

Assignment Phase 2

Software Frameworks ChatterApp

Ethel Beckett | s5125717
October 8, 2020

Documentation

I. Git Repository

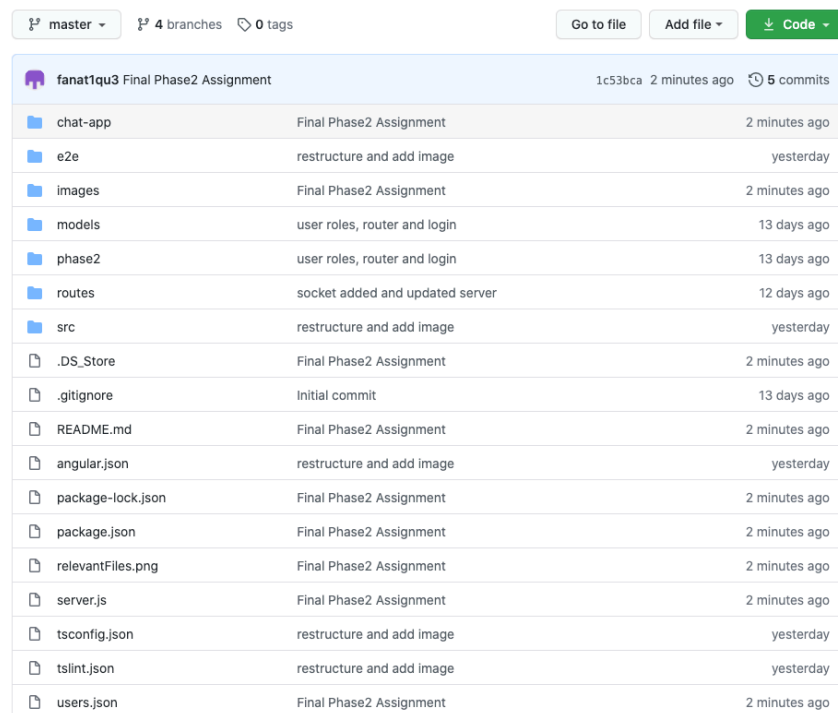
Git repository helped me with my back-ups that allow me to pull my commits when necessary. My repository is set to private to prevent unauthorised access. During the development I generated couple of branches as per the list below:

- Master
Is my master file, it is the foundation I have built from phase1.
- Restructure
I ended up restructuring my architecture to make it simpler and easier for me when I have to get the path of my routes.
- Socket update
Is where I updated and fix issues regarding the chat components
- Images
Is the branch where I added the image feature and also restructure the architecture again.

Below is the screenshot and link of my GitHub Repository

<https://github.com/ejbeckett/phase2.git>

Figure 1 Master



master 4 branches 0 tags		Go to file	Add file	Code
fanat1qu3 Final Phase2 Assignment		1c53bca	2 minutes ago	5 commits
chat-app	Final Phase2 Assignment			2 minutes ago
e2e	restructure and add image			yesterday
images	Final Phase2 Assignment			2 minutes ago
models	user roles, router and login			13 days ago
phase2	user roles, router and login			13 days ago
routes	socket added and updated server			12 days ago
src	restructure and add image			yesterday
.DS_Store	Final Phase2 Assignment			2 minutes ago
.gitignore	Initial commit			13 days ago
README.md	Final Phase2 Assignment			2 minutes ago
angular.json	restructure and add image			yesterday
package-lock.json	Final Phase2 Assignment			2 minutes ago
package.json	Final Phase2 Assignment			2 minutes ago
relevantFiles.png	Final Phase2 Assignment			2 minutes ago
server.js	Final Phase2 Assignment			2 minutes ago
tsconfig.json	restructure and add image			yesterday
tslint.json	restructure and add image			yesterday
users.json	Final Phase2 Assignment			2 minutes ago

