Web Applications – Exam #1 (deadline 2024-06-30 at 23:59)

“Ticketing system”

Preliminary version - the final version will be published on Jun 17, 2024 (approximately).

Please leave doubts and questions as COMMENTS in this doc, and DO NOT DELETE or RESOLVE any existing comment.

Design and implement a web application to create a ticketing system with the features described by the following requirements.

A ticket is an item which is composed of a state (either “open” or “closed”), a category among “inquiry”, “maintenance”, “new feature”, “administrative”, “payment”), an owner, a title, a timestamp, when it is initially created (i.e., submitted to the system), a non-empty initial block of text (i.e., the description of the request) and zero or more additional non-empty blocks of text. Each additional block of text has an associated timestamp, and an author (which can also be different from the owner of the ticket).

In order to make the blocks of text more readable for the viewers, all blocks of text must support the use of the bold tag (<b>) and the italics (<i>) tags to mark portions of the text in the blocks. Moreover, the system must allow users to insert newlines in the blocks of text when entering the text itself, and keep such formatting when visualizing the text.

A generic visitor of the website can, regardless of its authentication status, see a list of all tickets, sorted by the most recent one (on the basis of the date/time when it was created). The list must include ONLY the title, the date, the owner, the category and the state (open or closed).

The system allows users to operate on the tickets according to the following specifications.

Authenticated users can submit a ticket by entering all the information defining the ticket, including the initial block of text. No additional blocks of text are allowed during the first submission. Once the ticket is submitted, before sending the information to the server, a read-only confirmation page summarizing all the information to be submitted must be shown. Then, upon confirmation, the ticket information will be sent to the server and associated with the user submitting it, that will be the ticket owner. If necessary, the user can avoid submitting the ticket, going back to the editing phase where all form fields will be pre-filled with the latest information.

Initially, authenticated users see a list of all the tickets in the same form as the generic visitors. Then, they can select one or more tickets from the list and the ticket item will be expanded to see all the information related to that ticket, i.e., the content of the blocks of text (with date, time, author), displayed in chronological order from the oldest to the newest.

The authenticated user can, in addition, at any time decide to add a new block of text to any ticket, including the ones where he/she is the owner. For that new block of text the authenticated user will be automatically considered as the author.

The owner of a ticket can mark the ticket as closed at any time. No new block of texts can be added to closed tickets by any user.

One or more users in the system can act as administrators. An administrator can perform all user actions and in addition they can: 1) mark any ticket as “closed”; 2) mark any closed ticket as “open”; 3) change the category of the ticket.

Additionally, implement a second server that is in charge of computing an estimation of the amount of time it will take to close a ticket. To simplify the implementation, the estimation will be a function of the number of characters, excluding spaces, of the ticket title and category. The sum of characters, multiplied by 10, plus a uniformly distributed random number between 1 and 240, will be the estimated value. Note that only authenticated users can request such an estimate: normal users will receive a value rounded to the nearest integer number of days, while the administrator will receive the actual estimation.

For any authenticated user, such estimation is shown (with day-level precision) in the confirmation page of the ticket submission, while for the administrators it must be shown (with hour-level precision) also in the list of all tickets together with the title, date, owner, and state.

Note that, since the estimation of the ticket completion times are partially computed randomly, each time they will be requested from the server, they can be different.

The organization of these specifications into different screens (and potentially different routes) is left to the student and is subject to evaluation.

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 an SQLite file.
- The communication between client and server must follow the multiple-server pattern, by properly configuring CORS, and React must run in “development” mode with Strict Mode activated.
- The evaluation of the project will be carried out by navigating the application. Neither the behavior of the “refresh” button, nor the manual entering of a URL (except /) will be tested, and their behavior is not specified. Also, the application should never “reload” itself as a consequence of normal user operations.
- The root directory of the project must contain a README.md file and have the same subdirectories as the template project (`client`, `server` and `server2`). The project must start by running these commands: “`cd server; nodemon index.js`”, “`cd server2; nodemon index.js`” and “`cd client; npm run dev`”. A template for the project directories is already available in the exam repository. Do not move or rename such directories. You may assume that `nodemon` is globally installed.
- The whole project must be submitted on GitHub in the repository created by GitHub Classroom specifically for this exam.
- The project **must not include** the `node_modules` directories. They will be re-created by running the “`npm ci`” command, right after “`git clone`”.
- The project **must include** the `package-lock.json` files for each of the folders (client, server, server2).

- 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 ci` command can download and install all of them.
- User authentication (login and logout) and API access must be implemented with `passport.js` and session cookies, using the mechanisms explained during the lectures. The credentials must be stored in hashed and salted form. The user registration procedure is not requested *unless specifically required in the text*.
- Access to servers different from the authentication one (e.g., server2), must be implemented by using properly designed JWT tokens.

Database requirements

- The database schema must be designed and decided by the student, and it is part of the evaluation.

- The database must contain at least five users. Two of them must also be administrators. Three users, including an administrator, must have, for each one, at least one ticket open and one ticket closed. One of the tickets must only have the initial block of text. Two other tickets must have three blocks of texts, with at least one block of text by an author different from the owner. Blocks of text must contain bold characters, italics characters, and newlines. Each category must be used by at least one ticket.

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. Server-side:
    a. A list of the HTTP APIs offered by the server, with a short description of the parameters and of the exchanged objects.
    b. A list of the database tables, with their purpose, and the name of the columns.
2. Client-side:
    a. A list of 'routes' for the React application, with a short description of the purpose of each route.
    b. A list of the main React components developed for the project. Minor ones can be skipped.
3. Overall:
    a. A screenshot of the **list of tickets as seen by the administrator, where at least one ticket is expanded**, and a screenshot of **the submission form**. The screenshot must be embedded in the README by linking the image committed in the repository.
    b. Usernames and passwords of the users, including which one has the administration role.

Submission procedure (IMPORTANT!)

To correctly submit the project, you must:

- **Be enrolled** in the exam call.
- **Accept the invitation** on GitHub Classroom, using the link specific for **this** exam, and correctly **associate** your GitHub username with your student ID.

- **Push the project** in the **branch named "main"** 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: `final` is all-lowercase, with no whitespaces, and it is a git 'tag', NOT a 'commit message').

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
```

NB: **the tag name is "final", all lowercase, no quotes, no whitespaces, no other characters**, and it must be associated with the commit to be evaluated.

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

To test your submission, these are the exact commands that the teachers will use to download and run the project. You may wish to test them in an empty 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 ci; npm run dev)
(cd server ; npm ci; nodemon index.js)
(cd server2 ; npm ci; nodemon index.js)
```

Make sure that all the needed packages are downloaded by the `npm ci` commands. Be careful: if some packages are installed globally, on your computer, they might not be listed as dependencies. Always check it in a clean installation (for instance, in an empty Virtual Machine).

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 upper/lowercase characters, especially of `import` and `require()` statements, and of any filename to be loaded by the application (e.g., the database).