A social and learning application - Quizzer

Stephen Kelly

B.Sc.(Hons) in Software Development

April 19, 2022

Final Year Project

Advised by: Dr Joseph Corr
Department of Computer Science and Applied Physics
Galway-Mayo Institute of Technology (GMIT)

Contents

About this project

After years of studying and researching technology I wanted to build an application that has a positive outcome for its users. Recent trends in social media causing mental health issues inspired me to create a positive social application based on learning and sharing knowledge with another.

This document describes the system design, architecture, and developer decisions. I have taken the challenge to use new and emerging technologies, to develop this application. My application ”Quizzer” is for all ages, more specifically for anyone who enjoys learning information, interacting with others, and taking fun quizzes.

This is a full stack application and is consisted of a three layer architecture design. The technologies to make this possible are Python, React, and PostgreSQL database. I am taking on this project individually using the waterfall methodology for development.

The focus of this application is to serve as a social and learning ground, providing a community of users. Users can create servers and invite their friends, and have discussions in said servers. Users also have the ability to take a trivia quiz of their liking for learning interactivity. All of the code and documentation can be viewed in the GitHub repository.

Authors This project was developed by Stephen Kelly (G00361283) 4th year student G.M.I.T

Acknowledgements

I would like to thank my project supervisor Dr. Joseph Corr for guiding me throughout development of my project dissertation.

Chapter 1

Introduction

1.1 Project Aims

The aim for my project is to build a functioning web application through new and emerging technologies. The application is suited for all ages, specifically designed for those who enjoy learning and interacting with others. Below are the main objectives for my project:

- Develop a functioning full stack application.
- Learn and use the Python language
- Apply the waterfall methodology
- Create an application programming interface
- Save data from the application to database
- Research emerging technologies

Over the past few years of studying Software Development, I have gathered insightful ideas to create and develop a social and learning platform. There seems to be no current platforms or website that servers as a social hub with interactive learning. Having done extensive research on social medias and e-learning quiz applications I gathered the information to design my own social and learning application.

1.2 Project Specification

Below is a list of features that are included in the application:

- The user must be able to register
- The user must be able to log in and out
- The user must be able to setup and take quizzes
- The user must be able to create chat servers
- The user must be able to send messages
- The user must be able to upload images
- The user must be able to explore servers
- Users must be able to send invites to the server
- Application data stored on a database

1.3 Chapters Analyzed

Below is a brief description of the contents for each section of the document.

- **Introduction 1.0.1 :** The context is provided for my project, scope, objectives and purpose of project.
- **Methodology 2.0.1 :** I described my development approach, planning and research for the project.
- **Technology Review 3.0.1 :** I reviewed emerging and useful technologies for my project.
- **System Design 4.0.1:** I explained the implementation and architecture of the system built.
- **System Evaluation 5.0.1 :** I discussed testing, evaluating the specifications and if my scope was reached.
- **Conclusion 6.0.1 :** I summarised the context and objective of the project in hindsight, discussing any future implementations.
- **References 7.0.1 :** A list of referenced content used while composing my dissertation.

1.4 Github Repository Overview

All project files are available on GitHub via this link[1]

- **https://github.com/Stephen313k/ProjectDissertation**

1. **quizzer** Folder contains the full application
2. **screencast-pdf** Folder contains demo of application and dissertation in PDF format.

1.5 Project Context

From the beginning of the first semester we were told to brainstorm ideas for our project dissertation. I began to explore new technologies I previously had not used, as I would like to learn a new technology that is used in professional environments. I researched blogs and posts about uprising technologies, and tried to compose ideas for an application that is unique. After the first meeting with my project supervisor I better understood what was needed for a successful level 8 software development project. By working on a full scale application it's another addition to my portfolio to further prepare me to enter a professional software development environment.

1.6 Project Research

1.7 What is a social media platform?

Social media is an application that facilitates sharing ideas, information and images through servers and networks to communities and people across the globe. The five main types of social medias can be categorised as:

- Social Networks (Facebook, LinkedIn)
- Social News (Reddit)
- Micro-blogging (Twitter)
- Bookmarking Sites (Pintrest)
- Media Sharing (Youtube)

1.8 Why use social media?

Social media can be used as a platform to share knowledge in your field of expertise with others and gain credibility amongst a community. You can also view what others have posted in topics that interest you, to gain insight from others who have completed similar research and analysis. Social media has become the most popular way people get in contact with another, instant online discussions can build communities with like minded individuals who have the same interests, questions, and hobbies. [2]

1.8.1 What is an online learning platform?

A digital learning platform is a web-space for educational content that offer its users the resources they need to educate or participate in academic activities. Lectures, quizzes and related topic information is used for research and learning. Learning platforms are becoming increasingly popular especially after the Covid-19 pandemic with stay at home learning becoming the standard. Popular learning platforms are:

- Skillshare
- Moodle
- Codeacademy
- Udemy

1.9 SOAR Analysis

A SOAR Analysis is a framework for identifying strengths, opportunities, aspirations and results. SOAR strives to be a forward-thinking model to address the potential of a project. Using this model helped me narrow my scope and ultimately choose my main technologies. Through many hours of research I have decided that developing a Python-Django and React application would suit my needs best. The primary reasons are:

- React was designed for Facebook
- Discord (Social media with chat servers) is written in Python
- React is used for responsive applications
- Django has an active and helpful community
- Loading time with Django and React is very fast
- Django is a complete web development framework
- Python is one of the fastest growing lagnuages

The results of my research in the SOAR analytic framework for my consists of:
Strengths - I have developed previous e-commerce websites and have familiarity with building web applications. The concept of a learning and social

platform is a unique idea that has yet to be explored.

Opportunities - This project has the opportunity to prepare me for work environments and build up my portfolio. There is a recent trend in e-learning application through the Covid-19 pandemic, this is a good time to build a learning application.

Aspirations I hope to build on-top of this project in the future with advanced features and try to gather an active community. I hope this project can make a positive impact on society by building a friendly and informative community.

Result I have a specific scope and end-goal set-out with my project supervisor to determine if I am on track to deliver an end-product. Translating the success of a project can be difficult, however building a fully functional three stack architecture application is the standard I have set.

1.10 Exploring Surveys

While trying to scope out my project I explored many surveys to get an idea on the popularity of social medias sites.

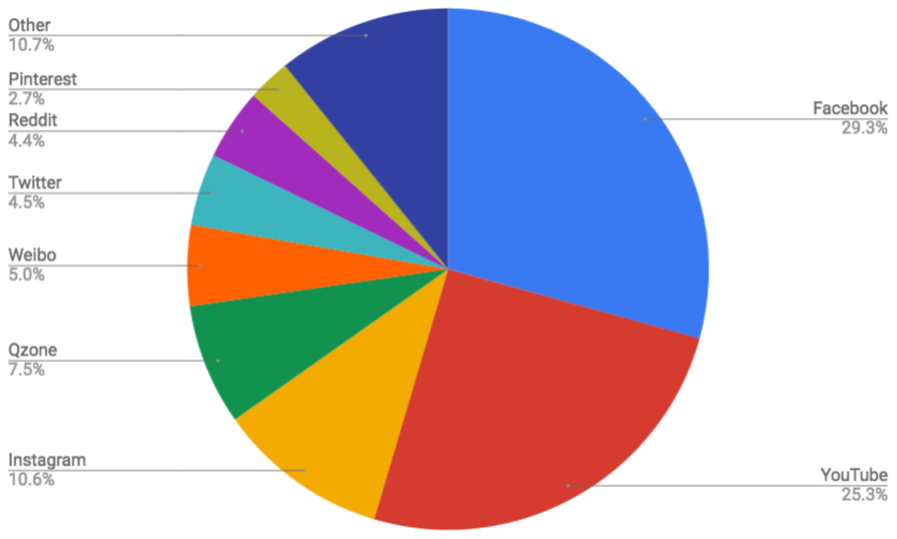


Figure 1.1: What is the most popular social media?

It is important to note that YouTube is the second most popular social media, YouTube contains a vast amount of educational content to it's users.

Not only does social media have a plethora of users, but the screen-time can also be excessive in most cases. Below is a survey on how many hours users think they spend on social media per day.

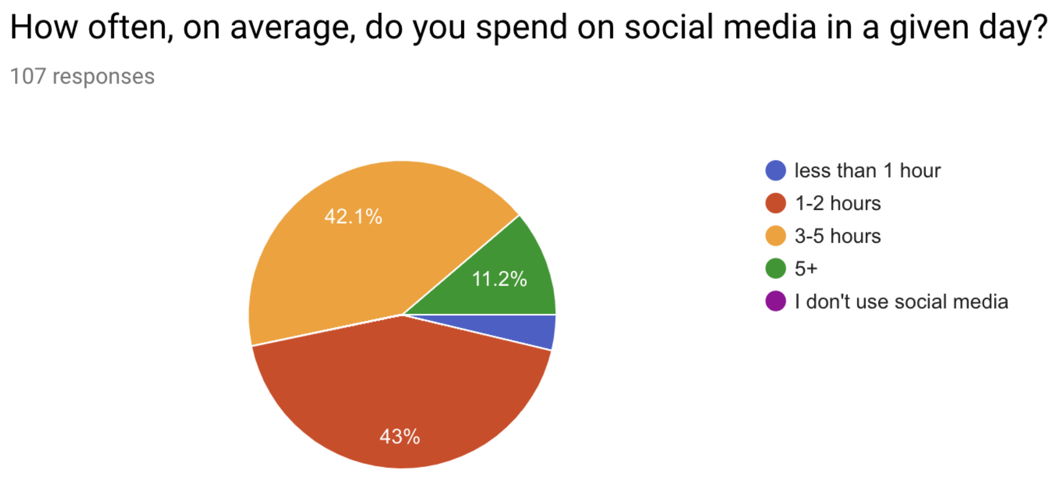


Figure 1.2: How long do you spend on social media per day?

As seen in the pie chart there is a tie between users who engage in social media up to five hours and those who engage up to three hours. Three to five hours everyday is an excessive amount of time spent. Since users are spending so much time on social media is important to research the type of content are the users are viewing. As seen below social media mainly consists of pictures and peoples opinions. There is a lack of educational content on social media, this is a good reason to start a social learning platform.

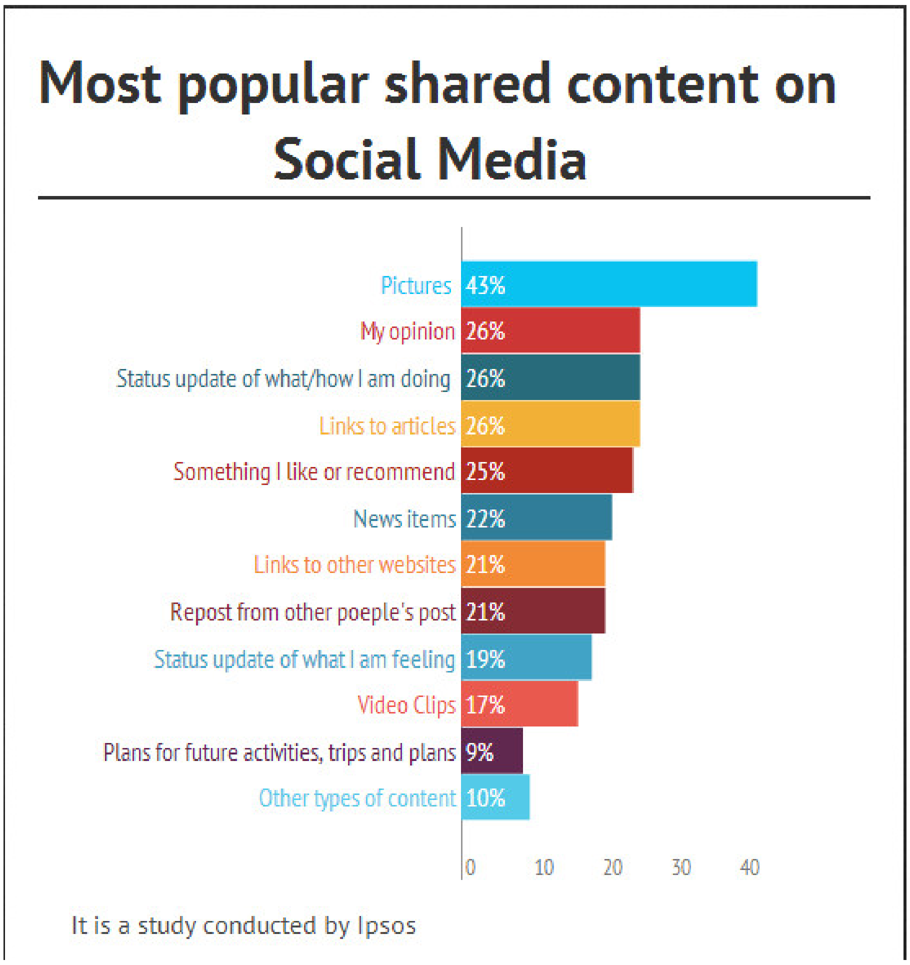


Figure 1.3: Shared content on social media

It is essential to understand what type of content the public want. The survey conducted below illustrates that thirty three percent of people wish there were more learning and educational content online. It is now clear that there is also a demand for learning applications.

Chapter 2

Methodology

Types of development methodologies were researched and analyzed prior to coding to see which development methodology would suit my project best. Prior to coding I designed a Gantt chart outlining my project components. As I am fully responsible for the whole project time management and planning is essential.

2.1 Version Control

Version control system is a tool used for managing source code typically during development of an application; or when an update is applied from its developers. GitHub is one of the best and most commonly used version control service that is utilised everywhere. GitHub was outlined in the brief to be used for the project version control system. It provides a detailed history of your commits, by using Git Tracking you can to see how the project was developed. Git is capable to handle small or large scale applications, it has a tool-kit based design, and is very fast and efficient in performance. Git allows you to change to a previous version of the application, called a rollback. A team of developers can work asynchronously on a project by using GitHub, many companies use this tool and it is essential to learn before heading into any work environment. There are two types of version control. [3]

Centralized

On centralized version control there is a single ”central” copy of your project on a server and all the commits are applied to that central copy. When pulling you are given the files you need but you cannot obtain a full copy of your project locally. An issue with centralized version control is if the copy of the project goes down or becomes corrupt, the developers are unable to access the code, or retrieve previous versions. An example of a centralized version control system is Subversion (SVN).

Distributed

For distributed control systems you don't have to rely on a single central version to store all the versions of your files. You can clone a copy of your repository locally so you have the all of the project files. Distributed systems do not have a single point of failure, teams are able to use any developers clone as a backup and retrieve it. There is little risk of losing your project using distributed version control. A popular distributes control system is Git.

Key benefits to use version control include:

- Trace-ability
- Document history
- Management overview
- Branching and merging

2.2 Development Methodologies

During the life-cycle of an application, it will undergo a specific type of development process, or methodology. There are different types of software methodologies to choose from to build an application. Bellow are four software development methodologies that are most commonly used. There is no one correct model as they all have their own advantages and disadvantages, varying greatly in development style.

- Waterfall
- Agile
- Feature-Driven Development
- Scrum

After extensive research I decided my project would be developed using the waterfall method. As there is no client for this project, I have a clear understanding of the goals that I have set. Without having to adjust to client feedback during development the waterfall model is the best method to approach as it focuses on the end goal in early development with precise planning. As seen below, the waterfall method is a sequential development process that flows to the next step upon completion. Due to it's linear process it is essential to understand the project scope as there is little flexibility with this methodology. [4]

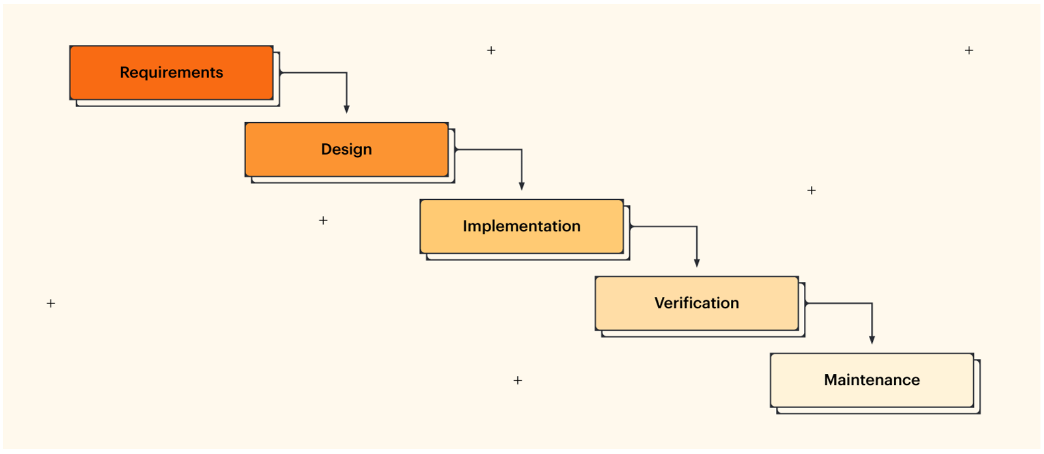


Figure 2.1: Waterfall Methodology

The most important phase of the waterfall model is the design phase. The key aspect in this phase is to gather all the requirements needed for the project, scoping out project deliverables and describing project functionality. This allows the project to be planned before development and gives an insight if the project scope is too large, or not up to standard. The concise planning makes starting the code less intimidating as you have a specific plan to follow.

2.3 Planning

For time management and scheduling of development I made and used a Gantt chart to follow by. The waterfall methodology is often visualized in the form of a Gantt chart, as it shows linear progression and each task in the chart is set up as steps. In the waterfall method each step is cascaded into the next step. In a Gantt chart you can see the previous phase "fall" into the next phase. I broke down the Gantt chart into three sections, one for each tier of my application. Inside each tier are the functions or objectives I have defined in my scope and project requirements.