Figure 2 Branches

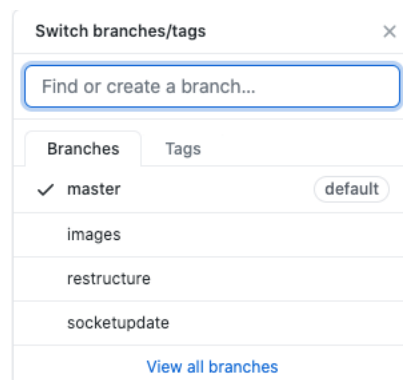
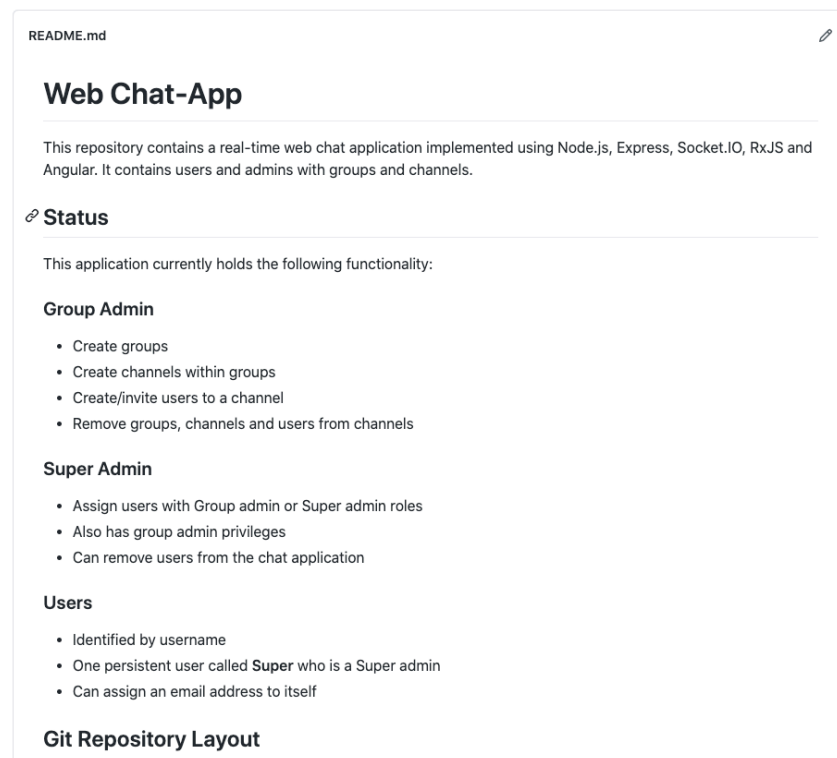


Figure 3 README.md



II. Data Structures

Client-side

The client side is composed of four (4) components such as channel, dashboard, group, and login components. The channel is where all the user can chat and create new channels for a group. The dashboard serves as the user profile where they can see their access level, the group components are where all the user roles are set, creating groups and channels and the login is where user can login using their username and password.

There are users that has been implemented such as Super Admin, Group Admin, and GroupAssist.

Group Admin

- Login functionality
- Create new user
- Create groups
- Create channels within groups
- Create/invite users to a channel
- Remove groups, channels and users from channels

Super Admin

- Assign users with Group admin, Super admin roles and group assist
- Has super admin privileges that can add, delete users and assign roles
- Can delete groups or add groups
- Can remove users from the chat application

Group-Assis

- User can remove user from a channel
- User can create new channel within a group
- Join a channel
- Add user to the channel

The client-side also have different services that organizes and share the business logic, models, or data functions with different components. The following are the services generated:

- Image service
- Socket service
- User service

The app module.ts is where all the modules and components used and imported.

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';
import { HttpClientModule } from '@angular/common/http';
```

```
import { AppRoutingModuleModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { LoginComponent } from './login/login.component';
import { DashboardComponent } from
'./dashboard/dashboard.component';
import { GroupComponent } from './group/group.component';
import { ChannelComponent } from
'./channel/channel.component';
import { SocketService } from './socket.service';
```

The app-routing.module.ts is also used to set all the routes for my components.

