Web Applications – Exam #1 (deadline 2025-06-22 at 23:59)

"Forum"

FINAL version - modifications highlighted with RED text.

Design and implement a web application to manage a forum where users can write posts and others can insert comments related to the posts. The application must implement the following specifications.

A **post** is an item composed of a unique title (among all posts), an author, a non-empty text, an optional maximum number of comments (including zero) that can be appended to the post, and a publication timestamp. Additionally, a list of zero or more comments is associated with a post.

A **comment** is composed of a non-empty text, a publication timestamp, and, optionally, an associated author. Comments without an author are considered "anonymous comments".

Note that the text of both posts and comments can be multiline, i.e., newlines can be inserted when entering the text and such formatting (newlines) must be kept when visualizing such text.

When visualizing the list of posts and/or the list of comments associated with a post, in any view, they must always be listed in chronological order, from the most recent to the less recent one, according to the timestamp, which must be shown in the numeric format year-month-day hour:min:sec. Timestamps refer to the time of the permanent creation in the system (e.g., the info is entered in the database). No handling of time zones is required.

Initially, a generic (authenticated or not) visitor of the website can only see, regardless of its authentication status, a list of all posts showing their title, author, text, timestamp, the number of comments (of any type) currently associated with the post and the maximum allowed one for each post, but not the list of comments appended to each post, except for the anonymous comments.

Anybody (authenticated or not) can append a comment to a post. Comments of non authenticated users will be anonymous comments. Anonymous comments count towards the limit of maximum comments. Note that, once inserted, anonymous comments can be manipulated only when explicitly specified in the following.

An authenticated user can:

1. See the list of posts as the generic visitor of the website, but in addition the user can see the list of appended comments. For each comment, the user can see the text, timestamp, author, and also the total number of "interesting" flags received by all users for that comment (see next point). The user can also see if it has previously marked such a comment as "interesting" for his/her, e.g, showing a flag icon next to the comment;
2. Mark any existing comment, even authored by anybody else, as *interesting*~~/not interesting~~ for his/her. Note that the interesting flag information is private and it has to be shown only to the user that inserted it;
3. Add posts by specifying their title, text and the (optional) maximum number of comments . Other information will automatically be set by the system upon confirmation: the timestamp, and the author (the logged-in user). A newly created post starts with an empty list of

comments in any case. When adding a post, the authenticated user is considered the author by the system;

4. Add comments to any post, regardless of the post author. When adding a comment, the authenticated user is considered the author of the comment. Thus, it cannot add an anonymous comment. Other information such as the timestamp is to be automatically set by the system upon confirmation;
5. Edit the text of comments for which they are the authors. Edit operations do not modify the publication timestamp and have no impact on the interesting mark. Text of posts in not required to be edited after creation time;
6. Delete the comments and posts for which he/she is the author. When deleting posts, all associated comments are removed from the system, regardless of the author, including anonymous ones.

A special category of users, named administrators, can, in addition to the previous operations, also delete anybody's posts or comments, including anonymous ones, and edit the text of any comment, including anonymous ones. Such users, to be able to act as administrators, must authenticate using the 2FA procedure with a TOTP. Note that such a category of users can also choose to authenticate without using the 2FA procedure, thus acting as authenticated users with username/password only.

The organization of these specifications into different screens (and potentially different routes), when not specified, is left to the student and is subject to evaluation.

For the 2FA procedure, for simplicity, use the following secret for all users that require it: `LXBSMDTMSP2I5XFXIYRGFVWSFI` (it is the same used during lectures and labs).

Project requirements

- User authentication (login and logout) and API access must be implemented with `passport.js` and **session cookies, using the mechanisms explained during the lectures**. Session information must be stored on the server, as done during the lectures. The credentials must be stored in hashed and salted form, as done during the lectures. **Failure to follow this pattern will automatically lead to exam failure.**
- 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. **Failure to follow this pattern will automatically lead to exam failure.**
- 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. **Failure to follow this pattern will automatically lead to exam failure.**
- 2FA verification must be implemented by using the libraries explained during the lectures, and with the secret specified in the exam text. **Failure to follow this pattern will automatically lead to exam failure.**
- The evaluation of the project will be carried out by navigating the application, **using the starting URL http://localhost:5173**. 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 user registration procedure is not requested *unless specifically required in the text*.
- 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.

- Particular attention must be paid to ensure server APIs are duly protected from performing unwanted, inconsistent, and unauthorized operations.
- The root directory of the project must contain a README.md file and have the same subdirectories as the template project (`client` and `server`). The project must start by running these commands: "`cd server; 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 and server).
- 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.

Database requirements

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

- The database must be implemented by the student, and must be pre-loaded with at least 5 users, 2 of which are administrators. Four users, including two administrators, should have published two posts each, with a variety of comments. One user and one administrator should have posts which have 1 less comment than the allowed maximum.

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, at least, the **post creation page**. The screenshot must be embedded in the README by linking the image committed in the repository.
   b. Usernames and passwords of the users, including if they are administrators or not.

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'), otherwise the submission will not be evaluated.

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').

Finally, check that everything you expect to be submitted shows up in the GitHub web interface: if it is not there, it has not been submitted correctly.

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)
```

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).