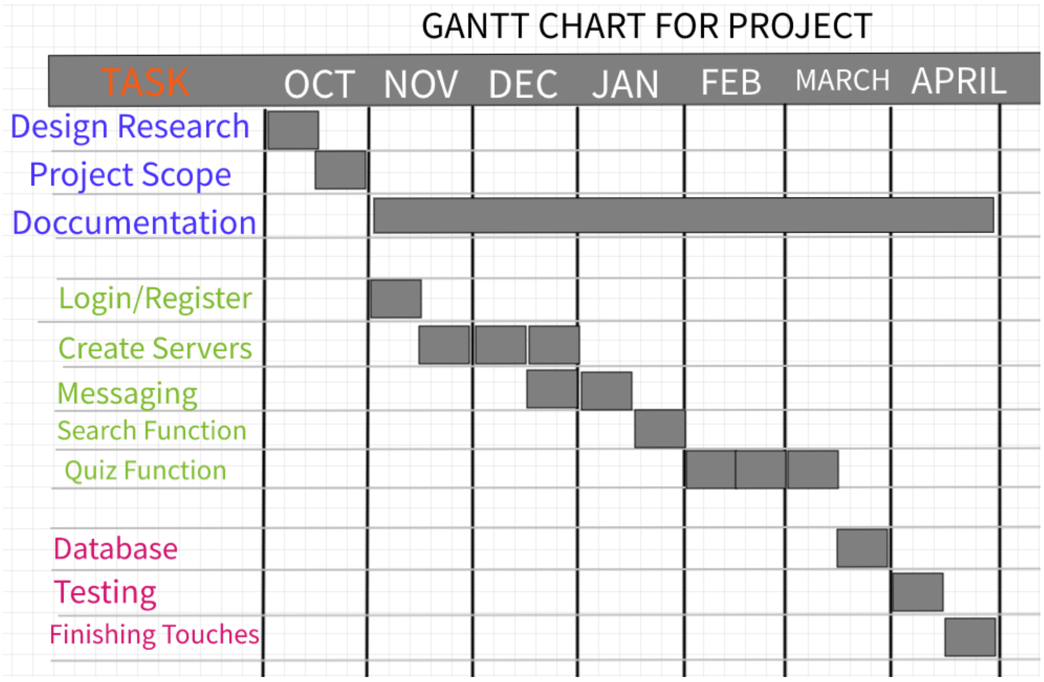


Figure 2.2: Gantt chart for planning the project

2.4 Setting Milestones

By implementing the waterfall method I set milestones for my project development, this is essential for the waterfall method as it sets specific goals to the rigid development process. During development when I reach a milestone I updated my GitHub repository and provided a demo in my weekly meeting with my project supervisor. By setting milestones I am able to focus on obtaining particular goals in my application without sidetracking into developing different features. If I am not reaching milestones at an efficient rate it would become clear to me early on and the scope can be redacted or adjusted. Below you can see the milestones i set out for myself prior to development.

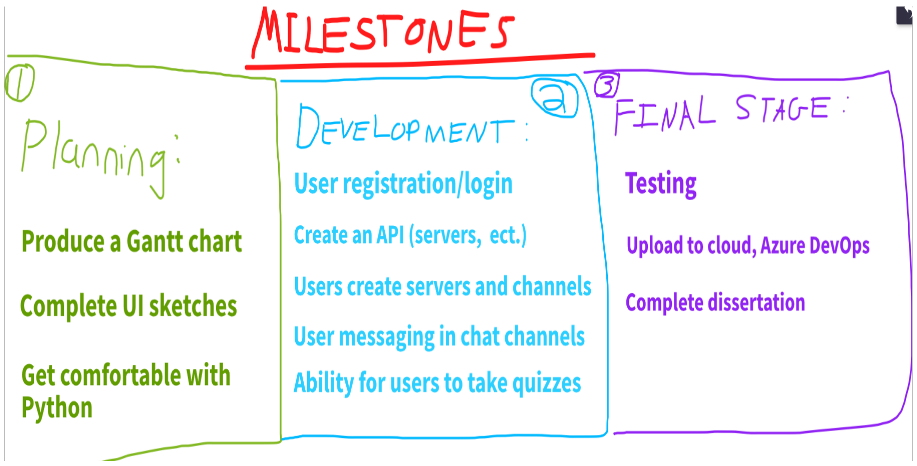

MILESTONES
① Planning:
Produce a Gantt chart
Complete UI sketches
Get comfortable with Python
② DEVELOPMENT:
User registration/login
Create an API (servers, ect.)
Users create servers and channels
User messaging in chat channels
Ability for users to take quizzes
③ FINAL STAGE:
Testing
Upload to cloud, Azure DevOps
Complete dissertation

Figure 2.3: Milestones

2.5 Development Environment

The project was developed using Visual Studio Code. This software is lightweight and easy to run, it features an interactive debugger so you can step through source code, inspect variables and execute commands in the console. VS Code has support for Git, in-fact it even has a consolidated view of all the details in your repository. This serves as a useful environment for me as I am required to use GitHub. Python in Visual Studio Code provides a lot of capabilities with the Microsoft Python extension.

2.6 Meetings

Me and my supervisor organized meetings via Microsoft Teams as on-campus attendance is being regulated due to COVID-19. It was important to be prepared and attend each week as I demonstrated my work and gained insightful feedback on my design and development. These meetings helped me observe my work from a different point of view as I had no team members. Meeting discussions consisted of:

- Does the project meet level 8 requirements
- Technology use and suggestions
- General feedback on the project
- Set goals for each week (demos, etc)

- Am I following the development schedule

The most important aspect was I treated these meetings that my supervisor was a client and I was showcasing them their product, on a weekly basis. I wanted to show sufficient work every meeting, this helped encourage my work ethic and kept me consistent in developing.

2.7 Testing

There are a variety of different approaches to testing. Testing is essential to ensure your application is running correctly, and to help create well built application. Black box and White box testing are the primary types of testing.

Black Box Testing

Black Box Testing is a testing technique where the functionalities are tested without having any knowledge of the code structure or development details. Black Box Testing focuses on the input and output of application and is entirely based on software requirements and specifications. Types of Black box testing include:

- Functional testing (tests for functions, by developers)
- Non-functional testing (tests performance, scalability, usability)
- Regression testing (check if new code updates caused issues)

White Box Testing

White Box Testing is a testing technique where the internal structure and code are tested to verify flow of input and output data to improve the usability, design and security of the given function. [5] White Box Testing tests code for the following:

- Security issues
- Expected output
- The flow of code
- Testing statements, objects, functions individually

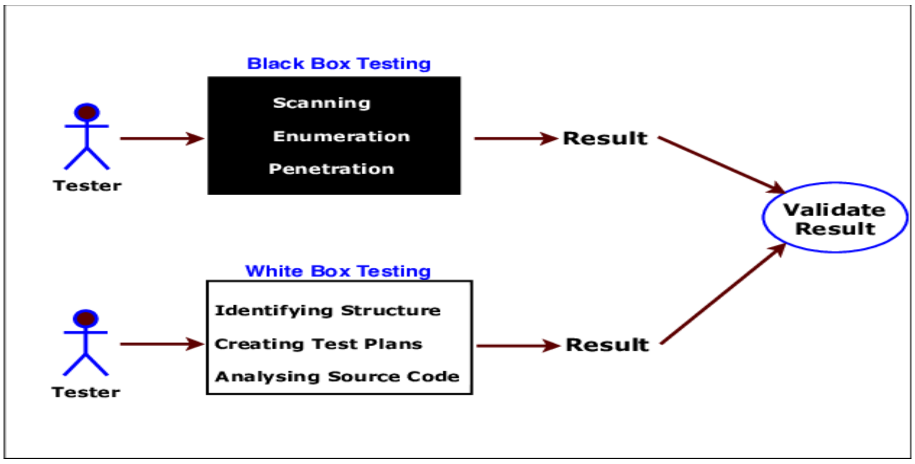


Figure 2.4: Black box and White box testing

2.7.1 Unit Testing

Unit testing is a type of software testing that tests its individual components. Unit tests are able to isolate a section of code and verify if that function is working correctly. Unit tests may consist of functions, methods, modules or an object. Unit testing is a White Box testing technique performed by the developer. Benefits of utilizing unit tests include:

- Helps fix bugs
- Unit tests help with code re-use
- Unit tests provide project documentation
- Ensure code meets quality standards
- Ultimately saves time

2.7.2 Test Coverage

Test coverage is able to measure the amount of testing performed by a set of tests. Test coverage gathers the information about which part of the program are executed when running the test to determine if the statements have been called. With this technique we can view how much code is being covered by tests.

- Assure quality of the test
- Determine which parts of application are not tested
- Prevent defect leakage ()

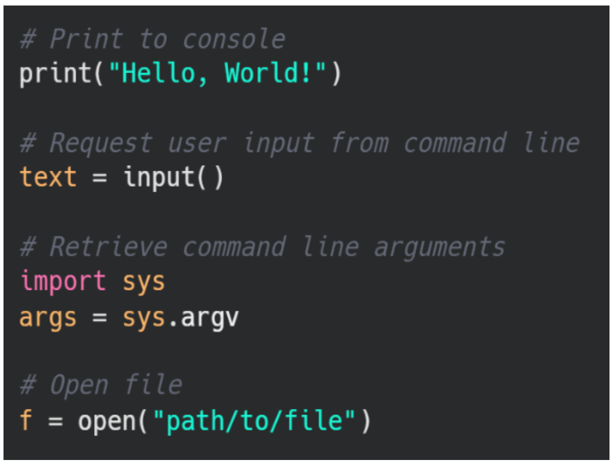
Unit testing and test coverage is used in my application.

Chapter 3

Technology Review

3.1 Python

While investigating efficient programming languages for web development, it became clear to me that Python is my best choice. This would be my first project using the Python language, meaning I would have to teach myself. It is a robust programming language with clear and easy to read syntax, making the learning curve less intense. Python is not to be viewed as just a beginner programming language as it has a plethora of libraries and powerful web frameworks. Python saves developers time by presenting solutions that would otherwise have to be built manually, this serves well for rapid development of an application. Python can run slow compared to other languages as the source code is interpreted during run time rather than being compiled to native code, making it run slower. Python can be paired with a powerful web framework Django to build web-application. Below are code snippets of sample syntax in Python.



```python
# Print to console
print("Hello, World!")

# Request user input from command line
text = input()

# Retrieve command line arguments
import sys
args = sys.argv

# Open file
f = open("path/to/file")
```

I will be using Python for the back-end of my application. The back-end refers organizes and saves data, it contains the code that allows for functionality of an application. The back-end uses logic and methods to create functions of the application. The user cannot access the back-end as it plays an important role behind the scenes.

3.1.1 Django

Django Rest framework (DRF) is the most popular and arguably most powerful full-stack framework for Python and is written in Python. Django is easy to work with, it has plenty features as it's described as "Batteries included" [6]. I am using Django as my web-framework for rapid development, as Django includes:

- User authentication
- Default Admin panel
- Model view controller
- Object Relational Mapper
- Unit testing framework
- CRUD (create, read, update and delete) interface.

Model View Template

Django utilises a Model View Template architecture, this design pattern contains three essential components: Model, View and Template. Each of these layers have different responsibilities.

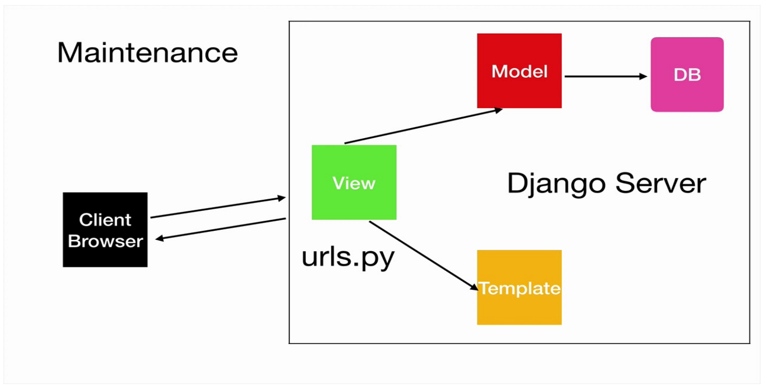


Figure 3.1: Model View Template

Model

The Model is a data access layer which contains the required fields and attributes of the data being stored. The Model helps handle the database. A Model is essentially a class in Python that uses an application programming interface (API) to communicate between the other layers. Models in Django offer CRUD opperations, this is so developers can easily manipulate the data currently stored in the database.

View

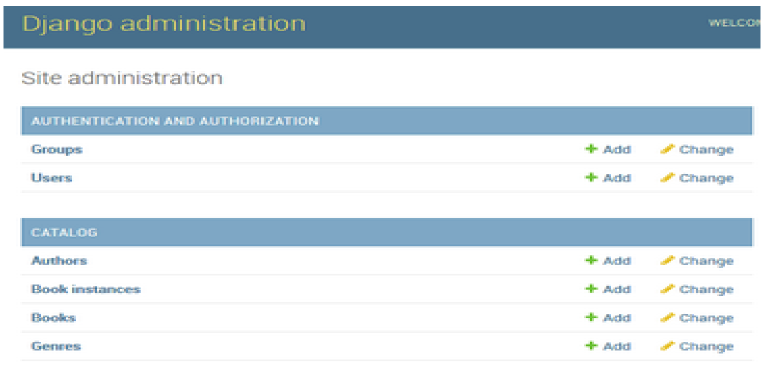
Views are Python classes that retrieve a web request and return a web response. The response can be a HTML template response or a redirect response that directs the user to another page. Views hold the logic to return information as a response to the user.

Template

The Template refers to the display and user interface of the page that is seen by the user. Templates usually HTML code and styling of sorts for aesthetic design. The contents of a template can be dynamic or static, a template does not have any logic in it and is only used to present data.

CRUD Interface

CRUD operations in Django refer to creating, updating, and deleting operation on the database. Django offers a graphic user interface on the administrator panel to modify data by using ”click and select”. This allows for easy handling on a relational database.


Django administration WELCOM

Site administration

AUTHENTICATION AND AUTHORIZATION		
Groups	+ Add	Change
Users	+ Add	Change

CATALOG		
Authors	+ Add	Change
Book instances	+ Add	Change
Books	+ Add	Change
Genres	+ Add	Change

Figure 3.2: Django admin panel

3.2 React.js

React.js is an open-source JavaScript library that is used to create a user interface for web applications. React solves the issue of temporal data changes, meaning the data in the application is not static, and is updated in real-time. React allows the display of data that is continuously changing as users engage or update information. React works as a component-based framework, a component can be used anywhere in the application. This is what makes react so unique. These design principles encourage the DRY (Don't repeat yourself) principal and reusable code. React components have a life-cycle of four phases:

- Initialization
- Mounting
- Updating
- Unmounting

Initialization is where the component is created, typically using props in the parameter of a constructor method. *Mounting* refers to when components are inserted into the DOM (Document Object Model) this is how the data is represented and accessed on a web-page. In this phase the component is rendered for the first time. The *updating phase* is as described in the name, this is where the component's state changes, and re-rending of updated information occurs. Finally there is *unmounting*, this is when the component gets unmounted and is removed from the DOM. [7]

I will be using the React JavaScript library to create user interfaces, route my application, and display dynamic data.

Figure 3.3: React

3.3 PostgreSQL

PostgreSQL database uses the SQL programming language to create and operate data in a relational database. It is important to learn how to query data using SQL commands in terminal. The database stores information on data tables, which are connected and have fixed data fields. The schema in SQL is composed of field types and tables. These tables and relationships must be designed before any logic is implemented in the program as it is difficult to implement schema changes. However, using the waterfall methodology schemes and tables are to be designed before any type of implementation.I will be using PostgreSQL with pgAdmin4 for my database technology.[8]

pgAdmin4

While PostgreSQL is a SQL database it offers a GUI tool used to manage and interact with the database. pgAdmin4 is a web-based GUI tool that visualises the already created Postgre database. The graphical interface simplifies the creation, maintenance and use of database objects. This tool can perform any sort of administration action related to the database. The dashboard tab shows analysis and activity statistics for the database [9]. As Quizzer is a social platform it is important to track server activity to see if there is a community growing. The server activity panel displays information about:

- Sessions (Handling and information of active sessions/users on the database)
- Locks (When multiple users update the same data at the same time the database is locked to receive input from the first user instead)
- Prepared Transactions

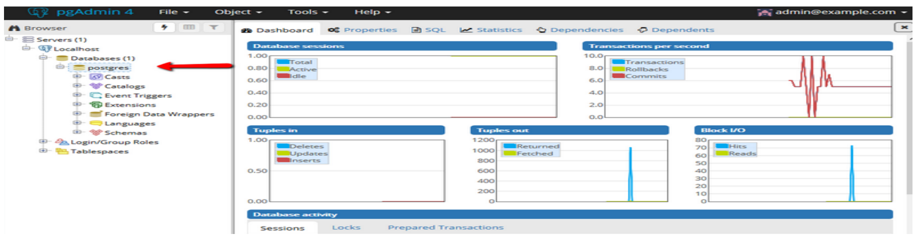


Figure 3.4: pgAdmin

3.4 Postman

Postman is a cloud-based application that allows developers to test API's utilizing a graphical interface. It has built in tools that support every stage of the API life-cycle. Postman supports automated testing, by creating test suites you are able to re-run them over and over again. Automated testing prevents error on the developers side and streamlines testing. Types of tests include:

- Unit tests
- Functional tests
- Integration tests
- Regression tests

Postman helps you save time when building an API it supports request tracking information. For every HTTP response in postman there are several status codes returned specifying if it was a successful request, empty response, unauthorized access and many more. After receiving a specific error message and status code it makes it easier to solve the issue, or to confirm the request is working properly. I will be using Postman to test my API endpoints.

Figure 3.5: Postman

3.5 Microsoft Azure

Microsoft Azure allows you to build, deploy, and manage applications without having to buy or maintain the underlying infrastructure. The Azure service has quick responsiveness and ensures consistent high availability to users. I have specifically chosen Azure as it has heavy integration with GitHub and Visual Studio Code. The security that azure has to offer is very appealing, it has developed the leading force in security processes. The Security Development Lifecycle(SDL) secures all private data to its core; services to the cloud remain secure and protected. Azure is one of the most trusted cloud platforms available and it's currently free for 12 months. Pros for using Microsoft Azure include:

- High Availability (5 hours of downtime per year)
- Data Security
- Scalability
- Cost-Effective

3.6 Visual Studio Code

Visual Studio Code is a free software to edit and manage source code. Developed by Microsoft this application is available to Windows, Linux and MacOS users.Visual Studio supports hundreds of languages, it provides syntax highlighting, bracket-matching, auto-indentation, box-selection, snippets and more. Visual Studio fully supports Python and contains a Git repository window, this displays details in your repository and the URL.

Visual Studio has an in-built debugger making the development more fluids as it maintains a single view with the code and debugger. This helps save time when navigating through code trying to find the error. There are tones of extensions on the market place available for the Visual Studio, this increases developer productivity as custom debuggers, ssh-terminals and DevOps pipeline connections become available. With the correct settings and extensions Visual Studio becomes more than an editor, but a powerful tool for productive workflow.

