Deliverable 3 – Project Phase 1 CSCE 5430 (Fall 2021)

CODE INSPECTION

(Source Code for Core Functionalities and their description)

Project Title - Sociogram (Web Application)

Team Name – Code6

Group Members –

1. Sathwik Gaddi
2. Tanvi Thirunathan
3. Satish Thammaneni
4. Sai Karthik koncherlakota
5. Raja Srinivas Nelluri
6. Sri Snigdha kotharu
7. Venkata Rajashekar Reddy Chintala
8. Putta Ramya Priya

Server.js and app.js:

It is the initial file for the NodeJS application. It contains the code to start and serve the application, connection to database, allowing access for cross origin resource sharing etc.

Server.Js:

```
const app = require("./app");
const debug = require("debug")("node-angular");
const http = require("http");

const normalizePort = val => {
  var port = parseInt(val, 10);

  if (isNaN(port)) {
    // named pipe
    return val;
  }

  if (port >= 0) {
    // port number
    return port;
  }

  return false;
};

const onError = error => {
  if (error.syscall !== "listen") {
    throw error;
  }
  const bind = typeof port === "string" ? "pipe " + port : "port " + port;
  switch (error.code) {
    case "EACCES":
      console.error(bind + " requires elevated privileges");
      process.exit(1);
      break;
    case "EADDRINUSE":
      console.error(bind + " is already in use");
      process.exit(1);
      break;
    default:
      throw error;
  }
};

const onListening = () => {
  const addr = server.address();
  const bind = typeof port === "string" ? "pipe " + port : "port " + port;
  debug("Listening on " + bind);
};

const port = normalizePort(process.env.PORT || "3000");
app.set("port", port);

const server = http.createServer(app);
server.on("error", onError);
server.on("listening", onListening);
server.listen(port);
```

App.js:

```
const path = require("path");
const express = require("express");
const bodyParser = require("body-parser");
const mongoose = require("mongoose");

const postsRoutes = require("./routes/posts");
const userRoutes = require("./routes/user");

const app = express();

mongoose
  .connect(
    "mongodb://localhost:27017/sociogramDB"
  )
  .then(() => {
    console.log("Connected to database!");
  })
  .catch((err) => {
    console.log(err);
  });

app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: false }));
app.use("/images", express.static(path.join("backend/images")));

app.use((req, res, next) => {
  res.setHeader("Access-Control-Allow-Origin", "*");
  res.setHeader(
    "Access-Control-Allow-Headers",
    "Origin, X-Requested-With, Content-Type, Accept, Authorization"
  );
  res.setHeader(
    "Access-Control-Allow-Methods",
    "GET, POST, PATCH, PUT, DELETE, OPTIONS"
  );
  next();
});

app.use("/api/posts", postsRoutes);
app.use("/api/user", userRoutes);

module.exports = app;
```

Phase 1 Core features -

1. Signup

Once the user enters the details for the registration, the entered data is validated and passed to backend REST API. Then the password is encrypted using Bcrypt hash method. All the details of the user are stored in the data base system.

Rest API for Sign Up – http://localhost:3000/api/user/signup

Controller for Sign up:

```
exports.createUser = (req, res, next) => {
  console.log();
  bcrypt.hash(req.body.password, 10).then(hash => {
    console.log(hash)
    const user = new User({
      email: req.body.email,
      password: hash,
      username: req.body.username,
      friends: [],
      interests: req.body.interests
    });
    user
      .save()
      .then(result => {
        res.status(201).json({
          message: "User created!",
          result: result
        });
      })
      .catch(err => {
        res.status(500).json({
          message: "Invalid authentication credentials!"
        });
      });
  });
}
```

Mongoose Schemas for Users -

It is a predefined schema written to ensure that the User data is always in this shape and stored likewise in the database.

```
const mongoose = require("mongoose");
const uniqueValidator = require("mongoose-unique-validator");

const userSchema = mongoose.Schema({
  email: { type: String, required: true, unique: true },
  password: { type: String, required: true },
  username: {type: String, required: true},
  friends: { type: Array },
  interests: { type: Array, required: true}
});

userSchema.plugin(uniqueValidator);

module.exports = mongoose.model("User", userSchema);
```

2. Login

when user enters the details for Login, after successful validation of data it is passed to backend REST API. Then the username is retrieved and the password is compared using Bcrypt hash compare method. After successful validation of data, a json web token is created with the details, a secret key encrypted and safely sends it to frontend in response.

