Documentation for Software Frameworks Assignment Phase 1 (3813ICT)

Git

Git Layout

GitHub was used as the Version Control System (VCS) for this project, the link to the repository can be found below.

https://github.com/kevinpho1997/SoftwareFrameworksAssignment

At the root of the Git repository is the node modules for the Angular app and its node modules. In the Angular app is the src folder containing the Angular components, HTML and CSS. The Angular app folder also contains the node.js server that handles HTTP requests.

Version Control Methodology

Throughout development, I tried my best to only commit working code to ensure that the app compiles if a revert to a previous version. This was particularly with main, as I would work on main directly when making small changes but I would use branches for more complex tasks. The branches allow testing of new functionality without affecting main, which would then be merged into main when they are completed. This allows main to be always be capable of building when pulled from and is best practice when working with a team.

Data Structures

The main data structures for this Angular project are:

- Users
- Groups
- Channels
- Permissions

Users is an array containing 0 or more user objects. Each user has the following attributes:

- `username`: string
- `email`: string
- `id`: number
- `role`: string

A User can have the roles: `default`, `super`, `gAdmin`, `gAssist`. These role respectively give the permissions of a default, super admin, group admin and group assistant.

Groups is an array containing 0 or more group objects. Each group has the following attributes:

- `id`: number
- `groupName`: string

Channels is an array containing 0 or more group objects. Each channel has the following attributes:

- `id`: number
- `channelName`: string

Users, Groups and Channels are stored in /server/data as serialised JSON files

Permissions is an array of permission objects. Each permission object contains a `groupId` (a reference to the `id` of a group object) and an array of members that will have the permission level. Each member will have a `userId` (a reference to an id of a User) and an array of `channelIds` (a reference to an id of a Channel), which are the channels the user is subscribed to.

```
{groupId: number, members: [
    {userId: number, channelIds: [number]
]}
```

REST API

For the following routes, the parameters and return values are as stated in previously in the "data structures" section, unless explicitly stated

Auth route

Route overview	**This route checks if a user exists in a predefined JSON. If the user exists, it will call a login function to log the user in; returning an error if they do not.**
Route path	/auth
Method	POST
Parameter/s	`username`
Return value/s	if (auth): `{valid: true, user: {username, email, id, role}}` if !(auth): `{valid: false, errors:{}}`
Description	This route takes a `username` in the http request. If there is a match in the users.json file then the route returns `valid: true` and the user information. If there is no match, valid: false is returned.

Set permissions route

Route overview	**This route gives app permissions to a user depending on the value of user.role**
Route path	/setPermissions
Method	POST
Parameter/s	`user: {username, email, id, role}`

Route overview	**This route gives app permissions to a user depending on the value of user.role**
Return value/s	***See below code block for proper formatting
Description	This route takes a `user` and returns the `permissions` object where default users have no permissions. Relevant permissions are returned according to the user `roles`.

```
{
    rolePermissions: {
        // super admin
        canCreateUser: boolean,
        canRemoveUser: boolean,
        canUpgradeUserSuper: boolean,
        // group admin
        canCreateGroup: boolean,
        canCreateGroupChannel: boolean,
        canCreateChannelUser: boolean,
        canInviteCreateChannelUser: boolean,
        canUpdateChannel: boolean,
        canUpgradeUserGAssist: boolean,
        // group assist
        canAddRemoveChannelUser: boolean,
        canCreateGroupChannel: boolean
    }
}
```

Create User

Route overview	**Creates a new user if the username doesn't exist**

Route overview	**Creates a new user if the username doesn't exist**
Route path	/user/create
Method	POST
Parameter/s	`{user: {username, email, id: undefined, role}}`
Return value/s	No user match: `users` user match: `error`
Description	The route reads `users.json`, and if there is no `user` match then it will add the `user` to `users` and assign a `userId`. It writes `users` to `users.json` and returns `users`

Update User

Route overview	**Updates a User**
Route path	/user/:id
Method	POST
Parameter/s	Request parameters: `id` Request body: `{user: {username, email, id, role}}`
Return value/s	`users`
Description	A `userID` and a `user` object is received and `users.json` is read. It matches the `userID` and `id` of the `user` object and replaces the values with `user`. `users` is written to users.json and the route responds with this return value.

Delete User

Route overview	**Deletes a user**
Route path	/user/delete
Method	POST
Parameter/s	Request parameters: `userId`
Return value/s	`users`
Description	A `userID` is received and `users.json` and `userInfo.json` is read. It matches the `userID` and `id` and splices the array element with that `userId` from `userInfo.json`. It then matches the `username` from `userInfo.json` and `users.json` and splices the array element where the username matches.

Get Users

Route overview	**Gets all users**
Route path	/users
Method	POST
Parameter/s	N/A
Return value/s	`users`
Description	onInit() of a page that requires a list of users, this function is called, which reads `userInfo.json` and returns the data as a JSON object.

Get User Channels, Groups and permissions

Route overview	**Gets Channels, Groups and permissions that belongs to the user**
Route path	/users/channels
Method	POST
Parameter/s	`{user: {username, email, id, role}}`
Return value/s	`userAccess`
Description	The route receives a user object and reads channels, groups and permissions from the .json/s and initialises `userAccess`. The route matches channels, groups and permissions with a user that has a `userid` matching the input parameter and pushes this data to `userAccess` and responds with this data.

Create Groups

Route overview	**Creates a new group**
Route path	/group/create
Method	POST
Parameter/s	`name: string`
Return value/s	`groupID: number`
Description	The route reads `groups.json`, and if there is no `group` match then it will add the `group`'s name to `groups` and assigns a `groupId`. It pushes `group` to the array of groups and writes it back to file. It returns `groupId`.