Figure 3.6: VS Code

3.7 LaTeX Overleaf

To compose my dissertation I used LaTeX. LaTeX is a software system for document preparation, it offers precise control over the formatting of documents. LaTeX is used for high-quality documentation, text in LaTeX documents look refined and polished because of it's typesetting algorithms to determine optimal layout of text and images. You can simply access a LaTeX editor online by the Overleaf service. Overleaf allows you to utilise a LaTeX editor in your browser. I had previously used Overleaf and was impressed with how fast it is to setup or import a document. Overleaf contains an auto-save feature which can be enabled or disabled. As LaTeX is not as intuitive to use compared to Microsoft Word, Overleaf provides documentation for all of the features. The learning curve for LaTeX is steeper compared to other editors, however it is worth spending the extra time to obtain a more professional end product.[10]

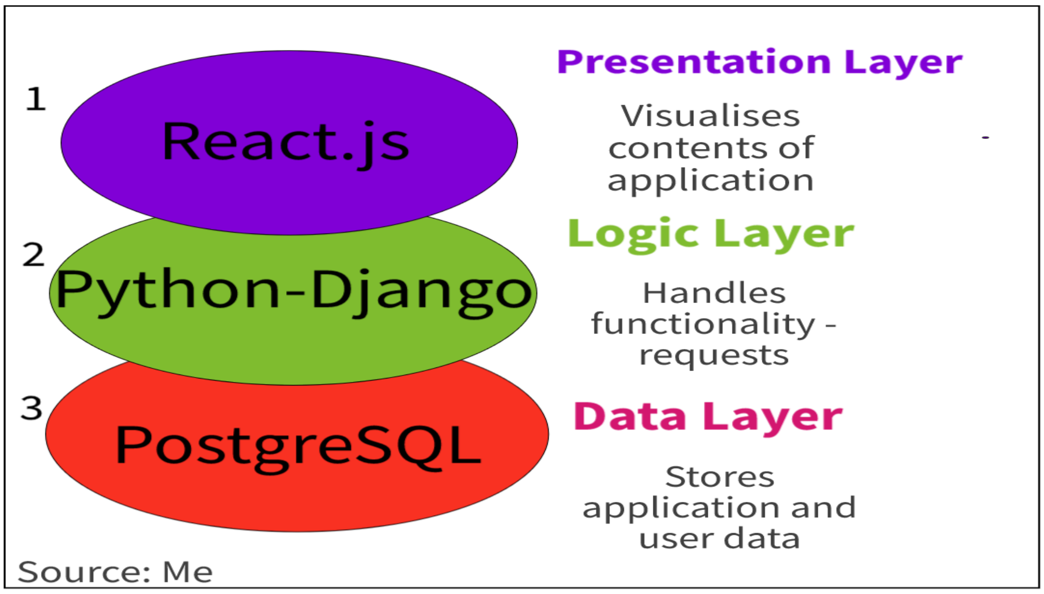
Figure 3.7: Overleaf

Chapter 4

System Design

4.1 Application Overview

My application is built using the three tier architecture design. The three main layers can be seen in the diagram below.The presentation tier involves the user interface and interaction. The Logic tier is where all of my models are built and calculations and commands are ran, it also transfers data to the two layers on either side. The Data tier stores and receives information from a database. The three main pieces of technology I am using is React, Python and PostegreSQL.



1
React.js
Presentation Layer
Visualises contents of application
2
Python-Django
Logic Layer
Handles functionality - requests
3
PostgreSQL
Data Layer
Stores application and user data
Source: Me

Figure 4.1: Three tier design

The diagram below is an API UML I created to help visualize the relationships and models I have created for the back-end API using Django.

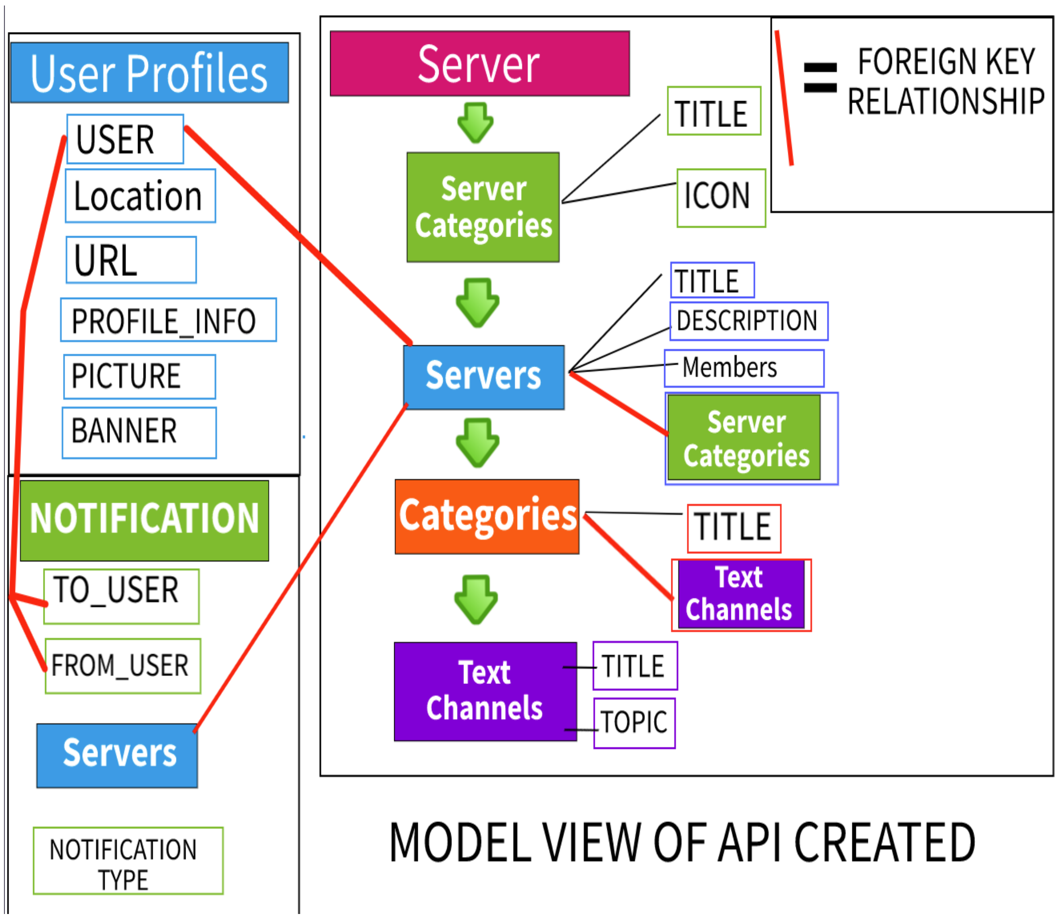


Figure 4.2: Visualise the model layout and relationships

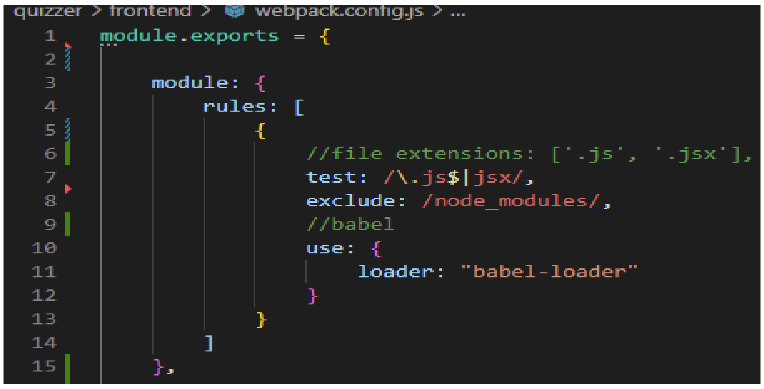
4.2 Functionality

Listed below are components and features in my application I designed. I will explain the system design by going through each section.

- Project setup
- Login/register
- Homepage layout
- Server functionality
- Messaging
- Notifications
- Quiz
- Database

4.3 Babel

Babel is a toolchain that is mainly used to convert code into a backwards compatible version of JavaScript in current and older browsers or environments. This allows for fix of certain bugs that may occur based on the browser version you are developing on. Before I began any code I configured and installed Babel; running ”npm install –save-dev @babel/core @babel/cli” in the shell. I created a configuration file in webpack.config.js so my project can read in JS files and JSX files, I had to specify to use the transpiler. Below on line 10 I identified the transpiler loader babel and set it as my default.

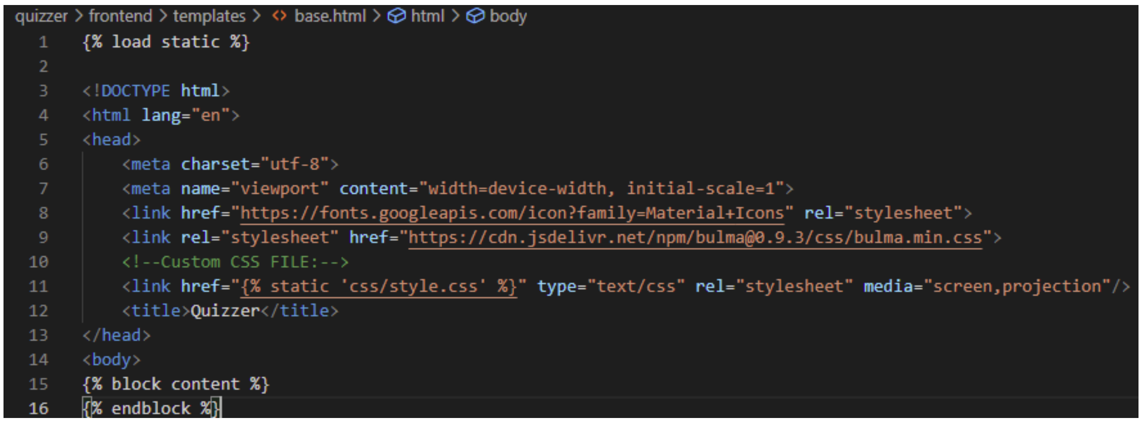


```
quizzer > frontend > webpack.config.js > ...
 1  module.exports = {
 2
 3      module: {
 4          rules: [
 5              {
 6                  //file extensions: ['.js', '.jsx'],
 7                  test: /\.js$|jsx/,
 8                  exclude: /node_modules/,
 9                  //babel
10                  use: {
11                      loader: "babel-loader"
12                  }
13              }
14          ]
15      },
```

Figure 4.3: Babel webpack.config.js

4.3.1 Styling

Bulma is used for styling my front-end components. I use Bulma for design on my page header, footer and main pages. I chose to use an off-white colour for my navigation bar and page background. This makes the coloured buttons in my website ”pop” more. I imported the complete Bulma CSS class in my base.html file, as it is the main styling framework for my application.



```
quizzer > frontend > templates > base.html > html > body
{% load static %}

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link href="https://fonts.googleapis.com/icon?family=Material+Icons" rel="stylesheet">
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bulma@0.9.3/css/bulma.min.css">
    <!--Custom CSS FILE:-->
    <link href="{% static 'css/style.css' %}" type="text/css" rel="stylesheet" media="screen,projection"/>
    <title>Quizzer</title>
</head>
<body>
{% block content %}
{% endblock %}
```

Figure 4.4: Import Bulma base.html

4.4 Login

The user login and registration system uses Django-Knox, this is a framework that makes the authentication of the API endpoints easier to handle. Knox authentication is token based and allows users to sign in the same account on multiple devices. Upon entering the application, the user can login using their email and password associated to the account. Users are directed to the login page if they are not signed in.

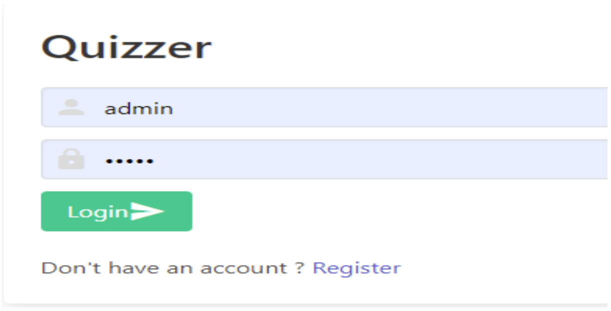
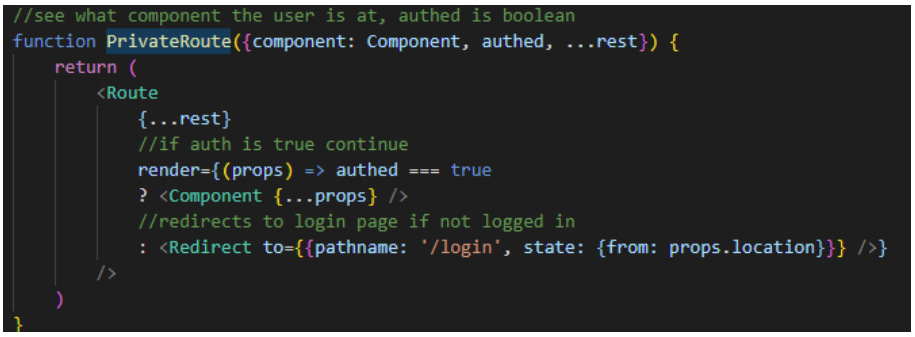


Figure 4.5: Login

Routing to login

In order to define if the user is not logged in and needs to be re-directed to the login page a private route function is setup. The function takes which component the user is attempting to access, if they are authorized (logged in) they will continue to the next component. If the user is not authorized they are re-directed to the login page.

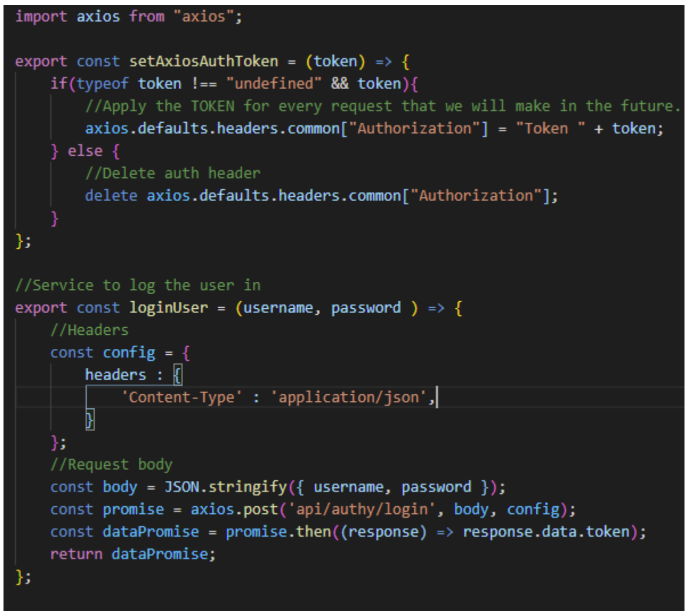


```
//see what component the user is at, authed is boolean
function PrivateRoute({component: Component, authed, ...rest}) {
    return (
        <Route
            {...rest}
            //if auth is true continue
            render={(props) => authed === true
            ? <Component {...props} />
            //redirects to login page if not logged in
            : <Redirect to={{pathname: '/login', state: {from: props.location}}} />}
        />
    )
}
```

Figure 4.6: Routing

Authentication

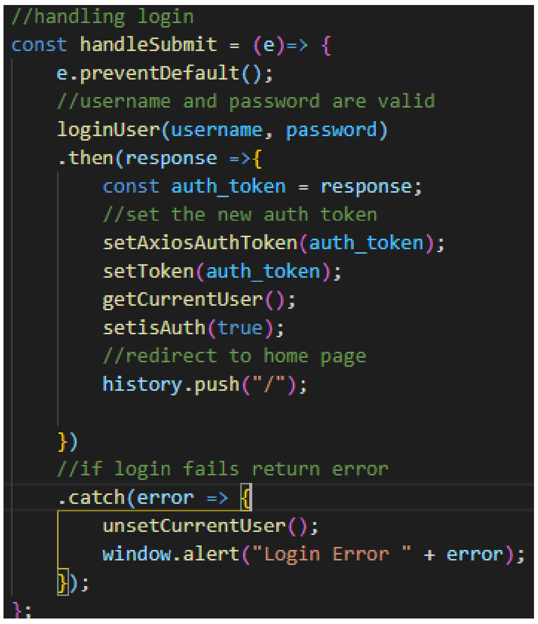
Below are methods to get the user token for authentication when singing up or logging in an account. Post requests to the API are handled using axios.



```
import axios from "axios";

export const setAxiosAuthToken = (token) => {
    if(typeof token !== "undefined" && token){
        //Apply the TOKEN for every request that we will make in the future.
        axios.defaults.headers.common["Authorization"] = "Token " + token;
    } else {
        //Delete auth header
        delete axios.defaults.headers.common["Authorization"];
    }
};

//Service to log the user in
export const loginUser = (username, password ) => {
    //Headers
    const config = {
        headers : {
            'Content-Type' : 'application/json',
        }
    };
    //Request body
    const body = JSON.stringify({ username, password });
    const promise = axios.post('api/authy/login', body, config);
    const dataPromise = promise.then((response) => response.data.token);
    return dataPromise;
};
```

Figure 4.7: Authentication

These methods created are used for the Login component to handle a submission request when a user attempts to login. If the username and password matches an account on the database a new authentication a token will be set for the user they are then logged in and redirected to the home page.

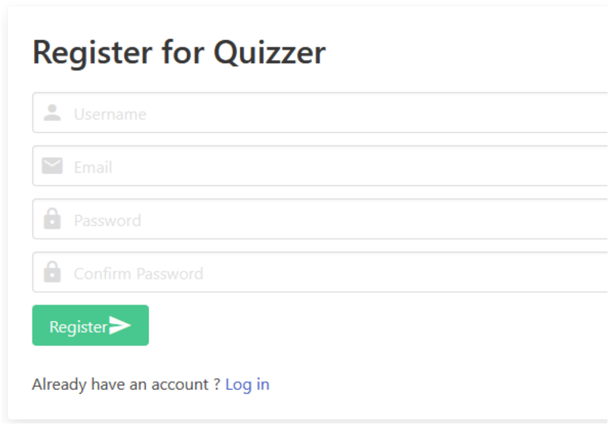


```
//handling login
const handleSubmit = (e)=> {
    e.preventDefault();
    //username and password are valid
    loginUser(username, password)
    .then(response =>{
        const auth_token = response;
        //set the new auth token
        setAxiosAuthToken(auth_token);
        setToken(auth_token);
        getCurrentUser();
        setisAuth(true);
        //redirect to home page
        history.push("/");

    })
    //if login fails return error
    .catch(error => {
        unsetCurrentUser();
        window.alert("Login Error " + error);
    });
};
```

Figure 4.8: Register

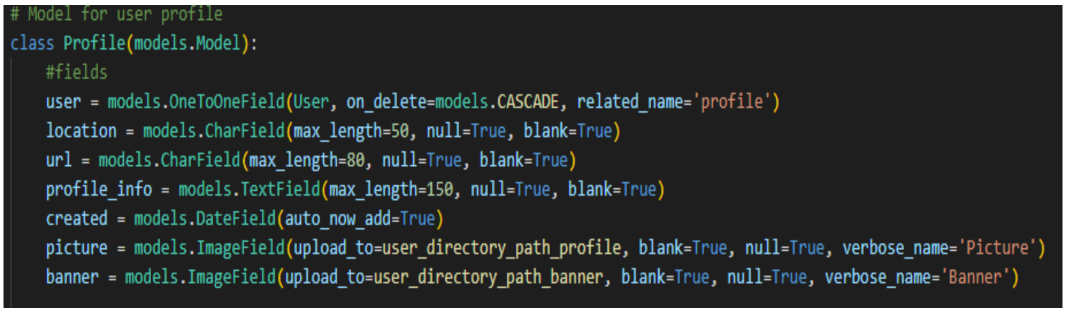
4.5 Register

If a user does not have an account they are able to register using a new username, email and password. This is done in the same fashion as the login methods using axios requests to the API for authentication and the request is handled in the register component. Users must use an unused email and matching passwords to register.



