Project_3:

QEYBOARDER: A Spatio-Temporal Social Network

PROJECT TEAM 17:
RUTGER VAN HUYSSTEEN, 23052228
BERNARD OOSTHUIZEN, 21570124
KAYLAN NAIDU, 22778063
ANNA DAVIES, 21161097
HARRY ODENDAAL, 22794484
JACQUES LE ROUX, 23068574

Project report submitted in partial fulfilment of the requirements in the Faculty of Science (Natural Science) at Stellenbosch University.

DATE OF SUBMISSION
21 / 06 / 2021

COMPUTER SCIENCE 334

CONTENTS

Table of Figures

INTRODUCTION

The objective of the Project, which in turn is the problem being solved, is for the group to create a web-based social media network which focuses on user interaction by allowing them to communicate with each other via posts, groups or chat rooms. “QEYBOARDER” is designed to allow users to communicate with each other via geo-tagged posts and/or messages where users can create or join groups where they can make posts, comment on existing posts and update or edit their own previous posts. Users can also join chat rooms where they can message each other in real time.

OVERALL DESCRIPTION

When the web app is opened the user is taken to the Login page, the user may then login to their account or switch to the Register page using the navigation bar, where they can create a new account. The account information is stored in a database table in the web app so the user can log in in future. When a new account is created an access token is generated, this is stored in local storage. When the user logs in they have the choice to stay logged in by choosing the ‘remember me’ option, this prevents the user from being logged out after approximately half an hour. If chosen there is another refresh token generated with the access_token, this is also saved in local storage. When the access token expires the refresh_token is used to generate a new access token to keep the user logged in. This enables the user to stay logged in for a much longer time or until they choose to logout and would then be taken back to the Login page.

Once the user has signed into the webapp they are taken to the Posts Feed page, changing pages is done using the options on the navigation bar. All posts are displayed by default in descending time order, this can be changed to ascending time. By inputting a search term, the posts can be filtered by location range, location quantity, category, user or group using options from a dropdown menu. On the Posts Feed page, the user can choose to create a new post which accepts a title, category, location, content and a group, before generating a post which appears in the posts feed. Posts can be commented on and comments can be deleted and updated.

One of the navigation bar options is the Chat page. Here the user is able to enter a chat room and send direct messages to another user and receive replies. The Search box in the navigation bar allows the user to search the web app by groups or users and displays all those related to the search in separate tables.

The User Profile page enables the user to update their username or email, or to delete their account. The account username or email is updated by changing the entries in the account database where all the user information is stored and if the account is deleted the user’s entries in the database are cleared.

Groups can be created in the Group Feed page of the webapp, all the groups the user is a part of are listed under Your Groups on the Group Feed page. The user inputs a group name and then has the option to add other admins, update the group name, delete the group or create a post. Adding other admins allows other users to make changes to the group, like updating or deleting it. All members of a group can create posts which are displayed on the Posts Feed page.

