Spatio-temporal social Network

Introduction

The problem we are solving.

The problem we are set to solve is to create a Spatio-temporal social network i.e., a social network for which both location and time is relevant to its function. It being a social network, the users must be able to interact with one another by joining groups, making friends and communicating over topics of mutual interests. The user must also be able to perform basic maintenance over their own profile they set on the social network.

Our solution

Welcome to GeoNet! Geonet is a basic browser based social network where anyone can sign up using only their email, setting a password and choosing a unique username. From there they can navigate to the global feed and view all the posts made by other users. On each post in the feed, they can view the location from which the post was made, all the relevant categories the poster tagged his/her post with, the date and time and most importantly the post itself. The user can also navigate to the comment section of the post by viewing the post in full. There the user can see other comments other users have made and/or make a comment on the post themselves.

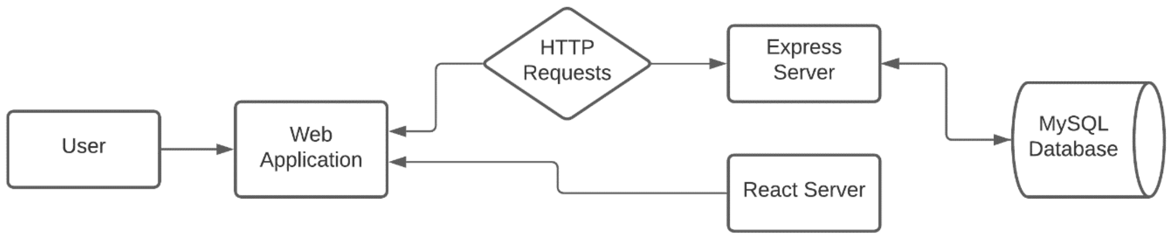
The user can also navigate to the groups page from which the user can view all the groups other users have made or create a new group. Once on a group page the user can create a post to that group. Each post is automatically tagged with the user's current location and to make a post requires three categories to be appended to the post. Once a post is made, it is both visible to the group on the group page and on the global feed. The user can also head over to a group searching page where the user can easily find specific groups. The user can also search for specific users and add or remove them to the user's friend list.

If the user so chooses, he/she can go to their profile page and perform various functions such as setting a profile picture, changing their username, changing their email address, updating their password or deleting their account.

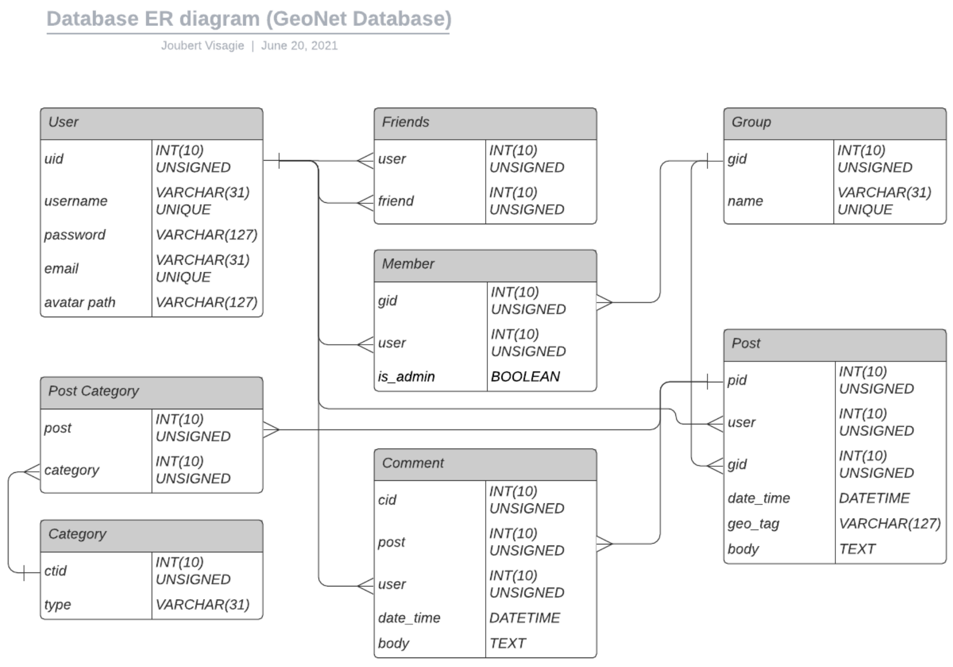
Overall description

How the system works

The following diagram shows a rough overview of how the system interacts with each other:



Data Modelling



This is an ERD that models the database structure.