Model

A Profile model is created to define data of for all users and which values it should contain. Using a Profile model helps with consistency of all users throughout the application, and can be linked with other tables (server, chat, and notifications) to manipulate and send data when logged in. Using this model they are able to edit their username, profile information, profile picture and profile banner image.

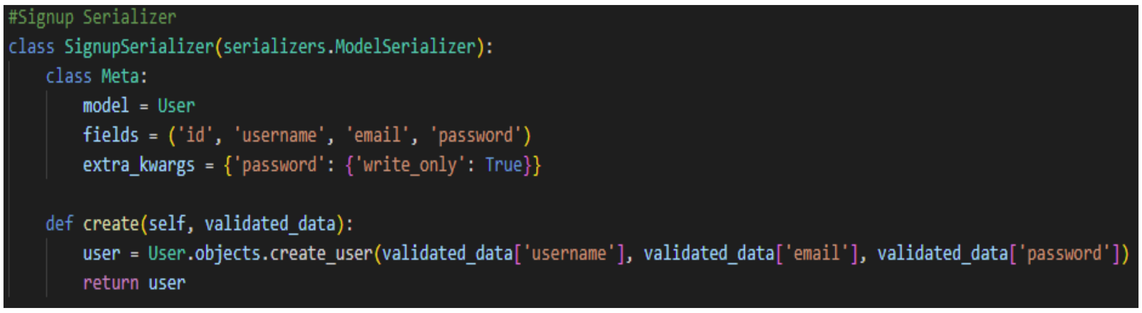


```
# Model for user profile
class Profile(models.Model):
    #fields
    user = models.OneToOneField(User, on_delete=models.CASCADE, related_name='profile')
    location = models.CharField(max_length=50, null=True, blank=True)
    url = models.CharField(max_length=80, null=True, blank=True)
    profile_info = models.TextField(max_length=150, null=True, blank=True)
    created = models.DateField(auto_now_add=True)
    picture = models.ImageField(upload_to=user_directory_path_profile, blank=True, null=True, verbose_name='Picture')
    banner = models.ImageField(upload_to=user_directory_path_banner, blank=True, null=True, verbose_name='Banner')
```

Figure 4.9: User model

Serializers

Serializers are responsible for converting objects into data that is able to understood by the JavaScript in my application. New users will be inputting raw data into the register fields, this data needs to be serialized and is eventually converted into JSON form for development. The user profile is serialized along with the input for registration.



```python
#Signup Serializer
class SignupSerializer(serializers.ModelSerializer):
    class Meta:
        model = User
        fields = ('id', 'username', 'email', 'password')
        extra_kwargs = {'password': {'write_only': True}}

    def create(self, validated_data):
        user = User.objects.create_user(validated_data['username'], validated_data['email'], validated_data['password'])
        return user
```

Figure 4.10: Signup Serializer

4.6 Homepage

Once signed in the user lands on the homepage displaying the quiz feature, navigation bar and their joined servers. The website technically has one navigation bar, and a div container on the left side which form an upside down "L" shape, a diagram is shown below. The top navigation bar has the Quizzer logo which redirects to home when clicked, a logout and notification button. The second navigation feature on the left contains images of active servers the user has joined, and an option to create or explore servers.

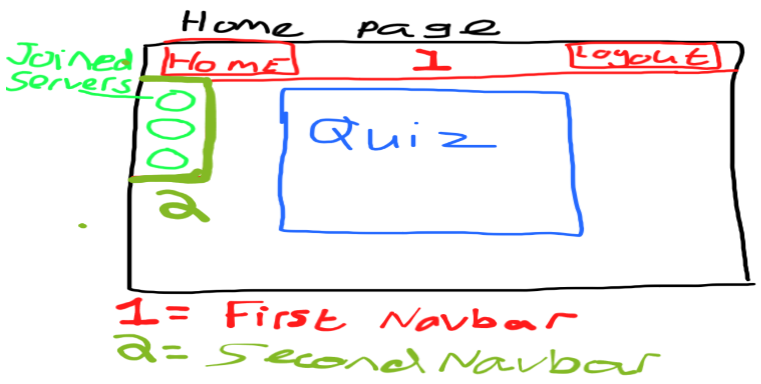
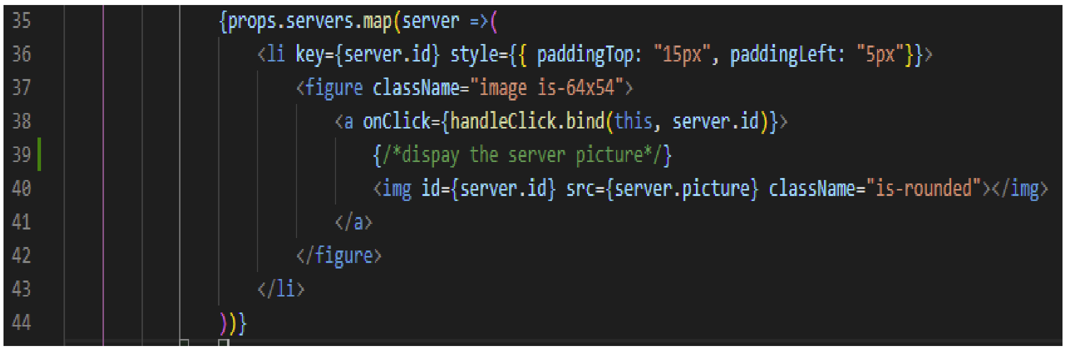


Figure 4.11: Home page sketch

Server List Panel

A list of joined servers is displayed on left of the homepage to make it easy for users to open server details and enter the room. To achieve this I created a function that acts as a container and used props to map through all the servers the user is associated with. They are displayed in a list and handle an onSubmit request when clicked. The user is then redirected to the server.

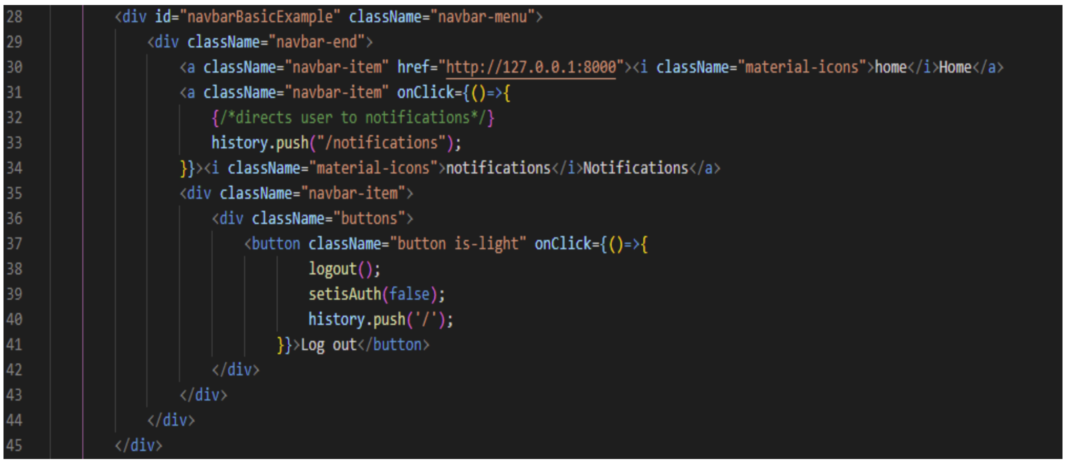


```
{props.servers.map(server =>(
    <li key={server.id} style={{ paddingTop: "15px", paddingLeft: "5px"}}>
        <figure className="image is-64x54">
            <a onClick={handleClick.bind(this, server.id)}>
                {/*dispay the server picture*/}
                <img id={server.id} src={server.picture} className="is-rounded"></img>
            </a>
        </figure>
    </li>
))}
```

Figure 4.12: Server List

Navigation Bar

The navigation bar contains three options the user can interact with when clicked. The website logo redirects the user to the homepage. The notification button to see if users have sent you an invite to join a server they are in, and the logout button to log the user out. I used a navigation bar from Bulma (Line 28), they contain documentation on templates for quick implementation.



```
28  <div id="navbarBasicExample" className="navbar-menu">
29      <div className="navbar-end">
30          <a className="navbar-item" href="http://127.0.0.1:8000"><i className="material-icons">home</i>Home</a>
31          <a className="navbar-item" onClick={()=>{
32              {/*directs user to notifications*/}
33              history.push("/notifications");
34          }}><i className="material-icons">notifications</i>Notifications</a>
35          <div className="navbar-item">
36              <div className="buttons">
37                  <button className="button is-light" onClick={()=>{
38                          logout();
39                          setisAuth(false);
40                          history.push('/');
41                      }}>Log out</button>
42              </div>
43          </div>
44      </div>
45  </div>
```

Figure 4.13: Navigation bar

4.7 Explore servers

When a user enters the explore servers page they have the option to search for a server by text, or filter search by server categories. When searching using the text field the user must enter a server name, or a partial server name and submit the search. Full server names are not required in the search, if the user searches the letter ”D” all servers beginning with the letter ”D” will be displayed. When filtering by server category the user has an option to select a category and all servers placed in that category will be displayed.

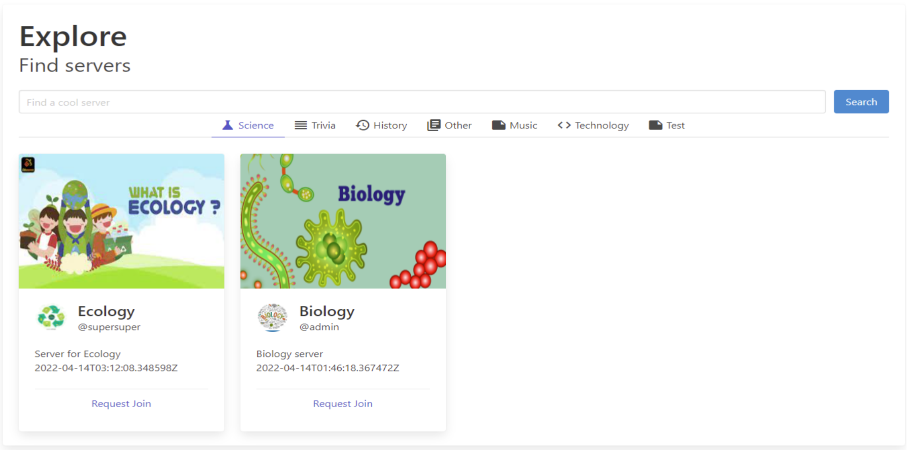
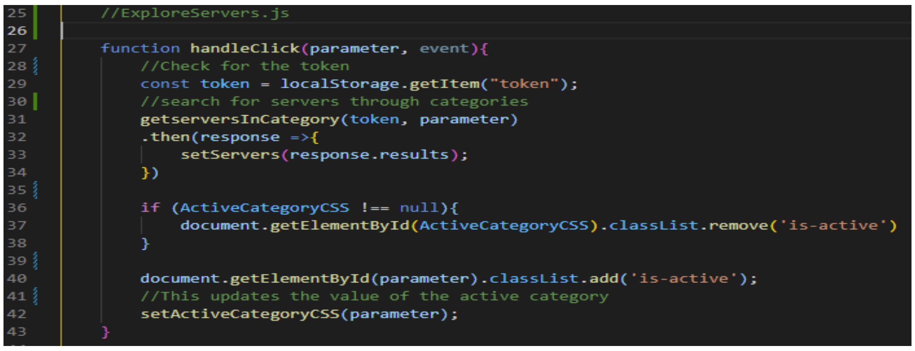


Figure 4.14: Explore server page

The JavaScript function below contains a promise and uses states to determine if there are servers in a category. The state *getServersInCategory* is used to return all of the active servers in the selected category. If the *ActiveCategoryCSS* is different to null, then it is no longer the selected server and the *is-active* status is removed.



```
//ExploreServers.js

function handleClick(parameter, event){
    //Check for the token
    const token = localStorage.getItem("token");
    //search for servers through categories
    getserversInCategory(token, parameter)
    .then(response =>{
        setServers(response.results);
    })

    if (ActiveCategoryCSS !== null){
        document.getElementById(ActiveCategoryCSS).classList.remove('is-active')
    }

    document.getElementById(parameter).classList.add('is-active');
    //This updates the value of the active category
    setActiveCategoryCSS(parameter);
}
```

Figure 4.15: Explore server code

4.8 Create servers

When the user enters the create server page they have multiple data fields to fill out to create their own server and obtain administrative rights. A server title, description, picture, banner, and category are required to be filled out before creating a new server. The upload image buttons open the file explorer when clicked, and a user can select and upload an image from their local machine.

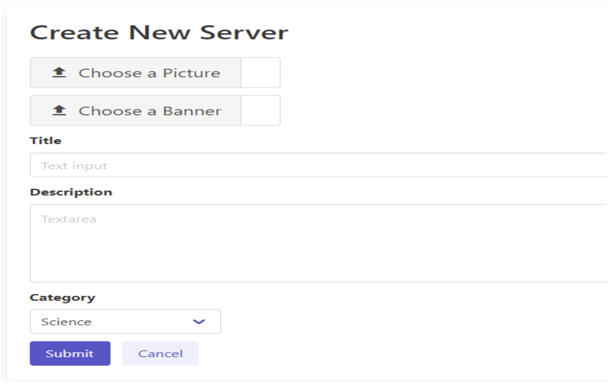
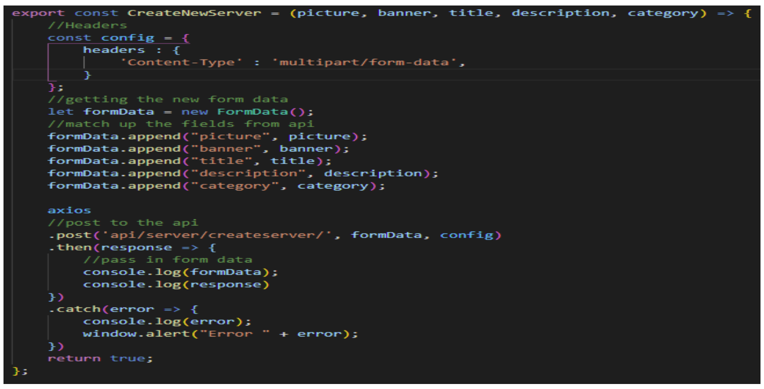


Figure 4.16: Create server page

To create a server a method is implemented with the server model fields in the parameters to take in new data for a server. I defined the content type and set it as multi form data as this request takes in multiple parts of data. To apply the new form data I listed the server model fields (that matches the API table) and append them with the new variables.A post request to the api using Axios is then called.

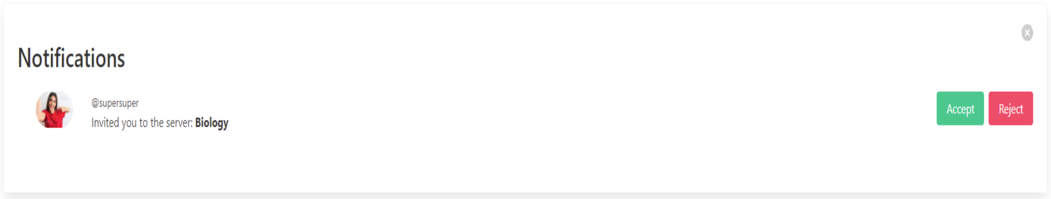


```
export const CreateNewServer = (picture, banner, title, description, category) => {
    //Headers
    const config = {
        headers : {
            'Content-Type' : 'multipart/form-data',
        }
    };
    //getting the new form data
    let formData = new FormData();
    //match up the fields from api
    formData.append("picture", picture);
    formData.append("banner", banner);
    formData.append("title", title);
    formData.append("description", description);
    formData.append("category", category);

    axios
    //post to the api
    .post('api/server/createserver/', formData, config)
    .then(response => {
        //pass in form data
        console.log(formData);
        console.log(response)
    })
    .catch(error => {
        console.log(error);
        window.alert("Error " + error);
    })
    return true;
};
```

Figure 4.17: CreateNewServer code

4.9 Notifications

When the user clicks the notification button on the navigation bar they are able to view all their invitations from other users to join their server. There is also a second notification type when a user requests to join a server, the notification is sent to the server administrator. The user has the option to accept or reject invitations. To display the users notification a post request using axios is used. It creates a new URL for getting notifications and returns a data promise. This method is called when the user clicks on the notification tab.


Notifications

@supersuper
Invited you to the server: **Biology**

Figure 4.18: Notification page

Seen below is the model for notifications, it contains a tuple for notification types and three other fields; for the user sending the notification, the user receiving the notification and the server that is being requested.

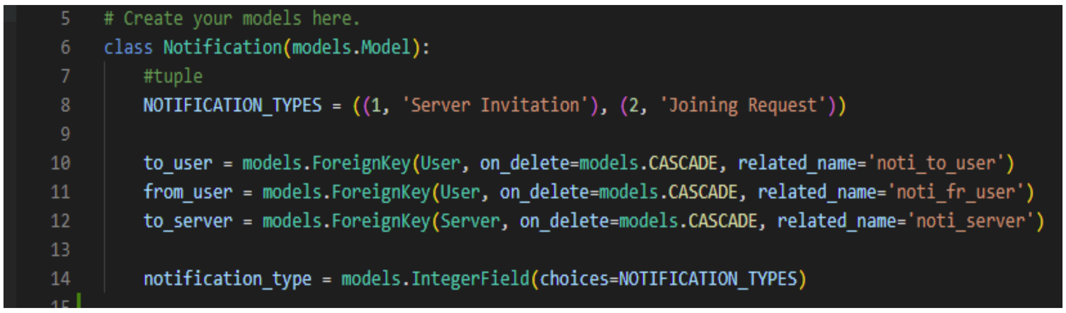
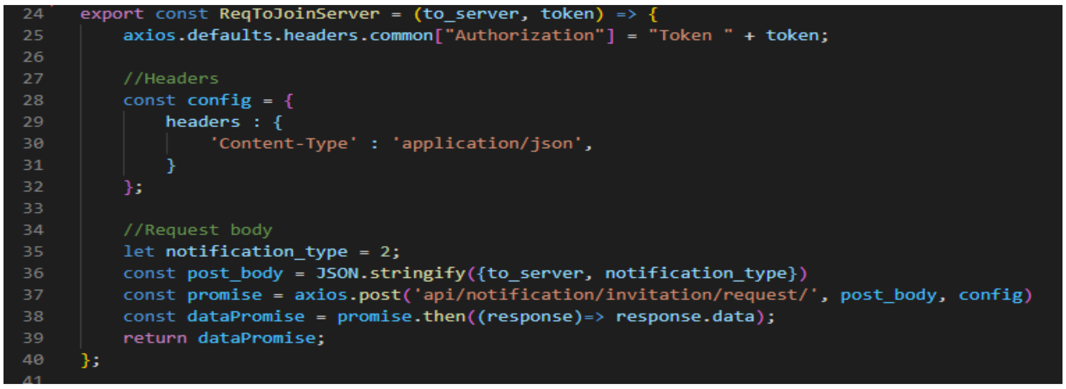

```
# Create your models here.
class Notification(models.Model):
    #tuple
    NOTIFICATION_TYPES = ((1, 'Server Invitation'), (2, 'Joining Request'))

    to_user = models.ForeignKey(User, on_delete=models.CASCADE, related_name='noti_to_user')
    from_user = models.ForeignKey(User, on_delete=models.CASCADE, related_name='noti_fr_user')
    to_server = models.ForeignKey(Server, on_delete=models.CASCADE, related_name='noti_server')

    notification_type = models.IntegerField(choices=NOTIFICATION_TYPES)
```

Figure 4.19: Notification model

The two different types of notifications can be defined by putting a *notification type* in the request body. When a notification is from a request to join a server the *notification type* is tagged as ”2” (on line 35). When a notification is an invitation to join a server the *notification type* is tagged as ”1”.

