Web Applications I – Exam # 1 (deadline 2023-06-25 at 23:59)

“CMSmall”

⚠ PRELIMINARY VERSION – OPEN TO COMMENTS AND QUESTIONS. PLEASE, DO NOT CLOSE ANY COMMENTS AS THEY CAN BE HELPFUL FOR OTHERS.

THE FINAL VERSION WILL BE PUBLISHED ON JUNE 14. ⚠

Design and implement a web application for a small Content Management System (CMS) with a minimal set of functionalities. The application must satisfy the following requirements.

The CMS has a back-office (i.e., the administrative/management area) and a front-office (i.e., the version of the web application visible to everybody, without authentication).

In the back-office, the CMS allows any authenticated user to create and manage content (pages). Each **page** has the following properties:

- A *title*.
- An *author* (by default, the logged-in user).
- A *creation date*, i.e., the moment of creation.
- A *publication date*. According to this date, the page may be “draft” (empty publication date), “scheduled” (the publication date is in the future), or “published” (the publication date is today or in the past).
- Some blocks of *content*.

A block of **content** can be of three types: header, paragraph, or image. A page must have at least an header and at least one of the other two types of blocks. Images should be selected from a list of preloaded images (at least four different images must be available). Blocks can be ordered during the creation and the editing of the page (see the next paragraph). Header and paragraph blocks must contain text.

Each authenticated user, after log-in, can see a list of all the created pages (i.e., by any author) in a dedicated screen with the options to:

- **Create a new page**, by inserting all the needed properties, adding at least one header block, and at least one of the other blocks within the page. The creation date and the author cannot be changed. The blocks can be *re-ordered at any time* within the page during the creation process. How the re-ordering works is left to the student (e.g., arrows to move up and down the blocks). Blocks can also be removed, during the page creation process.
- **Edit an existing page for which they are the author**. By editing the page, the authenticated user can change all the properties (except the creation date and the author) and the content (including adding/removing and changing the order of blocks, but it must always contain at least an header and another type of blocks).
- **Delete a page for which they are the author.**

The CMS supports a special type of authenticated user, who is the **admin** of the entire application. There is no limit on the number of admins that can be present in the CMS. Any admin can perform all the operations of an authenticated user and, in addition:

- **Edit or delete any page**, i.e., even if they are not the author.
- **Assign the authorship of a page to a different user.**
- Set up the **name of the website**, which must appear at the top of any screen (in the back-office and in the front-office).

In the front-office, instead, both authenticated and non-authenticated (anonymous) users will see the entire website with the defined name, a list of all *published* pages in chronological order (by publication date), and can read the full content of each page (and its properties).

The organization of these specifications in different screens (and possibly on different routes) is left to the student.

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 "two servers" 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 undefined. 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 two subdirectories (client and server). The project must be started by running the two 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. You may assume that nodemon is globally installed.
- 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 file, so that the npm install command might install them.
- User authentication (login and logout) and API access must be implemented with passport.js and session cookies. The credentials should be stored in encrypted and salted form. The user registration procedure is not requested.

Database requirements

- The project database must be implemented by the student, and must be pre-loaded with at least 4 users, with at least one who authored two pages, one who authored no pages, one who is an admin, and 2 pages per status (draft, published, programmed).

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 o the exchanged objects
    b. A list of the database tables, with their purpose
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
3. Overall:
    a. A screenshot of the **screen for creating a new page** and one for the **screen with the list of all pages**, both as a non-admin. These screenshots must be embedded in the README by linking two images committed in the repository.
    b. Usernames and passwords of the users.

Submission procedure

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: `final` is all-lowercase, and it is a git 'tag', nor 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
```

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 and run 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; npm run dev)
(cd server ; npm install; nodemon index.js)
```

Ensure that all the needed packages are downloaded by the npm install 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.

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.