# Assignment1\_frameworks

Zane Stevens -- s5072430

## Git Organisation

Git Organisation was simple and basic. Branches weren't used for any part of the project due to my personal proficiency with Git and Project development in general and have just been using the Master branch for all commits and pushes. Commits were made at the end of every development session with semi-detailed messages, though these are detailed in a way that I would understand each commit (which is bad practice).

Data Structures

Two .JSON files were used to manage the data of the website:

- One for User Data (name, status, email, password)

- One for Group Data (name, channels, channel members, channel name, group assists)

## Angular Architecture

The architecture was kept simple:

- Components for Login, User-Edit, User-Delete, Group/Channel-Edit/Delete and Chat (Main page)

- Services for Data Collection and Saving (root(should have named it something better)) and Sockets

- Models for Server, Listening for requests and sockets

- Routes were used in the navigation of all pages using the RouterModule import

## Node Server Architecture

As the Angular Architecture, the Node architecture was kept simple:

- Modules included the basic Express imports:

- Express, Cross-Origin-Requests (CORS), HTTP (for http requests through the node server)

As well as additional modules to handle data reading, requests, saving and editing

- Body Parser, File System (fs) and path

Also some addition Socket.io imports (I need to work on chat now, got to save time in the future)

- Io using HTTP

- Functions included Listen() which handles starting and listening for server requests and connect() which connects using Socket.io and handles socket implementation

- Beyond the two .JSON files, listen.js (which contained the Listen() function) and the socket.js (which contains the connect() function) where the only two files used including the original server.js which contained the API requests

## Division of Responsibilities

The division of duties between client and server was simple, and only required two (app.get and app.post(/postData) were just a test function) calls. The /saveData was used to save data to the users.JSON file and the /saveGroup saves data to the groups.JSON file.

On the client side, two http.get functions were used to retrieve data through the root.service file, allowing the client to use the data.

## Routes

Four routes (used in the above calls) were utilised, including:

- getAPIData(): Gets the user data, returning it as an Observable to be manipulated by client side functions

- getGroupData(): Get the group data, returning it as an Observable to be manipulated by client side functions

- saveAPIData(): Calls the /saveData route in the server, saving the data to the JSON file

- saveAPIGroupData(): Calls the /saveGroup route in the server, saving the data to the JSON file

## Details of Interaction

The interaction between server and client were basic:

- Sockets were used (but not required), where the server called the sockets whilst the client displayed and returned variables to be used by the server

- getData functions were called to the server, so that the server returns an Observable that will be manipulated by the client

- postData (saveData) functions were called using routes, so that the server recieves the request, saves the data to the JSON files and returns a "success" log to the server which is represented on the client console.log.