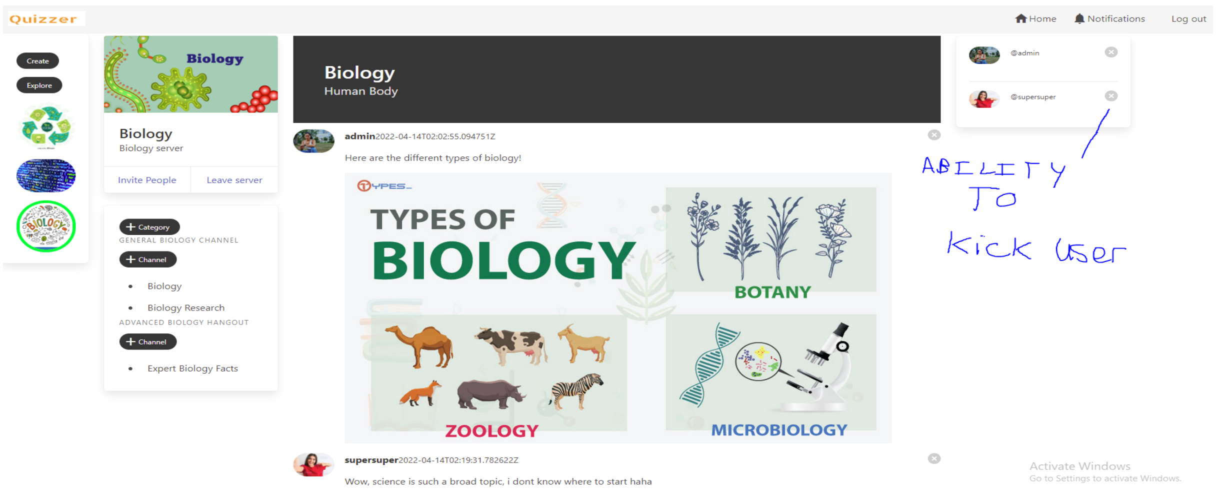


```
24  export const ReqToJoinServer = (to_server, token) => {
25      axios.defaults.headers.common["Authorization"] = "Token " + token;
26
27      //Headers
28      const config = {
29          headers : {
30              'Content-Type' : 'application/json',
31          }
32      };
33
34      //Request body
35      let notification_type = 2;
36      const post_body = JSON.stringify({to_server, notification_type})
37      const promise = axios.post('api/notification/invitation/request/', post_body, config)
38      const dataPromise = promise.then((response)=> response.data);
39      return dataPromise;
40  };
41
```

Figure 4.20: Notification code

4.10 Server

When a user enters a server page they are shown the server title, description and a list of members in the server. An option to leave the server or to invite another user is available. When the user selects a text channel (beneath the server description) the chat component is displayed.



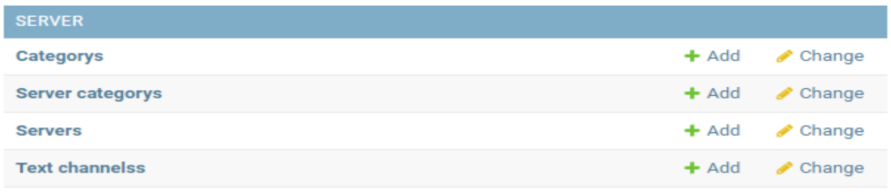
admin2022-04-14T02:02:55.094751Z
Here are the different types of biology!

supersuper2022-04-14T02:19:31.782622Z
Wow, science is such a broad topic, i dont know where to start haha

ABILITY TO Kick User

Figure 4.21: Server page

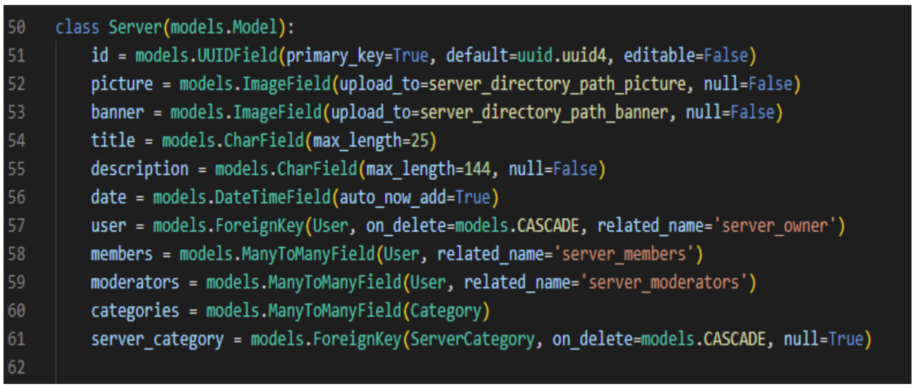
In the diagram below you can see a server is consisted of categories, server categories, servers and text channels.



SERVER		
Categorys	+ Add	Change
Server categorys	+ Add	Change
Servers	+ Add	Change
Text channelss	+ Add	Change

Figure 4.22: Server API in admin panel

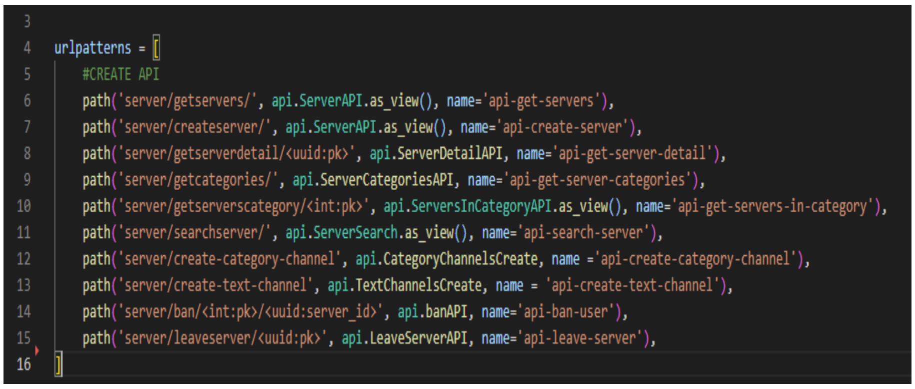
Below is the model for the Server (server/models.py). There are two foreign keys, *user* and *category*. User refers to the server creator, the profile who created the server is linked here. Every server must belong to a predefined category (Science, Entertainment, History), this is the use of the second foreign key as Categories is being linked. Many to many relationships are used for server members, moderators and categories, as multiple people can be in multiple servers.



```python
class Server(models.Model):
    id = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=False)
    picture = models.ImageField(upload_to=server_directory_path_picture, null=False)
    banner = models.ImageField(upload_to=server_directory_path_banner, null=False)
    title = models.CharField(max_length=25)
    description = models.CharField(max_length=144, null=False)
    date = models.DateTimeField(auto_now_add=True)
    user = models.ForeignKey(User, on_delete=models.CASCADE, related_name='server_owner')
    members = models.ManyToManyField(User, related_name='server_members')
    moderators = models.ManyToManyField(User, related_name='server_moderators')
    categories = models.ManyToManyField(Category)
    server_category = models.ForeignKey(ServerCategory, on_delete=models.CASCADE, null=True)

```

Figure 4.23: Server models

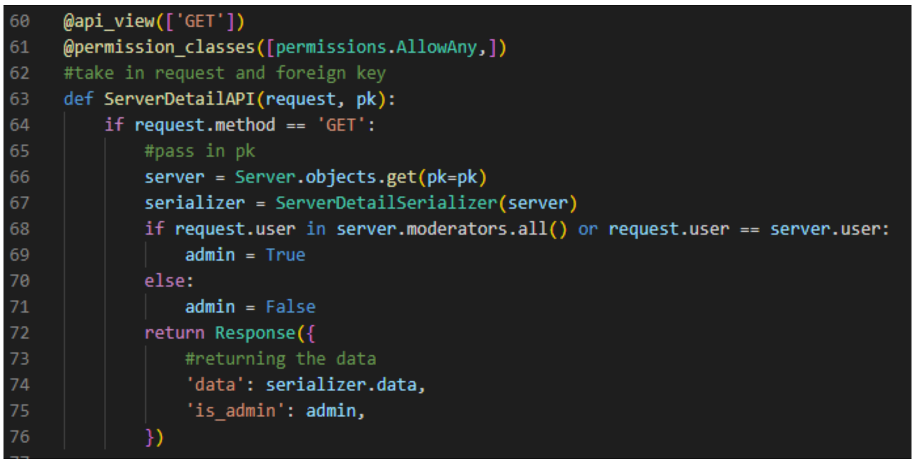
For routing and navigation of the web-page views are used and are linked with the API views/methods created. In the *server/urls.py* file I created basic patterns for the browser, when the browser URL matches a view URL this path is called. Inside of these basic urls I have imported my API views/methids and import them into the basic view. Below are my URL patterns for *Server*:



```
urlpatterns = [
    #CREATE API
    path('server/getservers/', api.ServerAPI.as_view(), name='api-get-servers'),
    path('server/createserver/', api.ServerAPI.as_view(), name='api-create-server'),
    path('server/getserverdetail/<uuid:pk>', api.ServerDetailAPI, name='api-get-server-detail'),
    path('server/getcategories/', api.ServerCategoriesAPI, name='api-get-server-categories'),
    path('server/getserverscategory/<int:pk>', api.ServersInCategoryAPI.as_view(), name='api-get-servers-in-category'),
    path('server/searchserver/', api.ServerSearch.as_view(), name='api-search-server'),
    path('server/create-category-channel', api.CategoryChannelsCreate, name ='api-create-category-channel'),
    path('server/create-text-channel', api.TextChannelsCreate, name = 'api-create-text-channel'),
    path('server/ban/<int:pk>/<uuid:server_id>', api.banAPI, name='api-ban-user'),
    path('server/leaveserver/<uuid:pk>', api.LeaveServerAPI, name='api-leave-server'),
]
```

Figure 4.24: URL patterns for Server

Below you can see the an example of a Server API view. In order to retrieve the server detail information, the request is taken in along with the users *pk* foreign key. The server requests the serialized data as it is used to return all of the server information, and if the user is the administrator or not



```
@api_view(['GET'])
@permission_classes([permissions.AllowAny,])
#take in request and foreign key
def ServerDetailAPI(request, pk):
    if request.method == 'GET':
        #pass in pk
        server = Server.objects.get(pk=pk)
        serializer = ServerDetailSerializer(server)
        if request.user in server.moderators.all() or request.user == server.user:
            admin = True
        else:
            admin = False
        return Response({
            #returning the data
            'data': serializer.data,
            'is_admin': admin,
        })
```

Figure 4.25: ServerDetail API view

4.10.1 Create categories

Inside every server are categories that members can create. For example a biology server may contain ”facts and learning” and a ”general biology” category. A category consists of a text channel or multiple text channels. To create a category the user clicks the add category button. A pop up is presented and the user must fill in a new title for the category and save changes.

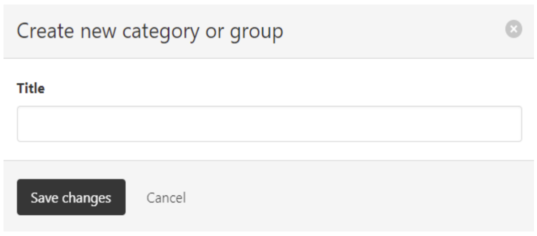
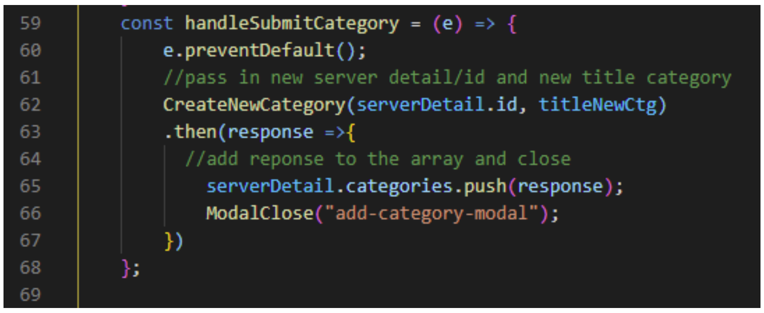


Figure 4.26: Create category page

After the user fills in the text field and requests to add a new category for the server the *handleSubmitCategory* method is used and brings in the new data the user passed in. The method *CreateNewCategory* is an axios post request to the API we created. The server ID and the title for the new category are passed in the method and the request is completed and added.



```
const handleSubmitCategory = (e) => {
    e.preventDefault();
    //pass in new server detail/id and new title category
    CreateNewCategory(serverDetail.id, titleNewCtg)
    .then(response =>{
      //add reponse to the array and close
        serverDetail.categories.push(response);
        ModalClose("add-category-modal");
    })
};
```

Figure 4.27: Submit method for category

4.10.2 Create channels

A text channel can be seen as a child of their given category. To create a channel the user must click the "+ Channel" button beneath the category it is going under. This links a relationship with an existing category to a new channel. There can be many text channels to a category. The user must fill in a title for the text channel and save changes to create the channel.

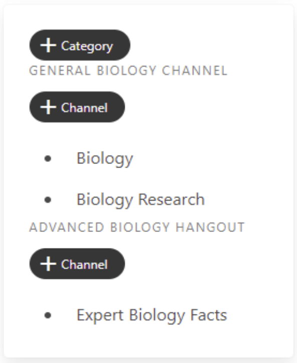
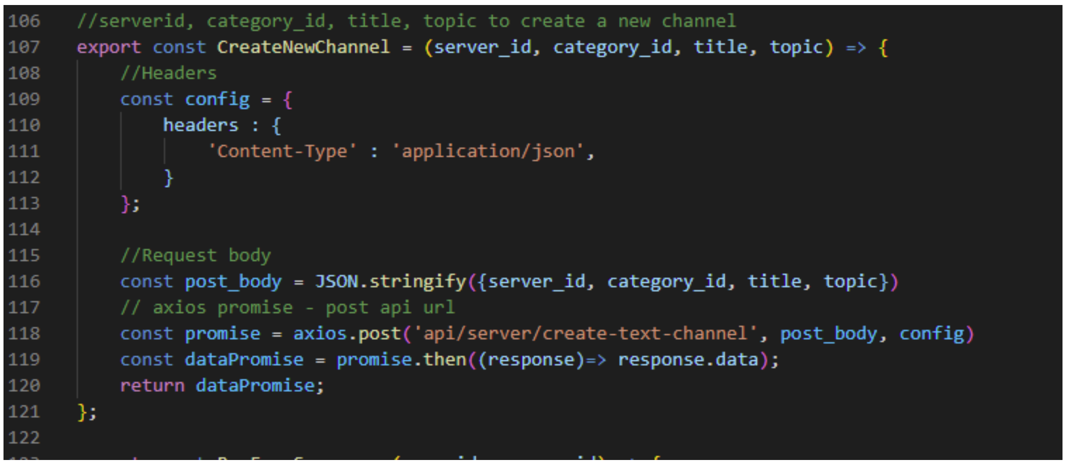


Figure 4.28: Options to add a chat category or channel

After the user fills in the title for the new text channel a submit button is used to call a method containing an axios post request to the API endpoint created "api/server/create-text-channel". The post request uses JSON format to handle the data.



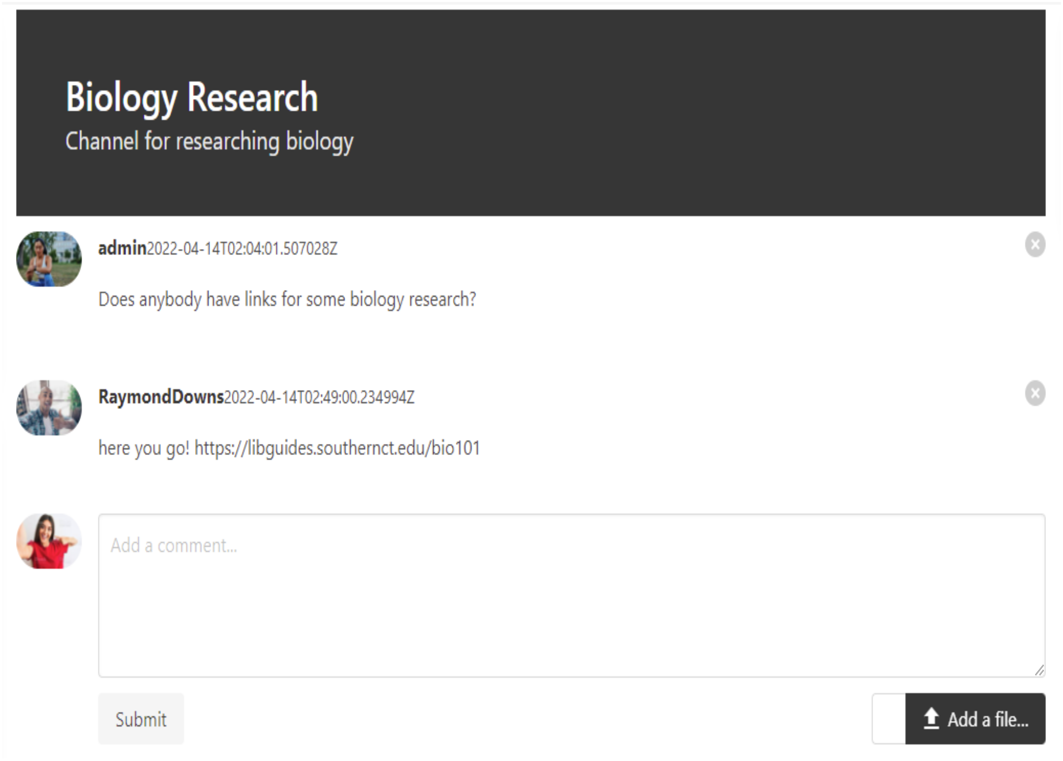
```
//serverid, category_id, title, topic to create a new channel
export const CreateNewChannel = (server_id, category_id, title, topic) => {
    //Headers
    const config = {
        headers : {
            'Content-Type' : 'application/json',
        }
    };

    //Request body
    const post_body = JSON.stringify({server_id, category_id, title, topic})
    // axios promise - post api url
    const promise = axios.post('api/server/create-text-channel', post_body, config)
    const dataPromise = promise.then((response)=> response.data);
    return dataPromise;
};
```

Figure 4.29: Create axios URL for new channel

The above method discussed is called in the *handleSubmitChannel* method. This method retrieves the category selected to build a channel under, it filters the category by passing an object until it matches the category for the channel. The text channel is then added.