CASE DIAGRAM

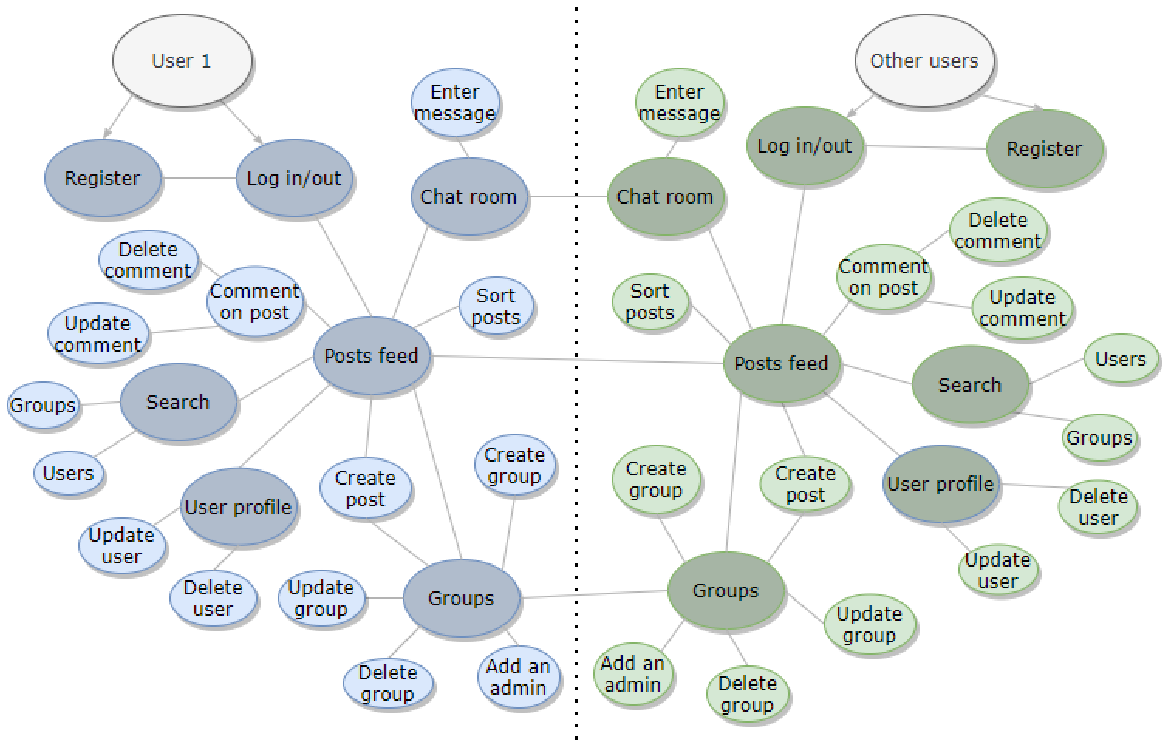

Figure 1: Case Diagram, showing the various actors and how they make use of the system

The pages in the case diagram, shown in darker colours, are all connected. The page being shown can be changed using the options on the navigation bar at the top of the screen which uses URLs to move between pages. It can be seen in the case diagram how each page of the web app connects to others and what operations are available on each one.

DATA MODELLING

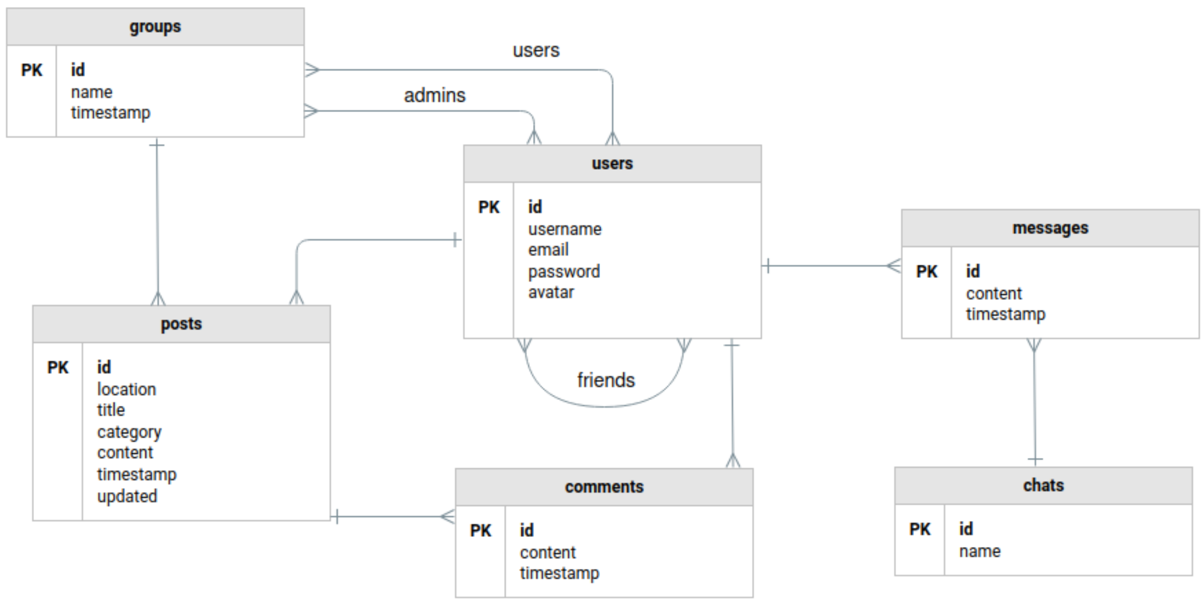

Figure 2: Entity Relationship Diagram – Showcasing the fields and relationships between tables in the database

The links with connectors indicate the one-to-many and many-to-many relationships, where in each table the ID field is the primary key.

1. Table Relationships

1.1. Groups

This table consists of one char field and one date field called name and timestamp respectively. It has two, many-to-many relationships with users where the one is for group participants and the other for group administrators.

1.2. Users

The users table contains information about each user. The username and email fields contain text and the password field contains a hash. Each user profile has an avatar image and the URL of the image is stored in the avatar field. Users are able to create posts, comments, groups and messages. There is a many-to-many field with itself, that allows users to be friends with each other.

