Web Applications I 2024/25 – Exam #4 (deadline 2025-01-11 at 23:59)

“Yearly Recap”

⚠️ DRAFT VERSION – Write your questions as COMMENTS in the document. Please do not delete nor resolve existing comments. The final version will be published on 2025-12-29.

Link for enrolling in the GitHub Classroom: https://classroom.github.com/a/9Pnry1-H

Develop a web application that allows users to create and publish an annual activity summary in the style of "Spotify Wrapped."

Each annual **recap** is linked to a specific **theme**. The application must include at least two different themes chosen by the student (anything that makes sense for a "My Year in..." format). Additionally, a recap must include the following mandatory properties:

- A **title**.
- At least **three pages** (each consisting of a background image with one or more overlapping text phrases).
- A **visibility** setting (public or private).
- The **author's name**.

The pages of the recap (background images and text) must be relevant to the theme chosen for the recap. For simplicity, the application provides a set of **predefined (non-editable) background images**. Each image defines the number of supported text fields (1, 2, or 3), and defines the specific position of these fields (different images may place text in different locations).

Authenticated users can *create* new annual recaps and *view* existing ones in a slideshow format.

To create a new recap, a user can:

1. **Start from a Pre-set Recap Template** and customize it**.** A *template* includes the minimum elements for a recap (3 pages with background images and placeholder text) and must be linked to one of the application's themes. The template theme must be visible during the template selection process.The pages of each template (both images and texts) must be relevant to the theme.
2. **Start from a Public Recap**. The user can choose a public recap created by someone else and customize it. In this case, the new recap must track the title and author of the original source and must retain the original theme.

During creation, the user can change background images (selecting only from those relevant to the theme of the original template or recap), edit text content, add or remove pages (maintaining a minimum of three). All text fields are optional (e.g., if an image supports two fields, the user can fill only one), but **at least one** field in the entire recap must be completed.

Before saving, the user must select the visibility. **If Private**, the recap is saved in a specific area of the user's profile, visible only to that user after authentication., if **Public**, the recap is displayed on the application's **homepage**, accessible even to non-authenticated users.

Any visitor to the site (authenticated or not) can view the list of public recaps and watch them in slideshow format.

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. APIs should be carefully protected and the front-end should not receive unnecessary information.
- The application should be designed for a desktop browser. Responsiveness for mobile devices is not required nor evaluated.
- The project must be implemented as a React 19 application that interacts with an HTTP API implemented in Node+Express. The Node version must be the one used during the course (22.x, LTS). The database must be stored in a SQLite file. The programming language must be JavaScript.
- 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.mjs`” 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. No other global modules will be available.
- 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. All required 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 an encrypted and salted form. The user registration procedure is not requested nor evaluated.

Quality requirements

In addition to the implementation of the required application functionality, the following quality requirements will be evaluated:

- Database design and organization.
- Design of the HTTP APIs.
- Organization of React components and routes.
- Correct usage of React patterns (functional behavior, hooks, state, context, and effects). Avoiding direct manipulation of the DOM is included in these rules.
- Code clarity.
- Absence of errors (and warnings) in the client console (except those caused by errors in the imported libraries).
- Absence of application crashes or unhandled exceptions.
- Essential data validation (in Express and React).
- Basic usability and user-friendliness.
- Originality of the solution.

NOTE: In the evaluation of the project, during the oral discussion, it is expected that the project has been fully developed by the candidate student, who should therefore possess detailed knowledge about the code, and especially about the design decisions and the adopted functions or solutions. It is not forbidden to collaborate with other students (but not sharing whole parts of the project), nor to utilize AI coding assistance, but the student still has 100% responsibility for the knowledge, understanding and capacity to explain the code.

Database requirements

The project database must be designed by the student and must be pre-populated (seeded) with the following initial data:

- **Users:** At least **3 users**.
- **Recaps:** Each user must have created at least **3 recaps**.
    - For each user, at least **one** of these recaps must have been created by **deriving** it from another user's public recap.
- **Themes:** At least **2 predefined themes**.
- **Templates:** At least **2 different templates** for each theme.
- **Images:** At least **12 predefined images per theme**, distributed as follows:
    - At least **4 images** with 1 text field.
    - At least **4 images** with 2 text fields.
    - At least **4 images** with 3 text fields.

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 item should take no more than 2-3 lines.

1. Server-side:
    a. A list of the HTTP APIs offered by the server, with a short description of the parameters and 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. Two screenshots of the application, one **during the creation of a recap** and one **showing the slideshow** of an existing recap. The screenshot must be embedded in the README by linking the image committed in the project repository.
    b. Usernames and passwords of the registered users.

Submission procedure

To correctly submit the project, you must:

- **Be enrolled** in the exam call.
- Use the provided **link** to **join the classroom** on GitHub Classroom (i.e., correctly **associate** your GitHub username with your student ID) and **accept the assignment**.
  https://classroom.github.com/a/9Pnry1-H
- **Push the project** into 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 with no spaces, 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.mjs)
```

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.

Be aware that Linux is case-sensitive for file names, while Windows and macOS are not. Double-check the case of import statements and all file names.