Rest API for Login – http://localhost:3000/api/user/login

Controller for Login –

```
exports.userLogin = (req, res, next) => {
  let fetchedUser;
  console.log(req.body.email, req.body.password);
  User.findOne({ email: req.body.email })
    .then(user => {
      if (!user) {
        return res.status(401).json({
          message: "Auth failed"
        });
      }
      fetchedUser = user;
      return bcrypt.compare(req.body.password, user.password);
    })
    .then(result => {
      if (!result) {
        return res.status(401).json({
          message: "Auth failed"
        });
      }
      console.log("aaaaaaaaaaa"+fetchedUser.username)
      const token = jwt.sign(
        { email: fetchedUser.email, userId: fetchedUser._id, username:
fetchedUser.username },
        "secret_this_should_be_longer",
        { expiresIn: "1h" }
      );
      res.status(200).json({
        token: token,
        expiresIn: 3600,
        userId: fetchedUser._id,
        username: fetchedUser.username
      });
    })
    .catch(err => {
     res.status(401).json({
        message: "Invalid authentication credentials!"
      });
    });

    }
```

3. Upload media -

Users can find an option in the Navbar to add a new post. When the user enters the post details in the provided form, they will be validated and sent to the Backend Api. They are then stored in the database.

Add post Rest Api: http://localhost:3000/api/posts (POST)

When the user selects view posts option from the Navbar, the backend server checks for the user authentication and the posts are retrieved from the database and shown in a list view. There will be options to edit and delete the post.

Retrieve Posts Rest API: https://localhost:3000/api/posts (GET)

Edit Post Rest Api: http://localhost:3000/api/posts/:id (PUT)

Delete Post Rest Api: http://localhost:3000/api/posts/:id(DELETE)

Create Post Controller -

```
exports.createPost = (req, res, next) => {
  const url = req.protocol + "://" + req.get("host");
  const post = new Post({
    title: req.body.title,
    content: req.body.content,
    imagePath: url + "../src/assets/" + req.file.filename,
    creator: req.userData.userId,
    privacy: req.body.privacy
  });
  post
    .save()
    .then(createdPost => {
      res.status(201).json({
        message: "Post added successfully",
        post: {
          ...createdPost,
          id: createdPost._id
        }
      });
    })
    .catch(error => {
      res.status(500).json({
        message: "Creating a post failed!"
      });
    });
};
```

Get Posts Controller –

```
exports.getPosts = (req, res, next) => {
  const pageSize = +req.query.pagesize;
  const currentPage = +req.query.page;
  const postQuery = Post.find();
  let fetchedPosts;
  if (pageSize && currentPage) {
    postQuery.skip(pageSize * (currentPage - 1)).limit(pageSize);
  }
  postQuery
    .then(documents => {
      fetchedPosts = documents;
      return Post.count();
    })
    .then(count => {
      res.status(200).json({
        message: "Posts fetched successfully!",
        posts: fetchedPosts,
        maxPosts: count
      });
    })
    .catch(error => {
      res.status(500).json({
        message: "Fetching posts failed!"
      });
    });
};
```

4. Find Similar Friends -

When user selects the View Friends button at the Navbar, people with similar interests will be listed. A user can add a friend and unfriend them based on their preference.

Rest Api for View similar people: http://localhost:3000/api/user/ (GET)

Rest Api for Add Friend: http://localhost:3000/api/user/addfriend/username (POST)

Rest API for Unfriend:

http://localhost:3000/api/user/unfriend/username (PUT)

Controller for getSimilarUsers –

```
exports.getSimilarUsers = (req, res, next) => {
      let similarUsers = []
      User.findOne({"_id" : req.userData.userId})
      .then(user => {
        User.find({$or: [{"interests":user.interests[0]},
{"interests":user.interests[1]}]})
        .then(filteredUsers => {
          similarUsers = filteredUsers;
          return User.count()
        })
        .then(count => {
          res.status(200).json({
            message: "Similar friends fetched successfully!",
            similarfriends: similarUsers,
            maxSimilarFriends: count
          });
        })
        .catch(error => {
          res.status(500).json({
            message: "Fetching similar friends failed!"
          });
        });
      })

    }

    exports.getMyFriends = (req, res, next) => {

      let myFriends = []
      User.findOne({"_id" : req.userData.userId})
      .then(user => {
        myFriends = user.friends
        res.status(200).json({
          message: "Friends fetched successfully!",
            friends: myFriends
        })
        })
        .catch(error => {
          res.status(500).json({
            message: "Fetching  friends failed!"
          });
        });

    }
```

Controller for AddFriend -

```
 exports.addFriend = (req, res, next) => {

      let id = req.userData.userId;

      User.findById({ _id: id})
      .then(user => {
        user.friends.push(req.body.username)
        User.updateOne({ _id: id}, user)
        .then(result => {
          if (result.n > 0) {
            res.status(200).json({ message: "Update successful!" });
          } else {
            res.status(401).json({ message: "Not authorized!" });
          }
        })
        .catch(error => {
          res.status(500).json({
            message: "Couldn't add Friend!"
          });
      })
    })
}
```