1.3. Posts

This table consists of two char fields; title and category, one text field called content and two date fields; timestamp and updated. As well as a location field which stores the longitude and latitude of where the post was made. Due to the posts table linking to the users table with a foreign key, a one-to-many relationship exists between users and posts.

1.4. Comments

Users are able to make comments on a post, where each comment has a content text field and a timestamp date field. There is a one-to-many relationship between users and comments. Due to comments also having a foreign key that link it to posts, there is also a one-to-many relationship between posts and comments.

1.5. Messages

Messages are created by users in chat rooms. This means there is a one-to-many relationship between users and messages and between chats and messages. Each message has a content text field and a timestamp date field.

1.6. Chats

Each chat represents a room where participants can join and chat in real-time. This table contains a char field called name.

OPERATING ENVIRONMENT

For the client-side system, the React Web app framework was used. React is a library for JavaScript that is used for creating user interfaces and allows for faster development when compared to used css and html. The Axios library was used to communicate with the API, and it can handle http requests and responses. It can also be used to receive data from the API, allowing the React application to work with that data.

The Django web framework was utilised for the server-side API. Django is a free, open-source Python Web framework that is designed to be fast, secure and scalable. Django has its own Object Relational Mapper (ORM) that allows for efficient filtering and ordering of data with the connected database. Django also has many packages that were useful for the web app, such as the GeoDjango package as well as the django-rest-framework (Django Software Foundation, 2021).

Django has great integration with PostgreSQL, which was the database used for the web app. PostgreSQL is a free, open-source Relational Database Management System (RDBMS) that has a wide variety of extensions. The docker image "postgis/postgis" is run using docker-compose to set up the PostgreSQL database. A volume is specified in the .yml file, so that the data is persistent. PostGIS is a spatial database extender for PostgreSQL, that allows full utilisation of the geographical data provided by the GeoDjango package used in the server-side API.

A "redis:latest" docker image was used for the chat room functionality. Redis acted as an in-memory database to store messages on the server and send them back to the clients. This Web app was designed to mainly run on a desktop or notebook browser.

SOLUTION

I. CLIENT

1. High-level description of the design pattern for the frontend

The individual pages are split into separate components, where components which are shared between these pages are also abstracted further into separate components so that it can be reused on multiple pages while only having to create it once. Axios is used to make requests to the backend, while a combination of redux, react-query and react's usestate is used to control the state of application. Local storage is used to store the access_token and refresh_token (if remember me box was ticked), where these tokens are then used to authenticate the requests which are made using axios.

2. Why use React

React has a high multiplier and is the most widely used frontend framework for Javascript and thus has the biggest community surrounding, which means that there are a lot of tutorials on how to use React. React is also faster to develop compared to vanilla javascript/css and html since a lot of the complexities have been abstracted and thus more can be done while writing less code. An example of this is that different parts of a page can be broken up into what is known as components, where these components can then be reused making total code written less and thus making debugging an easier task, since the component only has to be fixed at one place.

II. API

1. High-level description of the architecture

The Model-View-Controller (MVC) design pattern was used during the development of the application (Sinhal, 2017). In this pattern the model represents how the data is stored and the possible manipulations. The model contains the data that is displayed to the user in the view part.

In the MVC pattern, the view represents the user interface where the user sees data and interacts with the application. The controller part processes the requests and sends back the response. It controls the flow of information. There is logic that decides what data must be stored/fetched/changed and what must be sent back to the user. It connects the model and view.

2. Why use Django

The Django web framework was used to develop the API. There are many advantages when using Django (Hansen, 2017). Firstly, it is extremely fast. It helps developers to go from concept to a live application as fast as possible. This means it takes less time to build the application, resulting in lower development costs.

Django comes with built-in functionality and packages which eases the development. The object relational mapper (ORM) makes it easy to process and save data to the database. It automatically sanitizes database queries through the ORM. User authentication and password hashing is also included. It is easy to configure and connect SQL databases to Django. Django also comes with built-in functionality for location-based applications in the form of a package called GeoDjango. GeoDjango is designed to work well with PostgreSQL and the PostGIS extension used to store geographical data. This contributed to the reason why we chose Django along with PostgreSQL.

