# Assignment2\_frameworks

Zane Stevens -- s5072430

## Git Organisation

Git Organisation was simple due to the solo development nature of the project. Branches weren't used for any part of the project and the Master branch was used for all commits and pushes. Commits were made at the end of every development session with semi-detailed messages, which detailed what changes were made.

## Data Structures

A MongoDB database was used for data structuring. This consisted of One "assignment" database with two collections,

* Users
* Groups

## Angular Architecture

The architecture from the previous version of this project was kept, which includes:

* Components for Login, User-Edit, User-Delete, Group/Channel-Edit/Delete and Chat (Main page)
* Services for Database CRUD operations and Sockets, or rather calling the CRUD operations through the server URL's
* 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 as the previous project structured them:

* 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 Though it is worth noting that these were redundancies kept from the previous project. Also some addition Socket.io imports
  + Io using HTTP
  + MongoDB to access the database, as well as MongoClient to run these commands through
* Functions included Listen() which handles starting and listening for server requests and connect() which connects using Socket.io and handles socket implementation

## Division of Responsibilities

The division of duties between client and server was extensive. The client side Angular handled none of the Database CRUD operations, rather these operations were handled by the server after being called by the client side. The data created for this project technically doesn't exist on the server or client, rather this data is created by the addData.js file. This file cannot be accessed by the client side and was only used to load data into the database.

## Routes

Multiple routes were used, with each route accessing only one .js file that would handle CRUD operations:

* require('./routes/getGroups.js'): Retrieve (find({}) on the Group Collection) all group data from the database
* require('./routes/getUsers.js'): Retrieve (find({}) on the User Collection) all user data from the database
* require('./routes/addData.js'): Create the data used for the project.
* require('./routes/addUser.js'): Insert a new User into the user collection
* require('./routes/addGroup.js'): Insert a new Group into the user collection
* require('./routes/editUser.js'): Edits a Users' data before sending it to the database
* require('./routes/deleteUser.js'): Deletes a users' data
* require('./routes/deleteGroup.js'): Delete a group's data
* require('./routes/addUsertoChannel.js'): Add a user to a group.ChannelUsers array before updating the group that was edited
* require('./routes/updateGroups.js'): Blanket function used to save all data changed in the Groups collection

## Details of Interaction

The interaction between server and client was sparse to reduce possible security issues:

* Sockets were used, where the server called the sockets whilst the client displayed and returned variables to be used by the server
* Beyond sockets, interaction was limited to actions between the server and database which could only be called through the client.
* Changes to data are displayed/updated every time an action is performed, usually through a refreshing of the page (running ngOnInit again).