Web Applications I – Exam # 3 (deadline 2021-09-07 at 23:59)

"Study Groups"

FINAL VERSION – Modifications are reported in "red"

Design and implement a web application to manage some *study groups* for University students.

The application must satisfy the following requirements.

A *study group* (SG) is a set of students that decide to study together in preparation to a given university exam. Therefore, the SG is characterized by the information about the course (code, name, credits)[1], by a list of students who participate in the group, and by a schedule of (past and future) meetings of the group. Each course may have at most one ~~active~~ study group, and for clarity it is associated with a different color; this color will be used whenever the application shows information about that SG.

All the users of the system have a role of 'student', and each user may have extra roles ~~(each role has also access to all the features of the lower roles)~~. The defined roles are:

- General Administrator. Note: there may be more than one general administrator.
- Group Administrator (for one or more study groups). Note: each SG may have more than one group administrator
- Student (all users have this role, therefore all users may have access to its functions).

A user with role of General Administrator may:

- See the list of all study groups
- See the information of a selected Study Group: list of members, list of group administrators, list of past and future meetings
- Create a new Study Group, by entering the data about the course
- Remove a Study Group
- Add or remove group administrators for any Study Group, by selecting them from the list of current members of the SG

A user with role of Group Administrator may:

- See the list of Study Groups he/she may administer, and their members (this is the same functionality of the first 2 items of the General Administrator, but restricted to the SGs administered by the student)
- Define future meetings (date, time, duration, location) for the study group
- See the list of students for the study group
- Approve student requests for joining the study group
- Remove a student from a student group

A user with the role of Student may:

- See the list of all study groups

[1] No prior information is needed about the courses

- Ask to join a study group (the request must be approved by a group administrator)
- See the list of future meetings for all joined study groups; all meetings must the shown in the same list/table/calendar, using the color coding to identify each SG
- Sign-up for one future meeting for a joined study group or cancel a signed-up meeting. In case the user tries to sign up for a meeting that overlaps with an already signed-up meeting, the system will ask for further confirmation before proceeding.

Project requirements

- The application architecture and source code must be developed by adopting the best practices in software development, in particular those relevant to single-page applications (SPA) using React and HTTP APIs.
- The project must be implemented as a React application, that interacts with an HTTP API implemented in Node+Express. The database must be stored in a SQLite file.
- The communication between client and server must follow the "React Development Proxy" pattern and React must run in "development" mode.
- The root directory of the project must contain a `README.md` file and have two subdirectories (`client` and `server`). The project must be started by running the two commands: "`cd server; nodemon server.js`" and "`cd client; npm start`". A template for the project directories is already available in the exam repository. You may assume that `nodemon` is already installed globally.
- The whole project must be submitted on GitHub, on the same repository created by GitHub Classroom.
- The project **must not include** the `node_modules` directories. They will be re-created by running the "`npm install`" command, right after "`git clone`".
- The project may use popular and commonly adopted libraries (for example `day.js`, `react-bootstrap`, etc.), if applicable and useful. Such libraries must be correctly declared in the `package.json` and `package-lock.json` files, so that the `npm install` command might install them.
- User authentication (login) and API access must be implemented with `passport.js` and session cookies. No further protection mechanism is required. The user registration procedure is not requested.
- The project database must be included in the submission, and must be pre-loaded with *at least 5 users* and *2 study groups*, with the following mapping (the actual name of the users and groups is free to choose).
    - User 1: General Administrator, Student
    - User 2: General Administrator, Group Administrator for Study Group A, Student
    - User 3: Group Administrator for Study Group A, Group Administrator for Study Group B, Student
    - User 4: Group Administrator for Study Group B, Student
    - User 5: Student.

Contents of the README.md file

The README.md file must contain the following information (a template is available in the project repository). Generally, each information should take no more than 1-2 lines.

1. A list of 'routes' for the React application, with a short description of the purpose of each route
2. A list of the HTTP APIs offered by the server, with a short description of the parameters and o the exchanged objects
3. A list of the database tables, with their purpose
4. A list of the main React components
5. A screenshot of **the page for the list of all future meetings**. This screenshot must be embedded in the README by linking an image committed in the repository.
6. Username and password of the test users (see above).

Submission procedure (important!)

To correctly submit the project, you must:

- **Be enrolled** in the exam call.
- **Accept the invitation** on GitHub Classroom, and correctly **associate** your GitHub username with your student ID.
- **Push the project** in the **`main` branch** of the repository created for you by GitHub Classroom. The last commit (the one you wish to be evaluated) must be **tagged** with the tag `final`.

Note: to tag a commit, you may use (from the terminal) the following commands:

```
# ensure the latest version is committed
git commit -m "...comment..."
git push

# add the 'final' tag and push it
git tag final
git push origin --tags
```

Alternatively, you may insert the tag from GitHub's web interface (follow the link 'Create a new release').

To test your submission, these are the exact commands that the teachers will use to download the project. You may wish to test them on a clean directory:

```
git clone ...yourCloneURL...
cd ...yourProjectDir...
git pull origin main  # just in case the default branch is not main
git checkout -b evaluation final # check out the version tagged with
'final' and create a new branch 'evaluation'
(cd client ; npm install)
(cd server ; npm install)
```

Ensure that all the needed packages are downloaded by the `npm install` commands. Be careful: if some packages are installed globally, on your PC, they might not be listed as dependencies. Always check it in a clean installation.

The project will be tested under Linux: be aware that Linux is case-sensitive for file names, while Windows and macOS are not. Double-check the case of `import` and `require()` statements.