Calçada and Jorge Bernardino. Evaluation of couchbase, couchdb and mongodb using osspal. In *KDIR*, pages 427–433, 2019. https://www.researchgate.net/profile/Andre-Calcada/publication/336226226_Evaluation_of_Couchbase_CouchDB_and_MongoDB_using_OSSpal.

[24] Cornelia Győrödi, Robert Győrödi, George Pecherle, and Andrada Olah. A comparative study: Mongodb vs. mysql. In *2015 13th International Conference on Engineering of Modern Electric Systems (EMES)*, pages 1–6. IEEE, 2015. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7158433.

[25] Javascript history. https://www.w3schools.com/js/js_history.asp#:~:text=JavaScript. Accessed: 26-02-2023.

[26] Paul Wilton. *Beginning JavaScript*. John Wiley & Sons, 2004. https://d1wqtxts1xzle7.cloudfront.net/42788648/Beginning_JavaScript__4th_Edition.pdf?1455774838=&response-content-disposition=inline.

[27] Gregor Richards, Sylvain Lebresne, Brian Burg, and Jan Vitek. An analysis of the dynamic behavior of javascript programs. In *Proceedings of the 31st ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 1–12, 2010. https://plg.uwaterloo.ca/~dynjs/pldi275-richards.pdf.

[28] Kevin J Theisen. Programming languages in chemistry: a review of html5/javascript. *Journal of Cheminformatics*, 11(1):11, 2019. https://link.springer.com/article/10.1186/s13321-019-0331-1.

[29] Boris Cherny. *Programming TypeScript: making your JavaScript applications scale*. O'Reilly Media, 2019. https://books.google.ie/books?hl=en&lr=&id=Y-mUDwAAQBAJ&oi=fnd&pg=PP1&dq=.

[30] Gavin Bierman, Martín Abadi, and Mads Torgersen. Understanding typescript. In *ECOOP 2014–Object-Oriented Programming: 28th European Conference, Uppsala, Sweden, July 28–August 1, 2014. Proceedings 28*, pages 257–281. Springer, 2014. https://users.soe.ucsc.edu/~abadi/Papers/FTS-submitted.pdf.

[31] Remo H Jansen, Vilic Vane, and Ivo Gabe De Wolff. *TypeScript: Modern JavaScript Development*. Packt Publishing Ltd, 2016. https://books.google.ie/books?hl=en&lr=&id=yc7cDgAAQBAJ&oi=fnd&pg=PP1&dqe.

[32] About jsx. https://reactjs.org/docs/introducing-jsx.html. Accessed: 27-02-2023.

[33] Jessica Minnick. *Responsive Web Design with HTML 5 & CSS*. Cengage Learning, 2020. https://books.google.ie/books?hl=en&lr=&id=Imj6DwAAQBAJ&oi=fnd&pg=PP1&dq.

[34] Pierre Geneves, Nabil Layaïda, and Vincent Quint. On the analysis of cascading style sheets. In *Proceedings of the 21st international conference on World Wide Web*, pages 809–818, 2012. https://hal.inria.fr/hal-00643075/file/RR-7808.pdf.

[35] Vy Nguyen Thanh. Building a serverless full-stack blog application using aws amplify. 2022. https://www.theseus.fi/bitstream/handle/10024/780986/Vy_NguyenThanh.pdf?sequence=2.

[36] What is json. https://www.oracle.com/in/database/what-is-json/#:~:text=It%20is%20a%20text%2Dbased,a%20server%20and%20web%20applications. Accessed: 02-03-2023.

[37] What is a web token. https://jwt.io/introduction. Accessed: 02-03-2023.

[38] Michael Jones, John Bradley, and Nat Sakimura. Json web token (jwt). Technical report, 2015. https://www.rfc-editor.org/rfc/rfc7519.html.

[39] Why cypress. https://docs.cypress.io/guides/overview/why-cypress. Accessed: 28-02-2023.

[40] Malika Tasnim Taky. Automated testing with cypress. 2021. https://www.theseus.fi/bitstream/handle/10024/495908/taky%20malika%20final%20thesis%20ready%20to%20submit.pdf?sequence=2.

[41] Elis Pelivani, Adrian Besimi, and Betim Cico. A comparative study of ui testing framework. In *2022 11th Mediterranean Conference on Embedded Computing (MECO)*, pages 1–5. IEEE, 2022. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9797165.

[42] Why should you switch to cypress for modern web testing? https://dzone.com/articles/why-should-you-switch-to-cypress-for-modern-web-te?fromrel=true. Accessed: 28-02-2023.

[43] Most used programming languages among developers worldwide as of 2022. https://www.statista.com/statistics/793628/worldwide-developer-survey-most-used-languages/. Accessed: 28-02-2023.

[44] What is cucumber? https://cucumber.io/docs/guides/overview/. Accessed: 01-03-2023.

[45] cypress-cucumber-preprocessor. https://www.npmjs.com/package/@badeball/cypress-cucumber-preprocessor. Accessed: 08-03-2023.

[46] Cypress plugins. https://docs.cypress.io/plugins. Accessed: 08-03-2023.

[47] Getting started with cypress 10 and cucumber. https://blog.emumba.com/getting-started-with-cypress-10-and-cucumber-6b43ff68633b. Accessed: 08-03-2023.

[48] John Fisher, D Koning, and AP Ludwigsen. Utilizing atlassian jira for large-scale software development management. Technical report, Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States), 2013. https://www.osti.gov/servlets/purl/1097750.

[49] Ondřej Havazík and Petra Pavlíčková. How to design agile game for education purposes in jira. In *2020 7th International Conference on Control, Decision and Information Technologies (CoDIT)*, volume 1, pages 331–334. IEEE, 2020. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9263937.

[50] Rita Marques, Miguel Mira da Silva, and Diogo R Ferreira. Assessing agile software development processes with process mining: A case study. In *2018 IEEE 20th Conference on Business Informatics (CBI)*, volume 1, pages 109–118. IEEE, 2018. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8452664.

[51] How to set up a ai bot with kommunicate. https://www.kommunicate.io/blog/integrate-chatbot-in-react-js/. Accessed: 15-04-2023.

[52] Eric Keller and Jennifer Rexford. The" platform as a service" model for networking. *INM/WREN*, 10:95–108, 2010. https://www.usenix.org/legacy/events/inmwren10/tech/full_papers/Keller.pdf.

[53] Why heroku. https://www.heroku.com/what. Accessed: 30-01-2023.

[54] Patrik Danielsson, Tom Postema, and Hussan Munir. Heroku-based innovative platform for web-based deployment in product development at axis. *Ieee Access*, 9:10805–10819, 2021. https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=9317772.

[55] Hostinger features. https://www.hostinger.com/. Accessed: 28-01-2023.

[56] Kipp Hickman and Taher Elgamal. The ssl protocol. 1995. http://www.webstart.com/jed/papers/HRM/references/ssl.html.

[57] What is end to end testing. https://smartbear.com/learn/automated-testing/what-is-end-to-end-testing/#:~:text=End%2Dto%2Dend%20testing%20is,like%20from%20start%20to%20finish. Accessed: 03-02-2023.