Delete Group

Route overview	**Deletes a group**
Route path	/group/delete
Method	POST
Parameter/s	`groupId: number`
Return value/s	N/A
Description	Uses `groupId` and reads objects in the groups and permissions json files with matching ids. It splices the group from the these read arrays with regards to `groupId`. It then writes these arrays back to file

Add User to Group

Route overview	**Adds user to group**
Route path	/group/addUser
Method	POST
Parameter/s	`userId: number, groupID: number`
Return value/s	N/A
Description	Receives a `userId` and `groupId` and finds the index of the permissions json that matches `groupId`. If the userId does not exist, then it pushes the parameters to `permissions` array and writes it to file.

Remove User from Group

Route overview	**Removes User from group**
Route path	/group/removeUser
Method	POST
Parameter/s	`userId: number, groupID: number`
Return value/s	N/A
Description	Receives a `userId` and `groupId` and finds the index of the permissions json that matches `groupId`. If the userId exists, then it splices the parameters from the `permissions` array and writes it to file.

Create Channel

Route overview	**Creates a new channel**
Route path	/channel/create
Method	POST
Parameter/s	`{ name: string, groupId }`
Return value/s	`groupID: number`
Description	The route reads `groups.json` and `channels.json`, and if there is a matching `group` and no matching channel name then it will create the `channel` with an id of `channels.length`. It pushes input object to the array of `channels` and writes it back to file. It also read/writes the groups and channels with the new channel info.

Delete Channel

Route overview	**Deletes a channel**
Route path	/channel/delete
Method	POST
Parameter/s	`channelId: number`
Return value/s	N/A
Description	Uses `channelId` and reads objects in the groups, channels and permissions json files with matching `groupIds`. It splices the channel from the these read arrays with regards to `channelId`. It then writes these arrays back to file

Add User to Channel

Route overview	**Adds user to channel**
Route path	/channel/addUser
Method	POST
Parameter/s	`userId: number, groupID: number, channelID: number`
Return value/s	N/A
Description	Receives a `userId`, `groupId` and `channelID` and finds the index of the permissions json that matches `groupId`. If the userId does not exist, then it pushes an object containing `userId channelId[]` to `permissions` array and writes it to file.

Remove User from Channel

Route overview	**Removes user from channel**
Route path	/channel/removeUser
Method	POST
Parameter/s	`userId: number, groupID: number, channelID: number`
Return value/s	N/A
Description	Receives a `userId`, `groupId` and `channelID` and finds the index of the permissions json that matches `userId` and `channelId` and splices that object from `channelIds` and writes it to file.

Angular Architecture

Components

Login

A login form greets the user upon visiting the site prompting them to login. It is a form requiring a username and password inputs, which sends a HTTP POST request to /api/auth. If the login is valid (`valid = true`), then it sets the `username` and `password` into `localStorage`. It also simutaneously calls the permissions service's `updateUser()` that will update components that is subscribed to `loggedInUser`.

Chat - Core

The core functionality of the program. It consists of a navigation bar and a `routerLink` to each component.

Chat - Site permissions

Action	**Overview**	**Permission**	**Has permission**
Create user	Can create users with Group Admin role.	canCreateUser	Super Admin
Remove User	Can remove all users	canRemoveUser	Super Admin
Upgrade permission to super	Can upgrade another user to Super admin and lower permissions	canUpgradeUserSuper	Super Admin
Create group	Can create groups	canCreateGroup	Super Admin, Group Admin

Action	Overview	Permission	Has permission
Create group channel	Can create channels (subgroups) within groups	canCreateGroupChannel	Super Admin, Group Admin
Invite + create channel user	Can create and or invite users to a channel	canInviteCreateChannelUser	Super Admin, Group Admin
Update channel	Can remove groups, channels, and users from channels	canUpdateChannel	Super Admin, Group Admin
Upgrade permission to Assist	Can allow a user to become a Group Assis of the group	canUpgradeUserGAssist	Super Admin, Group Admin
Add/remove users from group channels	Can add or remove users in the group from channels within the group	canAddRemoveGroupChannels	Super Admin, Group Admin, Group Assist
Create channel within group	Can create channels within the group	canCreateGroupChannel	Super Admin, Group Admin, Group Assist

Chat - Action Menu WIP

Chat - Get Chat History WIP

Chat - Messages WIP

Groups WIP

Channels WIP

User Administration (UA)

UA - CRUD menu

The menu allowing users to perform create, read, update and delete on the main data structures. This menu and certain actions will only be available (and visible) to users who have the appropriate permissions (as show in the permission table above). These permissions are assigned `onInit()` to users by using `setUserRolePermissions` route to return the appropriate permissions based on the current user's role.

Create (register) user

Pressing the "register" button will bring a form that will allow a user to input `username` and `email`, which will be posted to the server to create a user.

Edit user WIP

Delete user

For super admins, pressing the "delete" button will delete the user from the database. The button and route will not be available for other users.

Guards

Auth (Login)

This guard is used to stop unauthorised users from accessing routes in which they do not have permissions for. It will give users an alert to say why they cannot access this route and redirect them to login if they need to authenticate themselves.

Services

Auth (Login)

`loginEvent()` Takes a `uname` and `pword` and posts it to the server to handle logins `logoutEvent()` Logs a user out of their account and removes their data from `localStorage`

User Administration (CRUD)

`registerUser()` Takes a `User` object and posts the data to the server for the creation of a new user. `getAllUsers()` Returns all registered users on the website. `deleteUser()` Takes a `userId` and posts it to the server to handle deletion of a user.