Another advantage is the rich library of Python packages that are available for use in Django. For example, there is a package called django-rest-framework which makes it easy to develop an application with CRUD routes. Django also provides good security measures and prevents developers from making many common security mistakes. It has protections against SQL injection, cross site scripting, cross site request forgery and clickjacking. The passwords are not saved in the database and thus only a hash of the password is saved to improve security.
Django has a built-in package called channels, which in turn makes it easy to work with the WebSocket protocol.

This eases the development of a real-time chat application. Another vital reason why our team chose Django is because some of our team members already had experience with Django and this improved our overall efficiency.

3. How the stack is implemented

Django follows the Model-View-Controller (MVC) pattern but uses different terminology. The view part is called template in Django and the controller part is called view. It is shown in the following diagram:

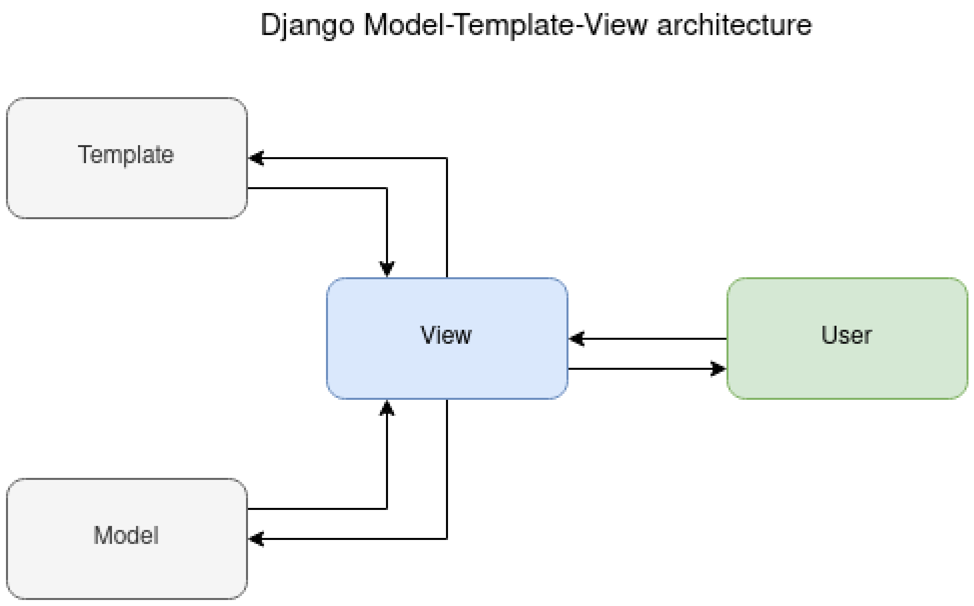

Figure 3: Django Model-Template View Architecture

Model represents the data that is required to display the view. Django comes with an Object Relational Mapper (ORM). This allows you to define how data is stored and manipulated using object oriented programming principles. Django uses something called templates for the view part of the MVC pattern and it uses views for the controller part. However, this application only required an API from Django, so the template part of Django was not used in this project. The user interface was handled using React.js.

Four tables were created by writing classes for each table in the Django model's file. The fields for each table were defined as properties in these classes. The database schema is set up in this models file. The Django views (controller part) interact with the models in order to store, retrieve and change data. The Django view processes the JSON request from the front-end and converts it into a format that Django can validate and be used to manipulate the database. This is done with serializers which the django-rest-framework package provides. It validates the incoming data from the front-end and along with the ORM updates/creates/retrieves data in the database. After this the Django view (controller) converts the data to JSON format and sends the response back to the front-end indicating success or failure along with data or validation errors.

There are permissions on each Django view, so it requires the user to be authenticated otherwise the data is read-only. The Django views are all connected to specific URLs relating to the different tables of data. This makes it easy for the front-end to send CRUD requests. A user has to login to use this application. When the user logs in, an encrypted token is sent from the back-end to the front-end. This access token is then sent as an authorization header each time the user makes a request to the back-end. This is known as JSON Web Token authentication.

The real-time chat application also has different controller parts. It works differently than with the HTTP protocol. When the WebSocket protocol is used the connection, stays open so that the server can send and receive messages from clients. Initially the client-side sends a WebSocket request to a URL registered on the back-end. The chat name is sent to the back-end and then it opens a WebSocket connection to send and receive messages to anyone else who connects to this room. This means that when a user sends a message on this channel, it will be broadcasted to all users in the chat room.

The front-end also has an open connection through the WebSocket API. Therefore, messages can be sent and received between users, and displayed without refreshing the browser. This allows people to have a real-time chat.

III. DATABASE

For the project PostgreSQL was used for the database. PostgreSQL is a free open-source relational database management system (RDBMS).

