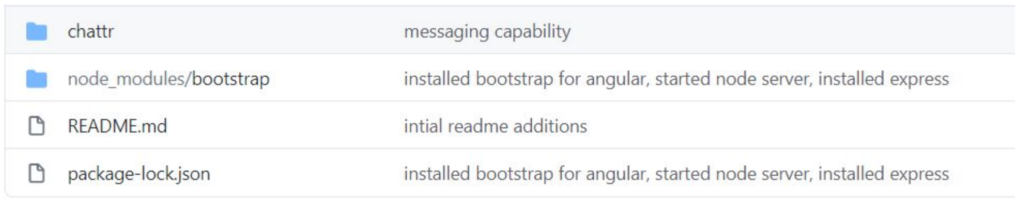
3813ICT Assignment Phase 1
Chattr: an online chat application
Conor Gould
s5007332
Trimester 2, 2020

Version control

Github repository: https://github.com/conorg000/3813ict

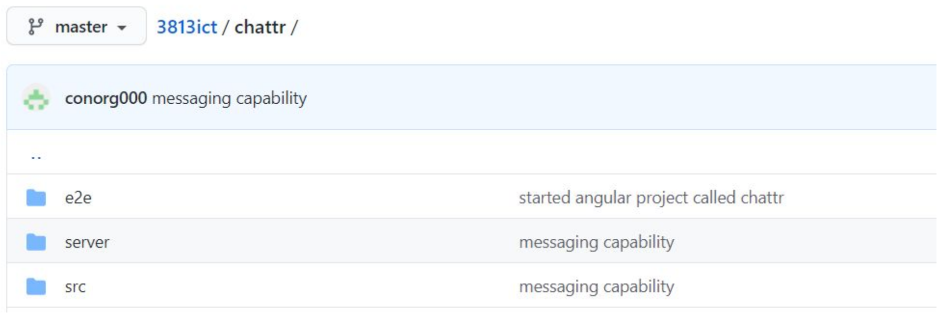
For the development of Chattr, Github was used as the distributed version control system. This helped the project in two ways. Firstly, it acted as a backup of all work done, and made it possible to roll back to previous versions if a new feature ended up behaving unexpectedly. Secondly, it allowed for the hypothetical case where several developers might work on the project simultaneously. While there was only one developer in this case, using Github would normally make it possible for different developers to work on separate components of the project and then later merge their contributions together.

The structure of the git repository is demonstrated below. In the outermost directory, there is a README file, containing information about the project. There is then a directory for the Angular project, "chattr".



chattr	messaging capability
node_modules/bootstrap	installed bootstrap for angular, started node server, installed express
README.md	intial readme additions
package-lock.json	installed bootstrap for angular, started node server, installed express

Inside the "chattr" project, there are plenty of files, most of which are automatically generated by Angular. The main directories used in the project are "server" and "src". In a nutshell, "server" contains the back-end logic of the web application and "src" contains the front-end components of the application.



master 3813ict / chattr /

conorg000 messaging capability

..	
e2e	started angular project called chattr
server	messaging capability
src	messaging capability

For each feature that was introduced, the changes to files in the local repository were committed and pushed to the master remote repository. A detailed description of the changes was provided, making it easy to roll-back to the right version of the code if necessary. A “gitignore” file meant that large folders containing node packages could be exempt from being committed to the repository, as this would have taken a long time. Even though there was only one developer working on the project, for the sake of developing a good habit, a git pull command was run each time work recommenced on the local repository. This checked for any changes to the remote version which may have taken place.

Data structures

On the server side, data is stored in a serialised JSON file called “database.json”. There is a key containing data on “users”, and another key containing data on “groups”. The “user” objects are stored in an array.

Each “user” object contains several fields: “username”, “email”, “id”, “pwd”, “role” (“superadmin” or “groupadmin”) and “groups”. The “groups” field contains an array of objects, each one containing “groupname” and “rooms” - an array of room names which the “user” belongs to.

In the “groups” section of the data, there is an object for each group. The object contains “groupname”, “rooms” and “groupassis”. The “rooms” field contains an array of rooms and their corresponding chat history. Each room is defined by its “roomname” and “history”, where “history” is an array of “message” objects. Each “message” object has a “user” string and a “message” string. The “groupassis” field contains an array of usernames who have been assigned as assistants for that group.

On the client side, there is a “user” class. It contains fields including “email”, “username”, “id”, “role”, “pwd”, “valid” - all of which are strings, except for “valid” which is a boolean.

Angular architecture

Components

Three components were created: Account, Chat, Login.

The Chat component was designed to host the communications for a group’s room. For the given user, it fetches the groups that the user is a part of and the rooms that they’ve joined. With those details displayed, it allows the user to choose a room and view the chat. It communicates with the AuthService. On initialisation, if a user is logged in, it calls the userGroups and groupData functions and routes to retrieve data on the current user’s group and rooms. When a room is selected and clicked, the seeChat function fires. This then displays the chat history for the selected room. If the user wants to send a message, the submit button triggers the sendMsg function which ends up writing the new message to the JSON file.

The Login component handles authentication. It provides a form for the user to enter their details and then logs the user in if the details are correct. It communicates with the AuthService. When a user submits their details, the itemClicked function is triggered. This verifies if the user exists or not.

The Account component was designed to be a place for users to edit their details, and for admins to edit users and groups. Unfortunately, this didn't get completed in this phase of the assignment. The plan would be to have a form which allows the administrator to add/remove users to/from groups and rooms. The new configuration would then be saved to the "database.json".

Services and Routes

The AuthService acts as a gateway to the key API routes used by the various components. It contains four routes:

- http://localhost:3000/api/auth for the login function
- http://localhost:3000/api/userdata for the userData function
- http://localhost:3000/api/groupdata for the groupData function
- http://localhost:3000/api/sendmsg for the sendMsg function

Node server architecture

Modules and Functions

In the api-login.js file, there is a post function for each API route in the AuthService.

- "/api/auth" is an authentication procedure called by the Login component. It loops through the existing users in "database.json" to see if any of the user objects match the submitted email and password combination.
- "/api/userdata" is a method which retrieves the group data for a logged in user (which groups they are signed up for, and which rooms).
- "/api/groupdata" is a method which retrieves all the group information and room chat history.
- "/api/sendmsg" is a method which adds a message object to the existing chat history for a room. It takes the submitted message and the sender's username and pushes them to the array of messages in the "database.json".

Modules used in server.js on the server side include Express, Body Parser, Path, Http, and Cors.

Files

There is a single file containing all of the initial serialised JSON data, “database.json”. The JSON file is read by API routes and then parsed to load the necessary data. If changes are made which require updating the data, the changes are made and then saved over the existing file.

Client-server architecture

The responsibilities of both the client and the server are divided in such a way so as to let the server handle the update and retrieval of data, while the client acts as the portal through which requests go to the server. More specifically, the client contains functions which interface with the application’s front-end. Certain actions trigger functions, which are then redirected via the AuthService to their respective routes on the back-end. The back-end, or server, contains logic which returns the data requested by the client.

Client-server interaction

As described earlier, actions triggered on the client side lead the server to supply, modify or update the data. This happens by reading and writing to the serialised JSON file “database.json” with functions like

Angular components are updated by having their values bound to values variables within the respective component. In input forms, for example, the “banana in a box” notation is used to link field values to variables, which can then pass on the input. Likewise, to display a variable in the HTML front-end, curly braces are used to embed the updateable variable in the web page.