Controller for Unfriend -

```
exports.deleteFriend = (req, res, next) => {
  console.log("in delete friend")
  let id = req.userData.userId;
  User.findById({ _id: id})
  .then(user => {
    for( var i = 0; i < user.friends.length; i++){
      if ( user.friends[i] === req.body.username) {

          user.friends.splice(i, 1);
      }

  }
    // user.friends.push(req.body.id)
    User.updateOne({ _id: id}, user)
    .then(result => {
      if (result.n > 0) {
        res.status(200).json({ message: "Update successful!" });
      } else {
        res.status(401).json({ message: "Not authorized!" });
      }
    })
    .catch(error => {
      res.status(500).json({
        message: "Couldn't delete Friend!"
      });
  })
})

}
```
`

User Authentication:

For every REST api request sent from the frontend source, the json web token will be mentioned in the request header for the user authentication. This will be authenticated at the backend to get the details of the current user.

checkAuth.js -

```
const jwt = require("jsonwebtoken");

module.exports = (req, res, next) => {
  try {
    const token = req.headers.authorization.split(" ")[1];
    const decodedToken = jwt.verify(token, "secret_this_should_be_longer");
    req.userData = { email: decodedToken.email, userId: decodedToken.userId };
    next();
  } catch (error) {
    res.status(401).json({ message: "You are not authenticated!" });
  }
};
```

Angular routing -

All the front screens are navigated by clicks using the app-routing.module.ts file. All the angular routed are defined in this file.

App-routing.module.ts -

```
import { NgModule } from "@angular/core";
import { RouterModule, Routes } from "@angular/router";
import { PostListComponent } from "./posts/post-list/post-list.component";
import { PostCreateComponent } from "./posts/post-create/post-
create.component";
import { AuthGuard } from "./auth/auth.guard";
import { FindFriendsComponent } from "./friends/find-friends/find-
friends.component";
import { SignupComponent } from "./auth/signup/signup.component";
import { ViewDiscussionComponent } from "./discussion-forum/view-
discussion/view-discussion.component";
import { HomePageComponent } from "./home-page/home-page.component";
import { FriendsListComponent } from "./friends/friends-list/friends-
list.component";

const routes: Routes = [
  { path: "", component: HomePageComponent},
  { path: "create", component: PostCreateComponent, canActivate: [AuthGuard]
},
  { path: "edit/:postId", component: PostCreateComponent, canActivate:
[AuthGuard] },
  {path: "viewPosts", component:PostListComponent, canActivate: [AuthGuard]},
  { path: "auth", loadChildren: "./auth/auth.module#AuthModule"},
  {path: "find-friend", component: FindFriendsComponent},
  {path: "viewDiscussions", component: ViewDiscussionComponent},
  {path: "addfriend/:fId", component: FindFriendsComponent},
  {path: "viewFriends", component: FriendsListComponent}
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule],
  providers: [AuthGuard]
})
export class AppRoutingModule {}
```

app.module.ts -

All the angular components and modules involved are defined in this file.

App.module.ts -

```
import { BrowserModule } from "@angular/platform-browser";
import { BrowserAnimationsModule } from "@angular/platform-
browser/animations";
import { NgModule } from "@angular/core";
import { HttpClientModule, HTTP_INTERCEPTORS } from "@angular/common/http";

import { AppComponent } from "./app.component";
import { HeaderComponent } from "./header/header.component";
import { AppRoutingModule } from "./app-routing.module";
import { AuthInterceptor } from "./auth/auth-interceptor";
import { ErrorInterceptor } from "./error-interceptor";
import { ErrorComponent } from "./error/error.component";
import { AngularMaterialModule } from "./angular-material.module";
import { PostsModule } from "./posts/posts.module";
import { FriendsModule } from "./friends/friends.module";
import { AuthModule } from "./auth/auth.module";
import { DiscussionForumModule } from "./discussion-forum/discussion-
forum.module";
import { HomePageComponent } from './home-page/home-page.component';

@NgModule({
  declarations: [
    AppComponent,
    HeaderComponent,
    ErrorComponent,
    HomePageComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    BrowserAnimationsModule,
    HttpClientModule,
    AngularMaterialModule,
    PostsModule,
    FriendsModule,
    AuthModule,
    DiscussionForumModule
  ],
  providers: [
    { provide: HTTP_INTERCEPTORS, useClass: AuthInterceptor, multi: true },
    { provide: HTTP_INTERCEPTORS, useClass: ErrorInterceptor, multi: true }
  ],
  bootstrap: [AppComponent],
  entryComponents: [ErrorComponent]
})
export class AppModule {}
```