4.10.3 Main chat

When a user is in a server they are able to enter a chat channel by clicking on the channel name. The main chat displays a record of the chat history. Each message is recorded in the main chat with a timestamp and the username who has posted a message. Administrators have the ability to delete any message they deem inappropriate. To submit a message in the main chat the user must enter text in the text field and press submit. The user also has the option to upload an image from their local machine.



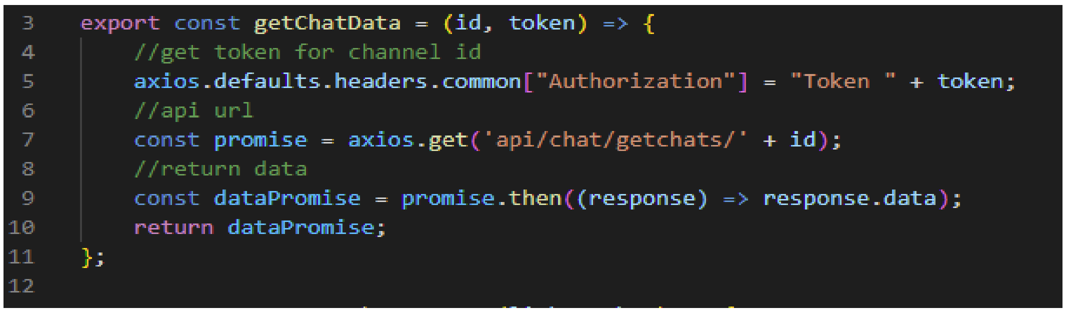
Biology Research
Channel for researching biology

admin2022-04-14T02:04:01.507028Z
Does anybody have links for some biology research?

RaymondDowns2022-04-14T02:49:00.234994Z
here you go! https://libguides.southernct.edu/bio101

Figure 4.30: Main chat

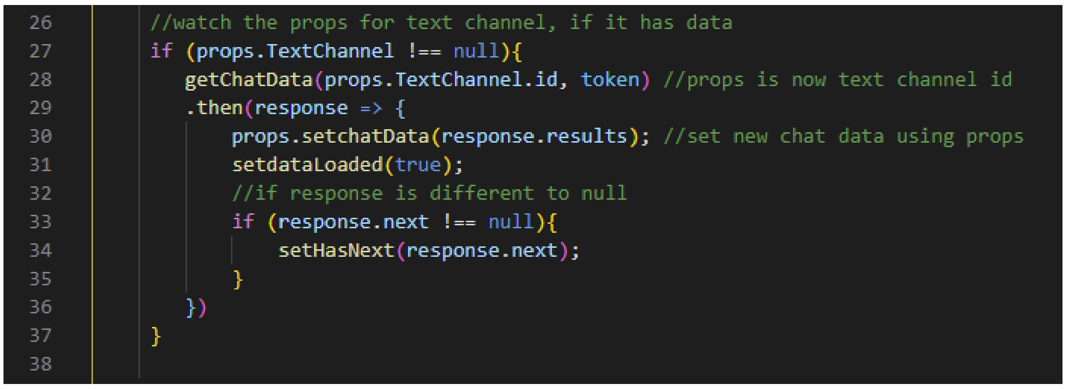
getChatData is used in the front-end service to get the chat data a function is created using axios, the method verifies the user is logged in, and creates an API URL specific to getting chats and the server id. A promise is then returned .



```javascript
export const getChatData = (id, token) => {
    //get token for channel id
    axios.defaults.headers.common["Authorization"] = "Token " + token;
    //api url
    const promise = axios.get('api/chat/getchats/' + id);
    //return data
    const dataPromise = promise.then((response) => response.data);
    return dataPromise;
};
```

Figure 4.31: Get chat data

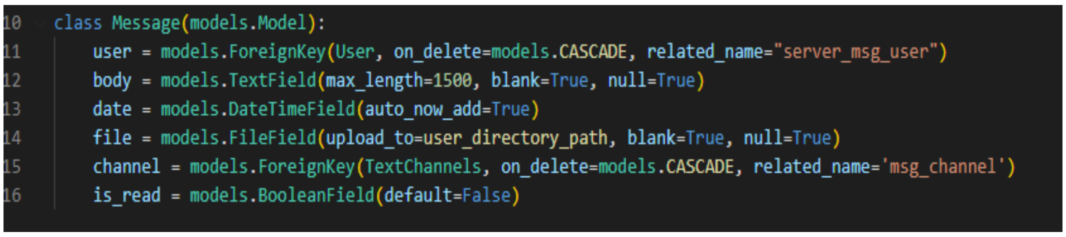
If the text channel contains data the method *getChatData* will be called using props as the text channel ID. If the response has data it will continue to load until complete. When all the data is read data loaded is set to true.



```javascript
        //watch the props for text channel, if it has data
        if (props.TextChannel !== null){
            getChatData(props.TextChannel.id, token) //props is now text channel id
            .then(response => {
                props.setchatData(response.results); //set new chat data using props
                setdataLoaded(true);
                //if response is different to null
                if (response.next !== null){
                    setHasNext(response.next);
                }
            })
        }
```

Figure 4.32: Props on TextChannel

Below is the model for Message (chat/models.py). There are two foreign keys, *TextChannels* and *User*. The message posted must belong to a user, and the message must belong to a relevant text channel. The rest of the model consists of text fields and file fields.

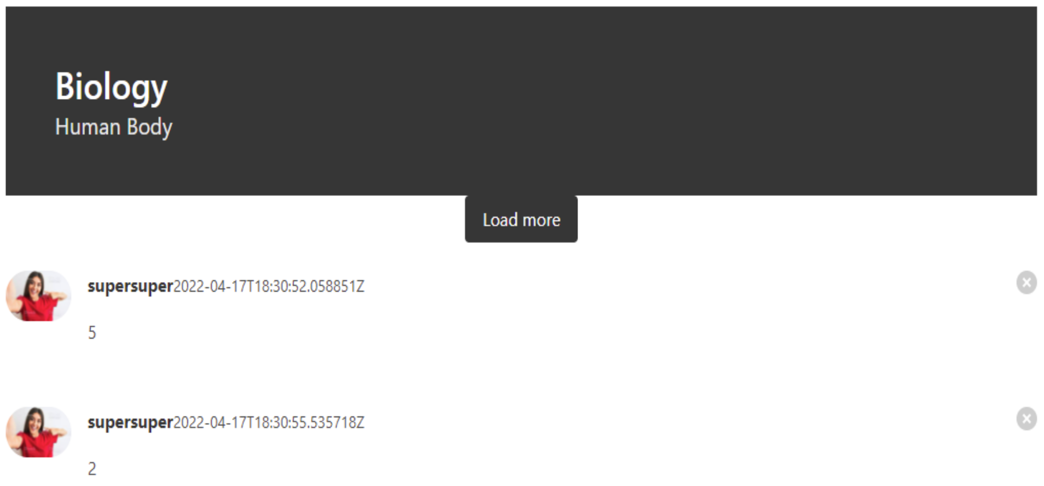


```python
class Message(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE, related_name="server_msg_user")
    body = models.TextField(max_length=1500, blank=True, null=True)
    date = models.DateTimeField(auto_now_add=True)
    file = models.FileField(upload_to=user_directory_path, blank=True, null=True)
    channel = models.ForeignKey(TextChannels, on_delete=models.CASCADE, related_name='msg_channel')
    is_read = models.BooleanField(default=False)
```

Figure 4.33: Model for Message

Load More

When the user enters a text channel the load more button is displayed for text channels with five or more posts. The chat API is setup as the ResponsePagination will only display five objects or messages. This to stop the user being overwhelmed with messages when they first open the channel. The load more button provides the option to load the next five chats and can continue doing so if there's messages available.



supersuper2022-04-17T18:30:52.058851Z

5

supersuper2022-04-17T18:30:55.535718Z

2

Figure 4.34: Load more button

4.11 Quiz

On the homepage users are presented the option to take a trivia quiz, the type of quiz can be determined by selecting the category and difficulty. By pul users are able to select one of twenty five different quiz categories ranging from science: mathematics to entertainment: cartoon/animation. The second field defines the difficulty of questions the quiz will contain, difficulty options include easy medium and hard. The last field is the array of questions asked in the quiz. The user can enter a number from one to ten to define how many questions the quiz will contain.

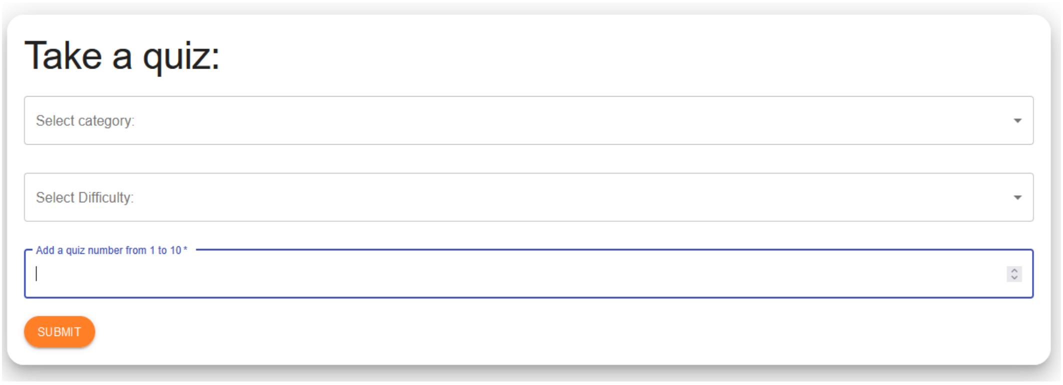
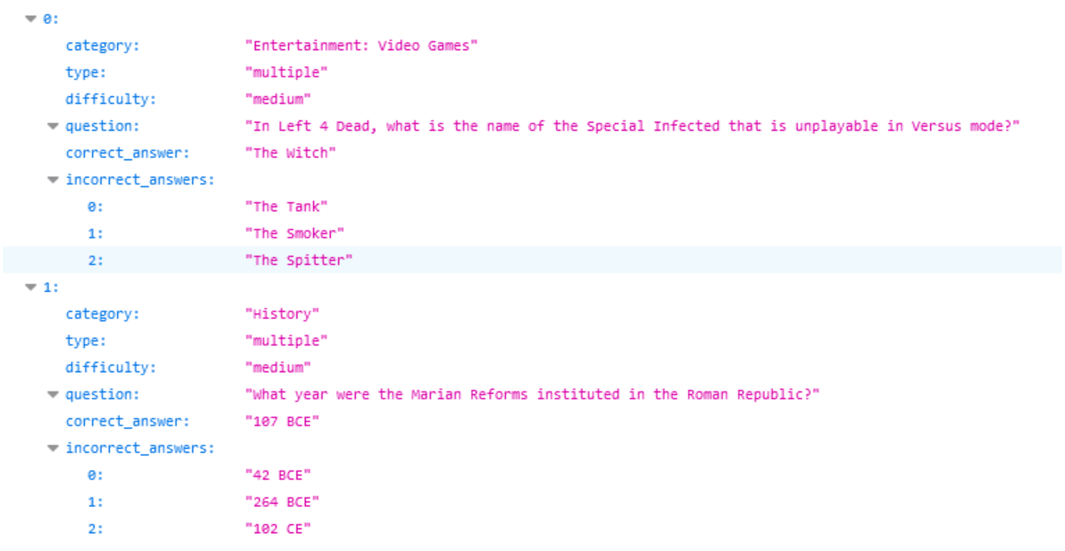


Figure 4.35: Quiz setup

Trivia API

Trivia API is a free JSON API, it is free to use and has an open database. The API can be broken down into categories: Number of Questions, Category, Difficulty and Question Type. Trivia API is a free to use and comes with well recorded documentation. Using this API does not require an API key, I am using a generated URL to retrieve trivia questions for my quiz function.

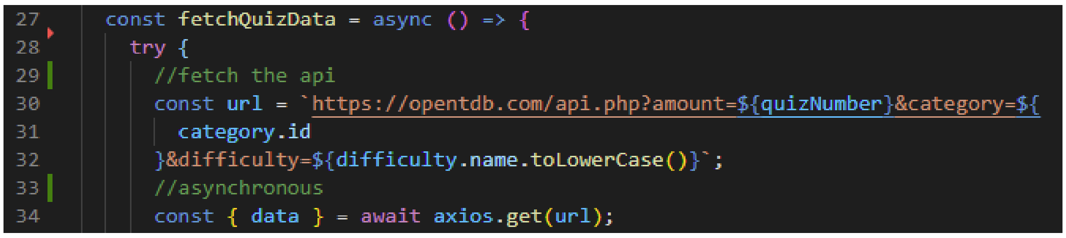


```
▾ 0:
    category:           "Entertainment: Video Games"
    type:               "multiple"
    difficulty:         "medium"
  ▾ question:           "In Left 4 Dead, what is the name of the Special Infected that is unplayable in Versus mode?"
    correct_answer:     "The Witch"
  ▾ incorrect_answers:
       0:               "The Tank"
       1:               "The Smoker"
       2:               "The Spitter"
▾ 1:
    category:           "History"
    type:               "multiple"
    difficulty:         "medium"
  ▾ question:           "What year were the Marian Reforms instituted in the Roman Republic?"
    correct_answer:     "107 BCE"
  ▾ incorrect_answers:
       0:               "42 BCE"
       1:               "264 BCE"
       2:               "102 CE"
```

Figure 4.36: Trivia API

4.11.1 Quiz Setup

To pull the API data into our application a function *fetchQuizData* is created. Using the generated URL from Trivia API an asynchronous request is called to the server.

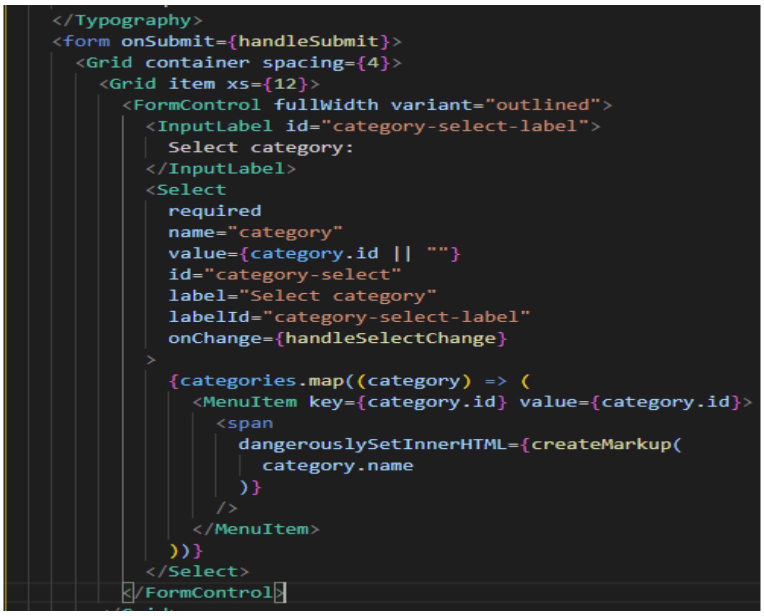


```
const fetchQuizData = async () => {
  try {
    //fetch the api
    const url = `https://opentdb.com/api.php?amount=${quizNumber}&category=${
      category.id
    }&difficulty=${difficulty.name.toLowerCase()}`;
    //asynchronous
    const { data } = await axios.get(url);
```

Figure 4.37: Getting quiz data from API

Initially when pulling the API the correct answer for each question was the first answer listed. A method to retrieve a random number using *Math.round()*is used to generate a random index (positioning on the question list) for the correct answer. Users will now have to figure out the correct answer.

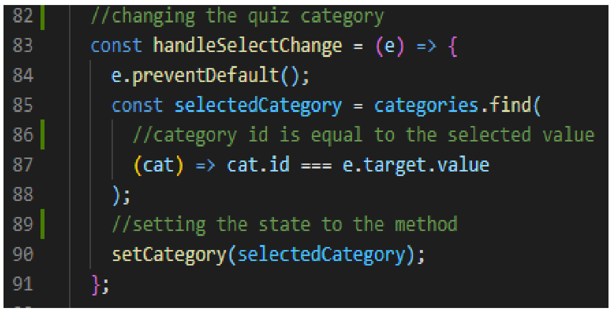
For the quiz component I am using Material-UI for styling and layout. When the user clicks the category or difficulty field a drop-down list of all the categories or difficulty levels are displayed. This is done by mapping out the key, *handle* methods are created for selecting said category and difficulty.



```
</Typography>
<form onSubmit={handleSubmit}>
  <Grid container spacing={4}>
    <Grid item xs={12}>
      <FormControl fullWidth variant="outlined">
        <InputLabel id="category-select-label">
          Select category:
        </InputLabel>
        <Select
          required
          name="category"
          value={category.id || ""}
          id="category-select"
          label="Select category"
          labelId="category-select-label"
          onChange={handleSelectChange}
        >
          {categories.map((category) => (
            <MenuItem key={category.id} value={category.id}>
              <span
                dangerouslySetInnerHTML={createMarkup(
                  category.name
                )}
              />
            </MenuItem>
          ))}
        </Select>
      </FormControl>
```

Figure 4.38: Quiz category select

The *handleSelectChange* method is used when a user selects a category from the drop down list. This method searches through all the categories in the API and matches it to the one selected by the user. The new *selectedCategory* state gets set in the *setCategory* method to select the category being fetched for the quiz.



```
//changing the quiz category
const handleSelectChange = (e) => {
  e.preventDefault();
  const selectedCategory = categories.find(
    //category id is equal to the selected value
    (cat) => cat.id === e.target.value
  );
  //setting the state to the method
  setCategory(selectedCategory);
};
```

Figure 4.39: Changing quiz category

4.11.2 Answer Questions

After the quiz has been initiated the quiz component displays the list of questions generated from the API. Each quiz is multiple choice and contains up to four different answers per question, true or false questions are also included. After the user has selected an answer for each question they can submit their answers and their result will be shown immediately. The user has the option to review the correct answers or reset the quiz and create a new one.