Operating environment

For the client application, we made use of React. The React server serves the client application from which the user can interact with a highly functional application from his/her browser. The client can then interact with the backend API with the use of standard HTTP requests. For the API we made a server using Express.js, which is a web application framework that runs on Node.js. Node.js is a JavaScript runtime that makes use of an event-driven non-blocking input/output model, meaning that after the server makes a call to the API, it moves on to the next API call while waiting for the data to be received. The API serves the client in several ways, either retrieving some data from the database or by serving profile pictures from its storage. For the database, MySQL was used.

The software used for deploying the API and the client makes use of Node.js and the Node Package Manager (NPM). There is then of course MySQL for the database. A computer installed with this software will be able to host the API and client servers. For the purposes of the project, the API and client servers are hosted on the same device together with the database. From the host machine, the client application can be accessed via the device or on the local network. In practice, the API and client can be deployed separately and the application accessed over the internet.

Solution

Design pattern

React employs the MVVM design pattern: Model-View- ViewModel. In the case of our application, we may see the data relevant to the social network as our Model. This includes the posts, comments, user data and metadata. This data model contains the information that we wish to serve to the users of our application. To serve this data we make use of a ViewModel which both contain code to display data and handle user inputs. User inputs are handled by the ViewModel to either update the Model, in this case data relevant to the social network, or to serve more data from our Model, like showing posts or other users. The View in the design pattern is simply the UI elements, in our case JSX together with UI libraries to create a stylised user interface. If we wish to explain this by means of an example, we can produce the scenario where we want to display and comment on a post made on the social network. From our Model, i.e., the database and API, a React component can request the post's data. The React component, which can be seen as our ViewModel, can then display the post on the View by making use of UI libraries and JSX and keep the state of the current post being displayed. If the user then wants to make a comment, the React component can then take the user input and, together with the stored state (post id, the logged-in user), alter the Model by sending it the comment with code inside of the component.

The backend API makes use of the Model View Presenter (MVP) design pattern. For our application, the Model can be defined as the data stored in the database and the methods used to either manipulate or access this data. The View in our case would be the client, which receives results from the Presenter based on the user's input and renders the data to the user. The View is free to present the data to the user in whichever way it wishes. Finally, the Presenter is the Node.js server and the routes created using the Express.js framework. The server receives a request from the client via the routes defined and works with the Model to return a result to the View.

High level description: Client

Architecture: React

React is a JavaScript bases library used for creating interactive user interfaces. The main selling point of using a UI library such as React is that it can keep state, meaning that depending on how the user interacts with the application its state may change independently from an API serving new data to the client. This allows for a client application that may imitate the behaviour of an application that is running natively on the machine. The client application can run code that dynamically changes the output to the user, sometimes without interfacing with the API. Another feature of React is that it can be used in combination with a wide variety of packages that each build on React's functionality to add more specific features for creating an interactive client application.

We chose React as our frontend due to its popularity, and therefore its thorough documentation and help guides online. We also chose React because we deemed it the simplest way to implement a more complex web application than that of, say, Flask and Python. Relative to the effort that would have been required to implement an as complex application in Flask and HTML, React was an obvious choice to create our web application. As such, development time was our deciding factor for using React.

Implementation: React

React encourages the use of components; breaking up functional parts of the application into units that can be reused or used multiple times at once. As in our implementation we broke up most of the parts of our application into components. Each component would then be responsible for keeping the state and serving a different purpose.