```
const routes: Routes = [
  {
    path: '', component:LoginComponent
  },
  {
    path: 'dashboard', component:DashboardComponent
  },
  {
    path: 'group', component:GroupComponent
  },
  {
    path: 'channel', component:ChannelComponent
  }
];
```

Server-side

The server side consist of one (1) file which is the server.js. The express, sockets and access to MongoDB is now on one file. Having different folders made me confused to access my api so I decided to use one file for all the server-side requirements and it is located at the root of my project.

III. Responsibilities between client and server

The **client-server** responsibility is to allow user to interact with the application. It also allows super-admin to assign roles to the user when a user is created. The user cannot access the application or any components of the application until they have their own log in detail. On API endpoint, it returns a list of objects that has been saved in the mongoDb via post and get method. The “service” to call the method via the server then this service will be injected to the app.module.ts as part of its dependency which enables me to create a RESTful api which makes the configuration of the controller simpler.

The backend server is utilised to communicate with the HTTP module. It allows me to obtain the data and display it to the front-end. The backend serves our data which is the server.js. A node express is used to allow Javascript to write server-application. The express frameworks were used to for the REST-API to serve the data in the application that has been requested and then send it to the front-end.

The files for the backend server has been compiled into one file which comprise of node express, mongo database, routes, and socket.

IV. Routes, parameters, return values, and purpose

There are couple of routes that was used for the application such as the following:

- App.routing.module.ts
Allows the path to be added to the routing module to allow access or navigate between components.
- RouterLink
Use to provide the path to the component
- Express routing
Serves as the endpoints that responds to the client's request that corresponds with the HTTP methods.
- Router
Enables navigation between components
- Routes and RouterModule
Adds directives and providers for in-app navigation between views defined in the application.

Parameters used in the application are the following:

- LoginComponent
Where user input their login details to be able to use the application. First point of interaction of the user.
- DashboardComponent
Serves as the profile of the user. It is where they can see the level of their access and apply some settings.
- GroupComponent
The group component is the page for all groups that were created. Depending on the user access they may able to add user, invite or add user, delete user, or create new channels.
- ChannelComponent
Is the page where users can chat real-time if they are part of the group and channel.

Return values used in the application are the following:

- Username = string
Users can choose whatever username they want, and this has to be unique.
- Password = string
Used to authenticate the user
- Email = string

Users email address that is associated with their username

- superAdmin = boolean
Set to Boolean if true or false
- groupAdmin = boolean
Set to Boolean if true or false
- groupAssis = boolean
Set to Boolean if true or false
- profileImage = string
used to store all images used
- messages
used to store all the messages

V. Angular Architecture

There are four (4) Components setup

- Channel component
- Dashboard component
- Group Component
- Login Component

Services

- Socket
- Users
- Image

Images (root)

- Where images are stored

Testing e2e

- Login.e2e-spec.ts
- Login.po.ts
- Public.e2e-spec.ts
- Public.po.ts

Server (server.js)

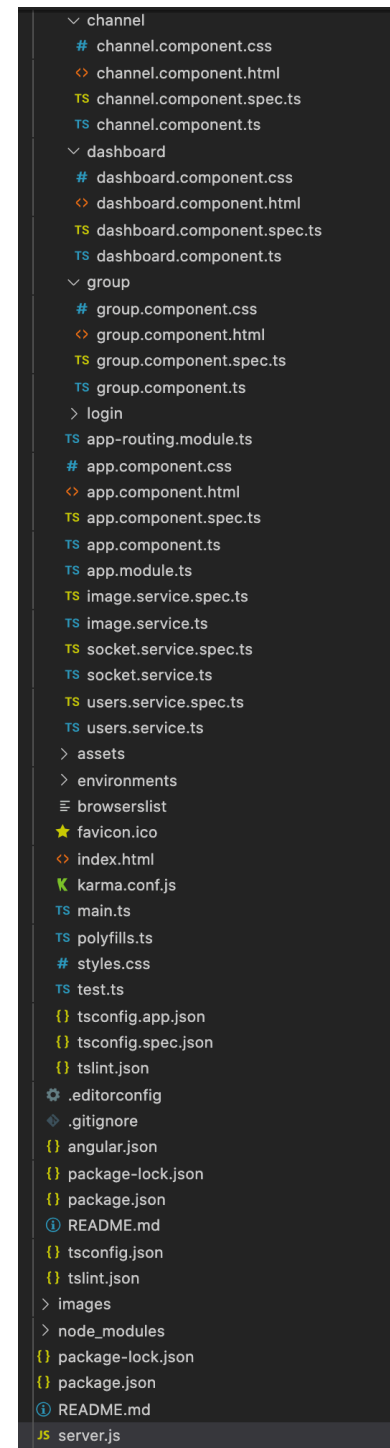
- Express
- Socket
- Body-parser
- Cors
- Socket
- MongoDB

VI. Interaction between client and server

The client and server work together to achieve a goal. They are created so that a user can access or interact with the application. Initially the user's points the browser to the URL of the application. This creates multiple request to the application server then the browser will wait for the response from the server. Each response that is received by the client is rendered as part of the application.

Once response is received the user will see the pages of the application that they can interact with which will trigger additional request from the user and obtain more data from the server.

The HTTP request is received from the browser to the backend. Those requests may contain in the HTTP or its request body which is intends to transmit data to the backend and back to the client with the response along with the data. The request is triggered by the client and whichever button or link the user clicked it will send a request a new request to the backend. In this scenario, I also use a middleware which can be called



every time it is triggered. It uses service, for example authentication for users. If the user is not logged in, it will prevent the user to access any functionalities and will be directed for user to login again.

Once the middleware is ready, it can be injected to the components where it is required. When the middleware has been triggered. The backend does not have idea how the data is going to look like the front-end takes care of how the data will be rendered to the client. The backends' goal is to only respond to the request with JSON responses.