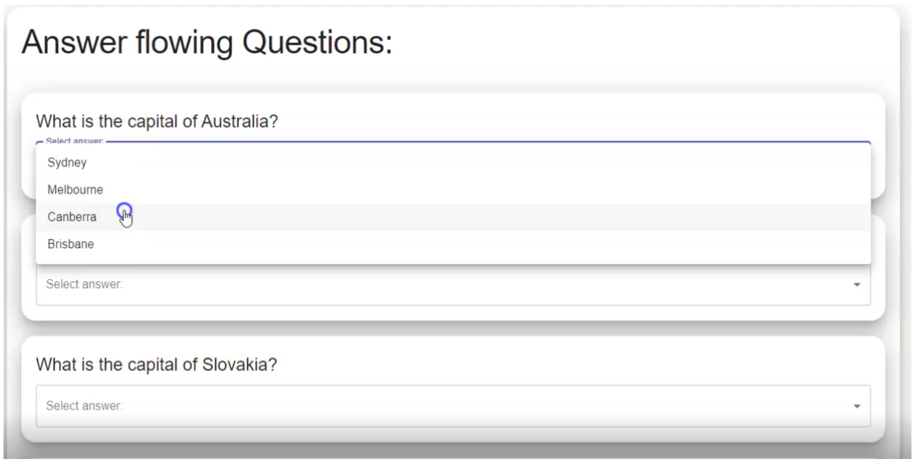
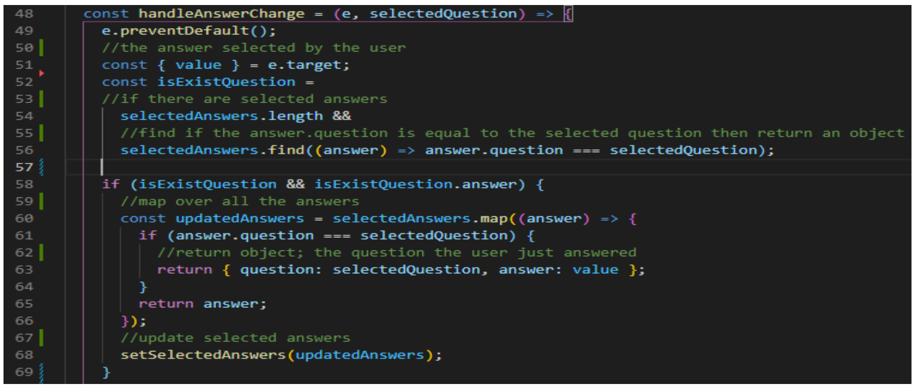

Answer flowing Questions:
What is the capital of Australia?
Select answer
Sydney
Melbourne
Canberra
Brisbane
Select answer
What is the capital of Slovakia?
Select answer

Figure 4.40: Answering questions

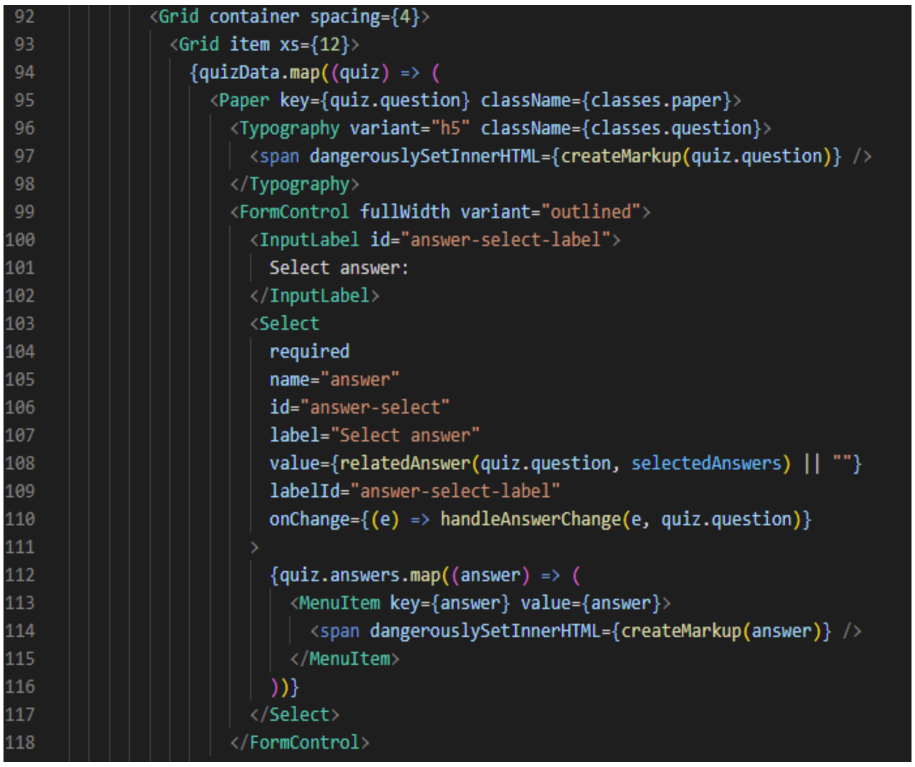
To display the quiz I use Material-UI grid and map through all quiz data. The *dangerouslySetInnerHTML()* renders the JSON data (quiz answers and questions) in a more readable format. There is an *onChange* method *handleAnswerChange* that is be called when the user selects a new answer.



```
const handleAnswerChange = (e, selectedQuestion) => {
  e.preventDefault();
  //the answer selected by the user
  const { value } = e.target;
  const isExistQuestion =
  //if there are selected answers
    selectedAnswers.length &&
    //find if the answer.question is equal to the selected question then return an object
    selectedAnswers.find((answer) => answer.question === selectedQuestion);

  if (isExistQuestion && isExistQuestion.answer) {
    //map over all the answers
    const updatedAnswers = selectedAnswers.map((answer) => {
      if (answer.question === selectedQuestion) {
        //return object; the question the user just answered
        return { question: selectedQuestion, answer: value };
      }
      return answer;
    });
    //update selected answers
    setSelectedAnswers(updatedAnswers);
  }
```

Figure 4.41: Selected answer

There is another nested map function being called inside the questions to display and map all of the answers. The *handleAnswerChange* updates a selected answer when chosen by the user. This is done by using states and setting a new value when a question is clicked. If there are selected answers the function will return the object and add them to the array of answered questions.



```
        <Grid container spacing={4}>
          <Grid item xs={12}>
            {quizData.map((quiz) => (
              <Paper key={quiz.question} className={classes.paper}>
                <Typography variant="h5" className={classes.question}>
                  <span dangerouslySetInnerHTML={createMarkup(quiz.question)} />
                </Typography>
                <FormControl fullWidth variant="outlined">
                  <InputLabel id="answer-select-label">
                    Select answer:
                  </InputLabel>
                  <Select
                    required
                    name="answer"
                    id="answer-select"
                    label="Select answer"
                    value={relatedAnswer(quiz.question, selectedAnswers) || ""}
                    labelId="answer-select-label"
                    onChange={(e) => handleAnswerChange(e, quiz.question)}
                  >
                    {quiz.answers.map((answer) => (
                      <MenuItem key={answer} value={answer}>
                        <span dangerouslySetInnerHTML={createMarkup(answer)} />
                      </MenuItem>
                    ))}
                  </Select>
                </FormControl>
```

Figure 4.42: Mapping out questions

4.11.3 Total Result

After completing the quiz the user will see how many correct answers they scored in the quiz, and the amount of questions in the quiz.

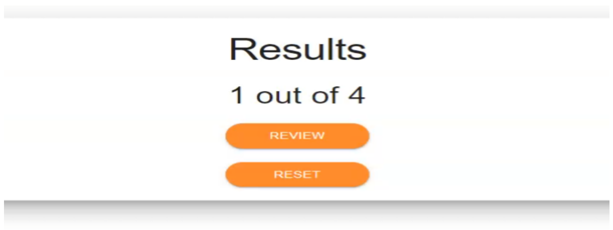
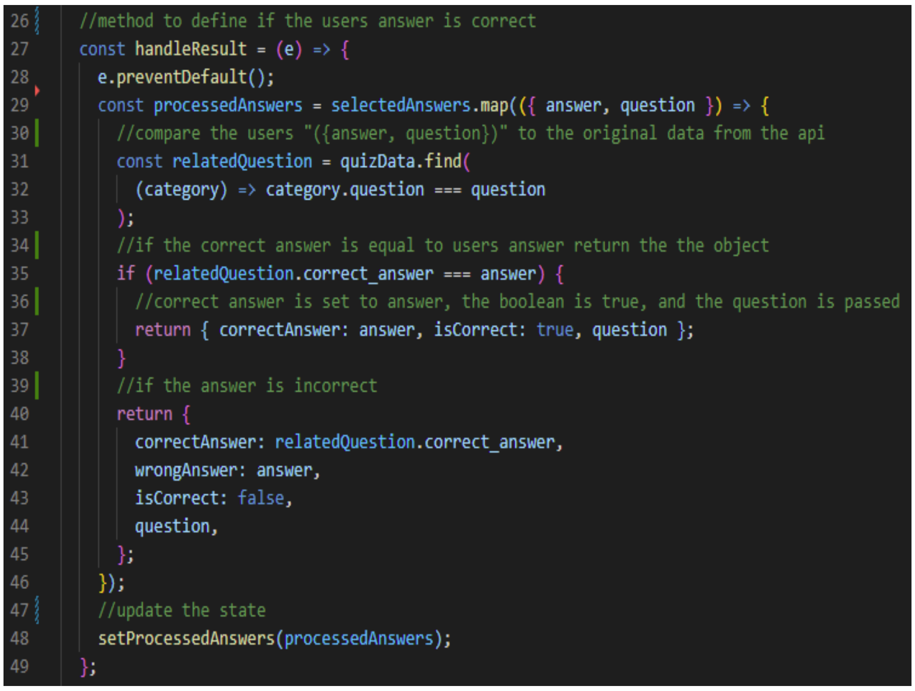


Figure 4.43: Total result of quiz

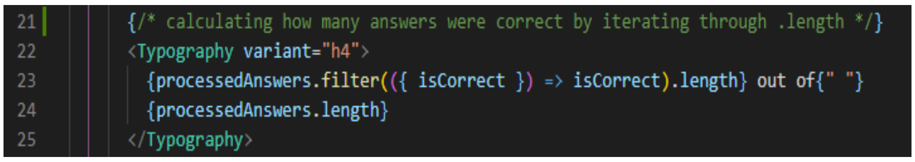
A method is used to determine if the users selected answer is equal to the correct answer from the API. The *selectedAnswers* are mapped out and if the correct answer is equal to the users answer the object is returned where the *isCorrect* boolean returns true. If the answer is incorrect the *isCorrect* boolean is set to false.



```
//method to define if the users answer is correct
const handleResult = (e) => {
  e.preventDefault();
  const processedAnswers = selectedAnswers.map(({ answer, question }) => {
    //compare the users "({answer, question})" to the original data from the api
    const relatedQuestion = quizData.find(
      (category) => category.question === question
    );
    //if the correct answer is equal to users answer return the the object
    if (relatedQuestion.correct_answer === answer) {
      //correct answer is set to answer, the boolean is true, and the question is passed
      return { correctAnswer: answer, isCorrect: true, question };
    }
    //if the answer is incorrect
    return {
      correctAnswer: relatedQuestion.correct_answer,
      wrongAnswer: answer,
      isCorrect: false,
      question,
    };
  });
  //update the state
  setProcessedAnswers(processedAnswers);
};
```

Figure 4.44: Determining correct or incorrect

To calculate the amount of correct answers the user has scored the answers are filtered using the *isCorrect* boolean. By running through the amount of correct answers using *.length* we can compare it to the amount of answers the user has chosen for the entire quiz. These two variables provide the users score.

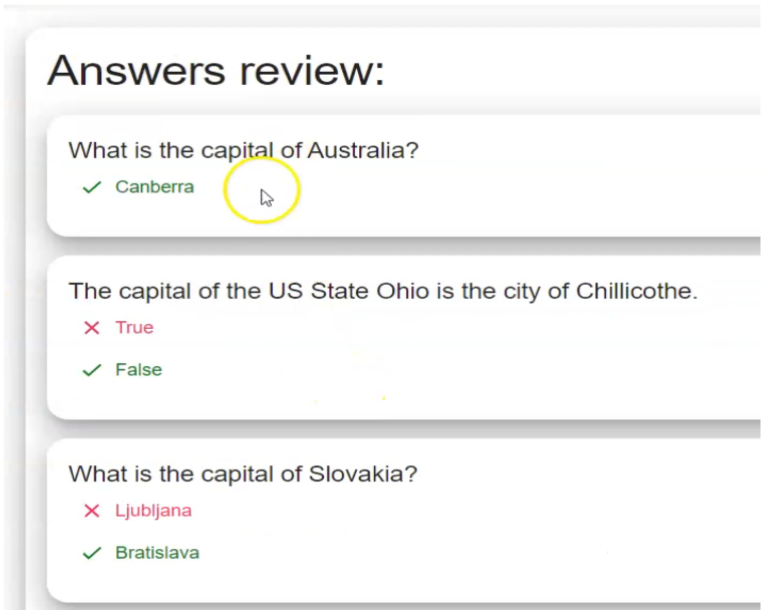


```
{/* calculating how many answers were correct by iterating through .length */}
<Typography variant="h4">
  {processedAnswers.filter(({ isCorrect }) => isCorrect).length} out of{" "}
  {processedAnswers.length}
</Typography>
```

Figure 4.45: Calculate correct answers

Quiz Review

When the user selects to review their answers a list of all the questions, correct answers and users answers are displayed. The user can see their submitted answer and the correct answer for the question, if the user chose the correct answer that answer is highlighted in green and is correct. If the user chose the wrong answer it is highlighted in red, with the correct answer above it. The user has the option to reset the quiz and compose a new one.



Answers review:

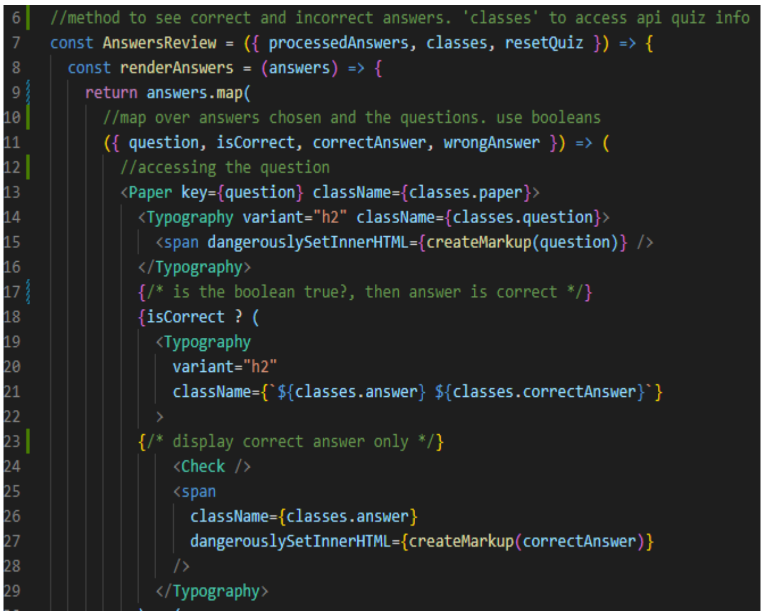
What is the capital of Australia?
✓ Canberra

The capital of the US State Ohio is the city of Chillicothe.
✗ True
✓ False

What is the capital of Slovakia?
✗ Ljubljana
✓ Bratislava

Figure 4.46: Answer review page

The answer review function takes in the users processed answers and the quiz information from the API being pulled *"classes" (Line 7)*. I mapped through all the questions and checked if each users answer is correct, I used a Boolean *isCorrect*. If this Boolean is true then user got the question right and the answer is displayed. If this condition fails then the answer is wrong, and the wrong answer will be displayed next to the correct one.



```
//method to see correct and incorrect answers. 'classes' to access api quiz info
const AnswersReview = ({ processedAnswers, classes, resetQuiz }) => {
  const renderAnswers = (answers) => {
    return answers.map(
      //map over answers chosen and the questions. use booleans
      ({ question, isCorrect, correctAnswer, wrongAnswer }) => (
        //accessing the question
        <Paper key={question} className={classes.paper}>
          <Typography variant="h2" className={classes.question}>
            <span dangerouslySetInnerHTML={createMarkup(question)} />
          </Typography>
          {/* is the boolean true?, then answer is correct */}
          {isCorrect ? (
            <Typography
              variant="h2"
              className={`${classes.answer} ${classes.correctAnswer}`}
            >
          {/* display correct answer only */}
              <Check />
              <span
                className={classes.answer}
                dangerouslySetInnerHTML={createMarkup(correctAnswer)}
              />
            </Typography>
```

Figure 4.47: Mapping out answers and correct questions

4.11.4 Database

To store all of the servers, and users details, pgAdmin is used on the PostgreSQL database. A new database ”quizzerdb” is created using the GUI that pgAdmin provides.

