MEANChat

Running the Application

Currently, to run there is a requirement of running the frontend and backend separately.

To run the angular project -> run `ng serve --open` from the root directory of the project.

To run the node.js backend-> run `cd server | node server.js` from the root directory of the project.

Testing

Integration Tests

The integration tests have been implemented using the `Mocha` and `Chai` libraries. To run the node.js integration tests, navigate into /server, and run `npm test`, this will run the predefined test script in package.json.

Angular Unit Tests

Unit testing inside of Angular was completed using 'Karma'. To perform this testing, from within the root application directory, run `ng test`.

End-to-End Testing

The end-to-end testing has been performed using the protractor API and Jasmine Library. To run these tests, from the root of the application run `ng e2e`, with an optional `-p` flag if you wish to specify a particular port.

Git Structure

The directory of the git repository follows a standard layout, with the majority of it representing a standard angular project, and the node server and API source code found in /MEANchat/server.

Throughout development, the approach of running 2 separate servers, 1 for angular and 1 for node was favoured, it led to a more accurate testing environment, and was faster to refresh following changes.

Data Structures

For this project, 3 different data structures have been defined: `User`, `Channel` and `Message`.

User:

```
{
    _id: ObjectID(),
    Username: string,
    Password: string,
    Email: string,
    ofGroupAdminRole: boolean,
    ofGroupAssisRole: boolean,
    groupList: [],
    adminGroupList: [],
    groupChannels: [],
    profilePicLocation: string
}
```

Channel:

```
{
    _id: ObjectID(),
    Group: string,
    Channel: string,
    User: string,
    messages: [{Message}]
}
```

Message:

```
{
    creatorName: string,
    creatorImg: string,
    content: string,
    createdAt: string
}
```

The User object represents the majority of information about a user, including their relations to groups and channels, as well as roles, the Channel object is mainly to store a Channel with it's associated Messages, and the Message object is the chat that has been sent to a Channel. Since the Message can be either text or an image, images have been converted to base64 and stored as strings.

Angular Architecture

Components

Component	Purpose
`Nav`	The navigation bar at the top of every page
`Login`	The login page found at root directory
`Home`	A user’s home screen, where the login redirects following success
`Group`	The group homepage, where clicking on a group on home will redirect to
`Channel`	The child component inside group, that reflects the content of the current channel
`Image`	The component used if a user wishes to change their profile picture

Services

Service	Purpose
`AuthService`	The AuthService handles all user authentication requests
`GroupService`	The GroupService handles all group related and channel related requests
`SocketService`	The SocketService is used for all requests related to Sockets.io

Routes

Route	Destination
`/`	The root directory
`Home`	The home page, redirected to following login
`Group`	The group page, redirected to after clicking on a group, contains channel component
`Image`	The profile picture upload page, included in the nav bar

Node Server Architecture

The general backend is run by the `server.js` file located in the /server folder. All the routes are broken up into a module each, in their own separate file in /server/routes.

```
MEANchat/Server
        \server.js
              /routes
                \channelDB.js
                \groupDB.js
                \userDB.js
                \users-channels.js
            /testing
          \test.js
```

Functions:

All files inside the routes folder have been grouped by object. In the server.js, the only other functions used are applying middleware, requiring modules, and running the server.

Responsibilities between frontend and REST API

Angular frontend: The angular application is the GUI, and a representation of data at the current state. Typically Angular has the ability to do small amounts of work with data, should no API exist, however since the project has an API, all data responsibilities have been delgated to the API/Database.

Node.js Backend:

The Node.js Server has been designed to be a REST API. All data processing has been done in the backend, the frontend simply sends a request to the backend, the backend makes the change in the Mongo Database, and returns the new result.

REST API Routes

All server-side endpoints have a request object coming in, and a response object returned.

Route	Request	Response	Purpose
`POST /api/adduser`	All fields required to create a User	The created User	Add new User to the system
`POST /api/removeuser`	User to be removed	Response with status	Remove User from the system
`POST /api/auth`	Request of user credentials	The authenticated User	Used when a User wishes to gain access
`POST /api/uploadimage`	FormData(); containing an image	Status of upload	Upload a User's image
`POST /api/getuserimage`	User	Name of User's image file	To serve the correct image URL
`GET /api/getallusers`	NIL	Array of Users	Get full list of Users
`POST /api/addgroup`	New group	Status	Add new group to the system
`POST /api/removegroup`	Group to be removed	Updated User	Remove a group from the system
`POST /api/getgroups`	Current Authenticated User	User's list of groups	Get all of User's groups
`POST /api/addchannel`	New channel	Updated User	Add new channel to current group
`POST /api/removechannel`	Channel to be removed	Updated User	Remove channel from current group
`POST /api/adduserchannel`	Channel and User to add	Updated User	Add User to current channel
`POST /api/removeuserchannel`	Channel and User to remove	Updated User	Remove User from current channel
`POST /api/getchannel`	Selected Channel string	Full Channel Object	Get channel's full data
`POST /api/updatechannel`	New Message Object	Status	Add new message to persistent data

Client<->Server interactions

Upon a request sent from the client, the server receives this request and processes, according to what route it was posted to. Usually, depending on the route, this involves a MongoDB request, and when that is finished, the result is posted back to the client. Should it be required, the client can now refresh the sessionStore with the new data, and refresh the local client variables to be pointing at the new values of the SessionStore/updated values, which will change the angular components if necessary.