1. Why use PostgreSQL

This database was selected because it had a good multiplier on the marksheet, and it did not seem too hard to learn how to use. PostgreSQL is also very good at writing a large amount of data and with being active for over 30 years, PostgreSQL is known to be very reliable. It is also supported with a lot of extensions, which makes it a great database to use for a variety of different applications. It is also scalable which means that even if the application gets an influx of users, PostgreSQL could still be used.

Django has built-in support for PostgreSQL, so it is easy to configure and connect a Django application with this database system. GeoDjango fully supports PostgreSQL with the PostGIS extension. This means there won’t be compatibility problems when working with geographical data. It is easy to install and use the geographical database extension. Another advantage is that there is already a docker image for running PostgreSQL with PostGIS. This makes the installation and configuration phase go much faster.

2. How is the Database implemented?

The configuration settings of the Django application are set up in order to connect to a local PostgreSQL server. A docker image of PostgreSQL with the PostGIS extension is run with docker-compose before the Django server gets started. This is all done locally, the application never went onto a production server.

If this app goes into production, the following setup could be used. Nginx can be used as the web server. It can take the requests from clients and direct them to the Django application. The Django application is run through an ASGI server. ASGI is required, instead of using WSGI, because the real-time chat requires asynchronous functionality. WSGI cannot handle asynchronous requests.

VIDEO SUBMISSION

Follow the link below to view the demo video for the social network platform, QEYBOARDER:

https://youtu.be/wv4rLPrz3AI

REFERENCES

Django Software Foundation. “Django.” *Django overview | Django*, 2021, https://www.djangoproject.com/start/overview/. Accessed 16 June 2021.

Hansen, Stevenn. “Advantages and Disadvantages of using Django.” *Advantages and Disadvantages of using Django | Hacker Noon*, 23 May 2017, https://hackernoon.com/advantages-and-disadvantages-of-django-499b1e20a2c5. Accessed 16 June 2021.

Sinhal, Ankit. “MVC, MVP and MVVM Design Pattern.” *MVC, MVP and MVVM Design Pattern. MVC, MVP, and MVVM are three popular… | by Ankit Sinhal | Medium*, 3 January 2017, https://medium.com/@ankit.sinhal/mvc-mvp-and-mvvm-design-pattern-6e169567bbad. Accessed 16 June 2021.

APPENDICES

APPENDIX A

PROJECT SPECIFICATIONS

Summary

Due date: 14 June @ 23:59 & Report due on 21 June @ 23:59
Deliverables: A written report and project demo with video
The Objective:

a. Implement a data model on a SQL-driven relational database
b. Create a robust web API to serve a client-side application
c. Create client software in the form of a web or mobile application

Implementation Description

You are required to create a web-based social network that is focused on users interacting with one another on geo-tagged posts and/or messages. Users must be able to interact with one another by **creating and joining groups** on which they make posts, and comment on these posts. At least one of the aforementioned interactions must be geo-tagged but all must have timestamps. These groups are fully sovereign i.e. groups run themselves. The user who creates the group is the group admin and they are the only one who may edit and delete it.

It is a **bonus functionality** that an admin may give another user admin permission.

Direct messages are another **bonus mark opportunity**. These messages should be in real-time which means that they will primarily be peer-to-peer and this can be achieved using frameworks such as WebRTC. One may store posts and group messages server-side.

Posts from all groups will be **centralized in a single feed**. In order for users to find posts most important to them, posts must have at least 3 categories. A user should be able to filter their feed by location, time, category, user, and group. MySQL 5.7 has built-in GIS support but some database technologies have useful extensions for this e.g. Spatialite, PostGIS. This should be considered during planning.

To acquire the upper echelon of marks, students must implement this project using a handful technologies and techniques that are not formally taught in this course. The teaching assistants will be able to assist with learning a technology of a student's choice within the limitations of the list of what each assistant knows. A comprehensive list of what are considered core requirements and what are considered bonus can be found under the heading "Marking Rubric".

Choosing your tech stack

This project encourages students to challenge themselves and they will be rewarded with credit appropriately. That being said, those who wish to confine themselves within the realms of what they have already learnt can do so and pass the project comfortably. Keep in mind that your group should choose their solution stack and delegate work such that each member plays into their strengths. Requirements will be segregated per solution and a multiplier will be added for the marks achieved in each solution i.e. if one made a mobile application but used Flask and SQLite, their client-side solution mark will be multiplied by 1.5 but server-side and database marks will be multiplied by 1.