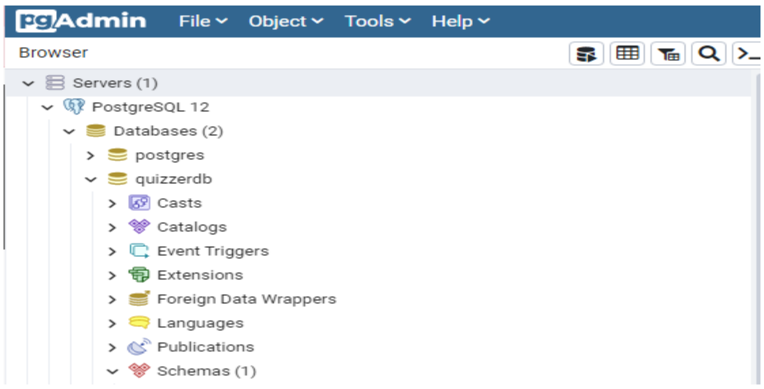
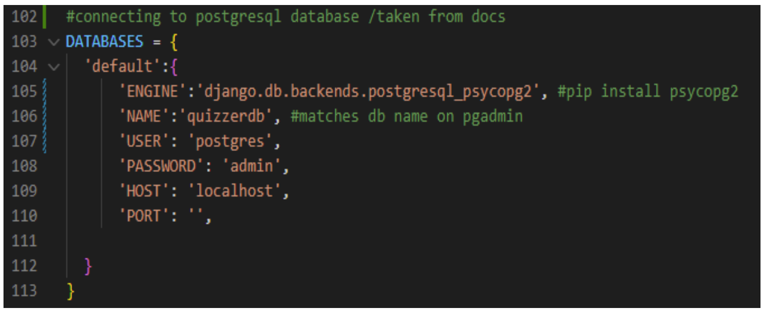


Figure 4.48: pgAdmin on Quizzer database

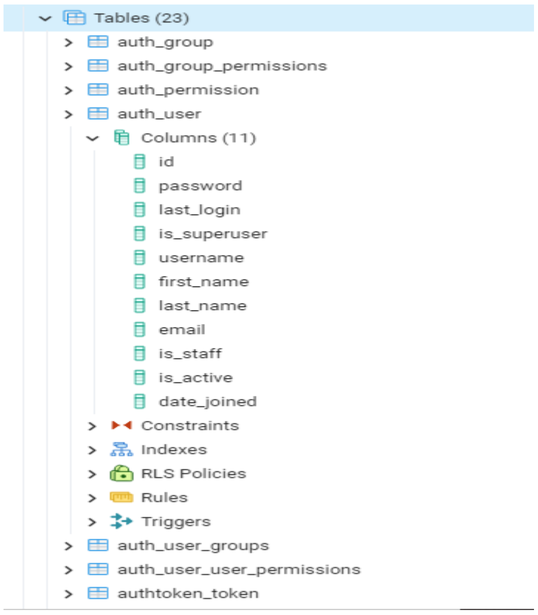
To import this database to my project I changed my *settings.py* according to the documentation. I changed the "NAME:" field to match the database i created on pgAdmin. The username and password for my account follows, and the default port is selected.



```python
#connecting to postgresql database /taken from docs
DATABASES = {
  'default':{
      'ENGINE':'django.db.backends.postgresql_psycopg2', #pip install psycopg2
      'NAME':'quizzerdb', #matches db name on pgadmin
      'USER': 'postgres',
      'PASSWORD': 'admin',
      'HOST': 'localhost',
      'PORT': '',

  }
}
```

Figure 4.49: Importing database

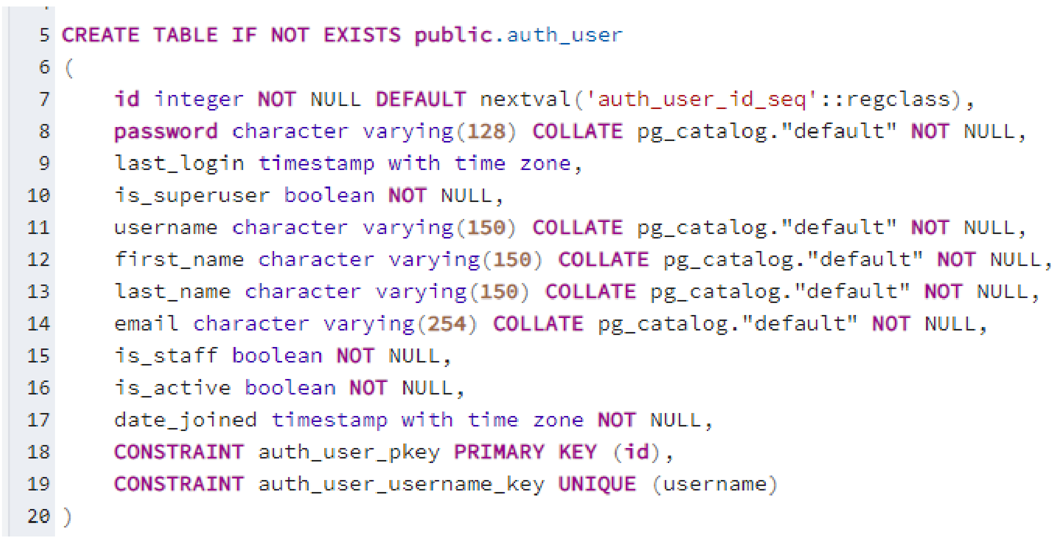
All of my tables and schema in my API I created are now available on the database and can be manipulated with SQL or the Create, Update, Delete, Edit graphical user interface.



```
Tables (23)
  auth_group
  auth_group_permissions
  auth_permission
  auth_user
    Columns (11)
      id
      password
      last_login
      is_superuser
      username
      first_name
      last_name
      email
      is_staff
      is_active
      date_joined
    Constraints
    Indexes
    RLS Policies
    Rules
    Triggers
  auth_user_groups
  auth_user_user_permissions
  authtoken_token
```

Figure 4.50: Tables on Quizzer DB

Below you can see the tables for the user profile imported into the PostgreSQL database from my API.



```
CREATE TABLE IF NOT EXISTS public.auth_user
(
    id integer NOT NULL DEFAULT nextval('auth_user_id_seq'::regclass),
    password character varying(128) COLLATE pg_catalog."default" NOT NULL,
    last_login timestamp with time zone,
    is_superuser boolean NOT NULL,
    username character varying(150) COLLATE pg_catalog."default" NOT NULL,
    first_name character varying(150) COLLATE pg_catalog."default" NOT NULL,
    last_name character varying(150) COLLATE pg_catalog."default" NOT NULL,
    email character varying(254) COLLATE pg_catalog."default" NOT NULL,
    is_staff boolean NOT NULL,
    is_active boolean NOT NULL,
    date_joined timestamp with time zone NOT NULL,
    CONSTRAINT auth_user_pkey PRIMARY KEY (id),
    CONSTRAINT auth_user_username_key UNIQUE (username)
)
```

Figure 4.51: User model in pgAdmin

Chapter 5

System Evaluation

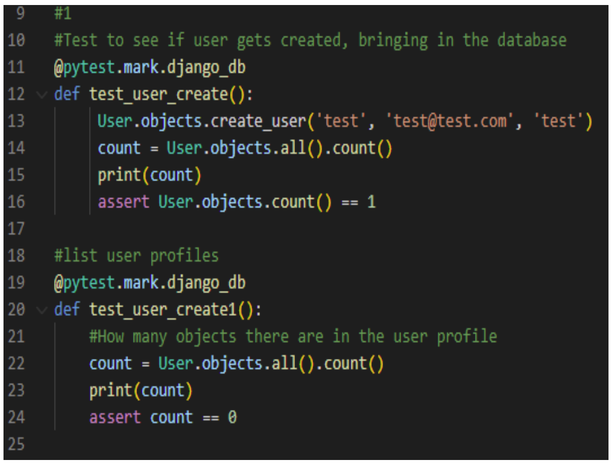
5.1 Testing

Testing plays a vital role to ensure your application is working as expected. Due to the linear process of the waterfall method testing is the last phase, unfortunately due to time constraints there is a limited amount of testing in my application. I have set up unit tests and ran a coverage test. During the development phase Postman was used to test my API endpoints and ensure they are working correctly.

5.2 Unit Tests

I have setup some unit tests in my application using Pytest, which is a fully-featured Python testing tool. Some of my tests include:

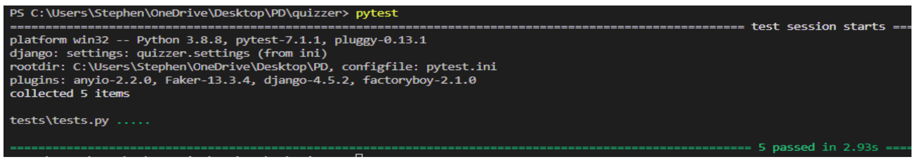
- User gets created and added to the database
- View and list all user profiles
- Setting a password for registration
- Ensuring password one and password two match on registration



```python
#1
#Test to see if user gets created, bringing in the database
@pytest.mark.django_db
def test_user_create():
    User.objects.create_user('test', 'test@test.com', 'test')
    count = User.objects.all().count()
    print(count)
    assert User.objects.count() == 1

#list user profiles
@pytest.mark.django_db
def test_user_create1():
    #How many objects there are in the user profile
    count = User.objects.all().count()
    print(count)
    assert count == 0

```

Figure 5.1: Unit tests

All of the tests above pass, this ensures that the user is added to the database, that the application can see all the users, that the passwords are eligible upon user registration

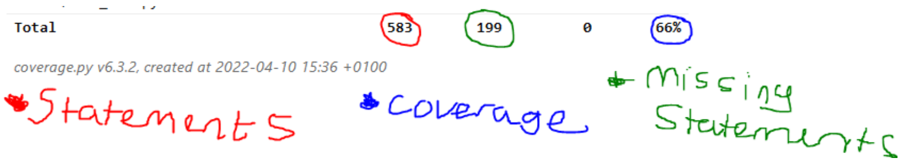


```
PS C:\Users\Stephen\OneDrive\Desktop\PD\quizzer> pytest
============================================================================================= test session starts ===
platform win32 -- Python 3.8.8, pytest-7.1.1, pluggy-0.13.1
django: settings: quizzer.settings (from ini)
rootdir: C:\Users\Stephen\OneDrive\Desktop\PD, configfile: pytest.ini
plugins: anyio-2.2.0, Faker-13.3.4, django-4.5.2, factoryboy-2.1.0
collected 5 items

tests\tests.py .....

============================================================================================== 5 passed in 2.93s ====
```

Figure 5.2: Running tests

5.3 Test Coverage

By running coverage tests I am able to see the percentage of my components that are covered by testing, and which ones should undergo new tests by the developer. After running a coverage test I can see that my application runs at 66 percent coverage.



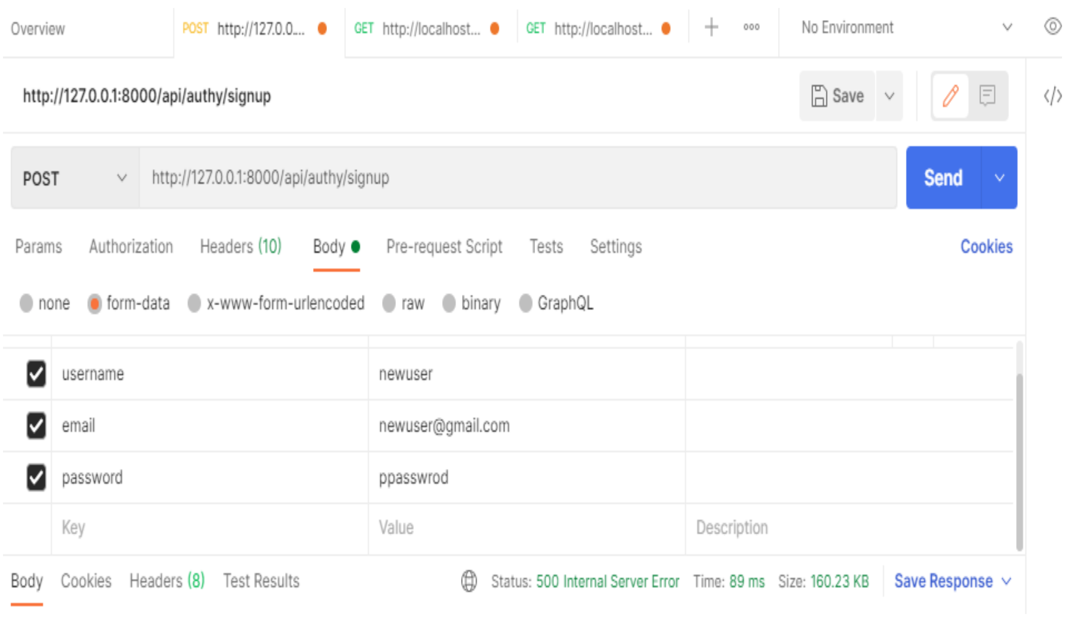
```
Total                                         583        199         0        66%

coverage.py v6.3.2, created at 2022-04-10 15:36 +0100
```

Figure 5.3: Test coverage report

Applications should aim for a test coverage of eighty percent or higher (trying to reach a hundred percent may be time costly). Based on my test coverage results I would have implemented more tests if I had more time in development.

5.4 Postman API Testing

I used Postman as a testing tool to test my network implementation and to test my API endpoints on the server. Postman is a free tool to use, and it helped me debug routing issues, server errors and more.



http://127.0.0.1:8000/api/authy/signup

POST http://127.0.0.1:8000/api/authy/signup

	Key	Value	Description
✓	username	newuser	
✓	email	newuser@gmail.com	
✓	password	ppasswrod	

Status: 500 Internal Server Error Time: 89 ms Size: 160.23 KB

Figure 5.4: Postman testing

5.5 Visual Studio Debugger

Visual Studio contains a debugger that helped me navigate and inspect the code during its execution flow. I was able to quickly inspect code and make adjustments without having to leave the developer environment.

5.6 Client Evaluation

Client side evaluation refers to the use and operations performed by the user while using the application. While in my design phase I ensured the layout of my application would be intuitive and self-explanatory. The user should be able to use all features of the website without having to read a *readme*. Key features of a website design is:

- Simplicity
- Navigation
- Visual Hierarchy
- Grid based layout

5.7 Performance

Visual Studio Code provides a lot of features for development. Studio code is able to measure app performance with profiling and diagnostic tools. The website is optimised using React.js tree structure of the DOM model to make the user interface respond faster. Performance tools that are used for measuring performance of a web-application include:

- Google PageSpeed Insights
- GTmetrix
- New Relic APM

5.8 Overall Evaluation

The basic system and functionality in the form of a flow chart when a user enters the application is shown below. Overall the user can create servers and take quizzes in the instance of a couple clicks. From the homepage the user is able to setup and take a quiz. By using Django and React the system is interactive and includes functionality. The system is stored in a secure and scale-able PostgreSQL database.

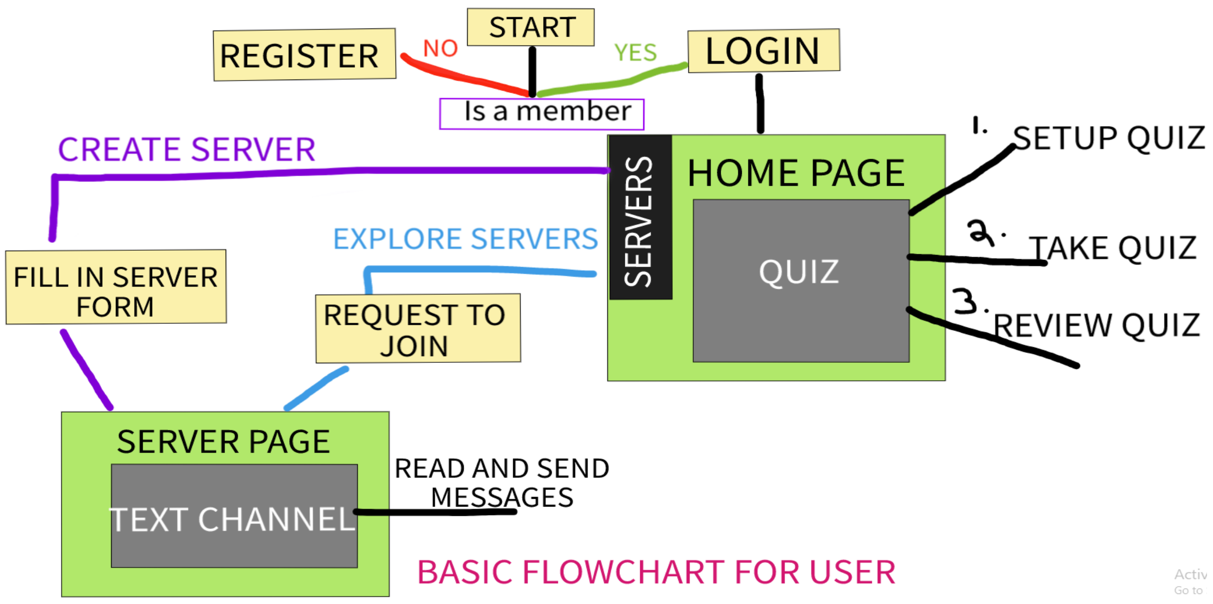


Figure 5.5: Flowchart for user

Chapter 6

Conclusion

6.1 Overview

I aimed to develop a social and learning platform, doing so there were many obstacles i had to overcome. I chose the waterfall methodology and in doing so the scope and aims for my project had to be precisely planned. In this section I will evaluate the goals I set prior to any development.

6.2 Project Aims

Below are my goals and aims I initially set:

- Create a functional three tier application
- Learn a new programming language
- Create my own API
- Create a quiz function
- Store quiz results on database
- Create an interaction hub for users
- Design a sleek website with fluid navigation

The only goal i came short of was storing the quiz results on a database, this was mainly due to time constraints. Individually developing this three tier application has helped me grow as a developer and equipped me with plenty of new skills. I have developed my first Python project, and learned the rest framework and how to build API's. Creating the quiz function made me more familiar with building complex functions in JavaScript, and calling an exterior API to an application.

6.2.1 Project Outcome

I was able to deliver a functioning application that has hit its goals and scope. Time management was an issue for me in the beginning of development; and learning a new language made this task feel overwhelming at first. Through weekly meetings and being able to discuss my project to someone else I realised I was falling behind schedule. I was able to quickly adapt by increasing productivity. This project has bettered my skills across the board from project planning to testing.

6.3 Discoveries

During early development I have noticed buggy issues with React version 18. Having automatically installed the latest version instead of a more stable and documented version made it difficult to fix errors. After countless hours of debugging I opted to rollback my react to version 17, this was more stable and documented well online. Going forward I will be more cautious if what version of technologies I am installing.

6.4 Managing a Full Stack Application

The biggest challenge I faced while developing this application was getting three separate layers of technology to communicate with another. The python community is very helpful for beginners so that allowed me to spend additional time getting more familiar with JavaScript functions. Github provided very useful for development as I could experiment with code on a previous version of my application. I have learned if a project is not managed and planned properly from the beginning it will spiral downhill. Picking the correct methodology of development is essential for successfully managing an application, if a teams methodology lacks an identity their productivity will be less significant.

6.5 Application Limitations

In further development I would like to make my application compatible with more platforms, upgrading the quiz feature, and adding direct messaging.

6.5.1 Quiz

The quiz feature could be improved, instead of users selecting quiz categories and options from an API being pulled, it would be useful to create my own API instead. Users would then be able to create their own quizzes and take their own quizzes. Quiz results could then be stored on the PostgreSQL database.

6.5.2 Messaging

While the main social purposes of the application is in chat-rooms, it would be useful to incorporate direct messaging for users. This allow users to communicate if they do not want their messages to be viewed by server members.

6.5.3 Platforms

Additional Platforms: Currently this application is only supported by browsers. I hope to deploy this application on the Android and Apple mobile iOS platforms as many users tend to check social media on their handheld device, this would be a nice feature for accessibility.

6.6 Reflection

Overall I am proud and happy with my project. Creating a three tier application was initially overwhelming and proved to be a challenging task. That being said, this year alone I have gained a lot of insight and experience towards becoming a well-rounded developer. I was originally unsure if I would hit the scope for my project as I had to learn a new programming language, before any development. I am happy I chose to do this project individually, as I have a full understanding of the system built, however if I had team members this application would see a boost in features and functionality, ultimately providing a better user experience. Overall there were some productive and unproductive weeks in development, since I had no team members the weekly meetings with my supervisor encouraged me to continue on. I have learned how important planning and research are, especially with larger projects. The precise planning and structured timelines gave me an idea of what it would be like to collaborate with a team in a professional environment. I would like to thank my supervisor, Dr. Joseph Corr for his time and thoughts on how to design a well structured application and dissertation.

Bibliography

[1] StephenKelly - ”Githublink”

[2] cumbria.ac.uk - ”Why-use-social-media”

[3] support.atlassian - ”types-of-version-control”

[4] synopsys.com - ”top-4-software-development-methodologies”

[5] practitest.com - ”black-box-vs-white-box-testing”

[6] betterprogramming.pub - ”Everything-about-django”

[7] freecodecamp.org - ”why-use-react-for-web-development”

[8] ubuntu.com - ”what-is-postgresql”

[9] sqlshack.com - ”pgadmin-postgresql-management-tool”

[10] medium.com - ”how-to-use-overleaf”