We would have several components in a hierarchical structure each nested within each other. If we consider this by means of an example, we may think of the main application having a child component, like in our case, a page that displays a list of posts and each post on the list having the component to display location, where the location displayed is dependent on the post. Each of these being separate components they function with some data passed on to them by their parent, with data that are being retrieved via API calls and maybe statically as well.

Additional packages

Axios

Axios is the library used by the client to handle standard HTTP requests. It interacts with React by asynchronously updating the state and having React handle the changes accordingly. It handles the sending and reception of data and error catching.

Material-UI

Material-UI, as the name suggests, is a React library used to provide a stylish user interface with the use of prewritten CSS and JavaScript. The use of a UI library means that all the styling of the web application is uniform and easy to maintain without writing your own CSS or JavaScript. The UI elements that Material-UI provides are used to display relevant information regarding the social network and to bind user inputs to certain JavaScript functions.

High level description: API

Architecture: Express

Express.js is a free and open-source JavaScript based framework for Node.js which is used to create web applications and APIs. Express provides a robust set of features for web and mobile applications. Express allows us to use and create various middleware's that add functionality to our API endpoints. This middleware is normally in the form of plugins that can be easily installed using the node package manager.

Among the reasons Express.js was chosen was to have a uniform JavaScript codebase, which would make the debugging of code easier. Its popularity and thorough documentation, in conjunction with the multitude of online resources and tutorials was also a deciding factor in choosing to use Express. Additionally, the ease of use of the Express framework was a driving force behind our adoption of it in our project.

Additional packages:

JSON Web Token (JWT)

A JWT is a token that is digitally signed and provides a way for two parties to communicate securely. The token may contain any information of one’s choosing, which in this case was the authorisation information (specifically the user ID). The server generates a secret which is used to sign the token sent to the client, as well as verify the token sent from the client to the server. Additionally, the token is signed using the HMAC SHA256 algorithm. The signing and verification of the token is handled by the JavaScript library *jsonwebtoken*.

Bcrypt

Bcrypt is a JavaScript library whose purpose is to hash passwords and verify hashed passwords stored in the database. The library is based on the Blowfish cipher and makes use of a cryptographic salt to hash the password. This ensures that the hashed passwords will always be unique, even if different users decide to use the same original password. The advantage of storing hashed passwords in the database is that if the server is compromised, attackers cannot make use of the passwords obtained to gain access to a user’s account.

Knex

Knex is a SQL query builder for Node.js that acts as an intermediary between the MySQL database and the Objection ORM. It executes the queries generated by the ORM, and interfaces with the database to obtain the relevant information requested in the query. This minimises the need to write raw SQL strings which can lead to vulnerabilities such as SQL injections.

Objection

Objection is an ORM for Node.js that is built on Knex. It allows for the database tables to be defined as models in JavaScript, as well as for relational mappings between tables to be defined in the models. The models can also be customised to have additional properties that may not be present in the database schema. As mentioned earlier, the ORM interfaces with the database via Knex, which executes the queries generated by the ORM.

MySQL

To access the MySQL database with Node.js, a MySQL driver is required. The driver is used by Knex to establish a connection to the database, following which manipulation and retrieval of information are done.

Cors

Cors is node.js middleware that allows for cross-origin resource sharing of data from the server. Cross-Origin Resource Sharing (CORS) is an HTTP-header based mechanism that allows a server to indicate any other origins than its own from which a browser should permit loading of resources. This allows many clients to access resources hosted on the server and make requests to the server.

Multer

Multer is node.js middleware for handling multipart form data requests, which are primarily used for file uploads. Multer allows us to read files from a form as well as text data. Multer can also be used to write files to local storage, and filter which files can be uploaded, and which should be skipped. We used Multer to handle the uploading and saving of a user's profile picture on the server.

High level description: Database

The database is a MySQL database that consists of eight tables. The database uses primary and foreign keys to establish relationships between the tables. In the ERD diagram previously shown in the Data Modelling section, the edges(lines) show the relationships between the tables using keys. From the ERD we can also see the database is modelled in the third normal form, because every non-prime field of a table is dependent on the primary key of that table. This eliminates duplication and reduces data anomalies.

Video

The project demo video can be found here.