01UDFOV/01TXYOV – WEB APPLICATIONS I

TASK MANAGER APP: THE END

*Duration of this lab: 3 hours (**2 weeks**)*

During this last lab, you will update your React-enabled task manager to support multiple "pages" (through routes), to add login features, and to enable new functions. This lab will last **2 weeks**, and it will complete our task manager app. In the *first* week, you should be able to complete the Exercise 1 and to start Exercise 2. Exercise 3 as well as the completion of Exercise 2 should be done in the second week.

EXERCISE 1 – ROUTES

Update the task manager you developed in the previous lab to support multiple "pages". In particular, *restructure* the React application to implement each filter and the modal for adding a new task as separate pages. List all the URLs that your application is going to serve in the README.md file, with a 1-line description for each page.

In addition, add a new page with a form, which will be used to log in a user. The page should be well structured in term of needed components and suitable states. The login form will have two *mandatory* fields: email and password, both validated in a proper way from the client.

The next exercise will focus on the login process, for now, just add the login page and the needed routes to support the following use case: when a user submits a properly filled form, she will be redirected to the task list and she will see a "*Welcome, User!*" message on the top of the page.

Hints:

1. *You can use the solution available for Lab 9 as a starting point, if you prefer: https://github.com/polito-WA1-2020/lab9-react-meets-rest*
2. *As a starting point, you might want to have a look at the exercise developed during the lectures:* https://github.com/polito-WA1-2020/client-server-example/tree/with_router

EXERCISE 2 – LOGIN

Implement the login process, as shown in the lectures, to allow authorized users to see their tasks. In particular, by exploiting JWT, you should perform the following changes and additions to the database, the server, and the client. For the client, you might consider using *hooks* for handling the state.

1. [**Database**] Create a new user table (mandatory columns: *id, name, email, hash*), with at least one user. Use `bcrypt` to generate the hash of a password of your choice, as we do not store plain text passwords in the database.
2. [**Database**] Edit the task table so that a user may have one or more tasks. Associate the newly created user to all tasks.
3. [**Express**] Create a `getUser()` and a `checkPassword()` method to get, respectively, information about a specific user from the database and for checking if the user's password received via the API

corresponds to the hash stored in the database for the same user. Install and use the `bcrypt` module to check the hashes.

4. [**Express**] Implement a route for logging in a user and, in case of success, to create and send a JWT token to the client. Set the expiration of the token to 7 days, put the user id and name in the payload of the token, and send it in a cookie.
5. [**Express**] Protect the other routes so that they will be accessible only if a valid token is passed.
6. [**Client**] Edit the logic behind the login form to send the email and the password inserted in the form to the server, via fetch.
7. [**Client**] Handle the receipt of a wrong or successful login. In the former case, the client should display a suitable error message (e.g., "*Wrong username*", "*Wrong password*" or similar) and continue to show the login form. In the latter case, the client should handle the received JWT token, and show a "*Welcome, {name of the user}*" message upon redirect to the page containing the list of tasks.

OPTIONAL

In the client, create a logout link, which will ask the server to delete the JWT token (stored in the cookie) and perform a redirect to the login form.

Hints:

1. *You can use the following website to generate the hash of a password, according to bcrypt:* https://www.browserling.com/tools/bcrypt*. If the generated hash starts with $2y$, replace that part with $2b$, as the node bcrypt module does not support $2y$ hashes.*
2. *You can install the bcrypt node module with npm. Documentation is available at https://github.com/kelektiv/node.bcrypt.js*
3. *For JWT in the server, you might want to install and use the following two modules: https://www.npmjs.com/package/jsonwebtoken and https://www.npmjs.com/package/express-jwt*
4. *As a starting point, you might want to have a look at the exercise developed during the lectures: https://github.com/polito-WA1-2020/react-scores/tree/with_auth and https://github.com/polito-WA1-2020/react-scores-server/tree/with_auth*

EXERCISE 3 – SHARING TASKS

Update the web app developed so far to enable the *shared tasks* functionality. In particular, tasks publicly shared (i.e., tasks whose *private* property is false) should be visible to any user (including non-logged ones), while the other tasks will be visible to their owner, only.

Create a separate page (and route) to visualize such publicly shared tasks, e.g., named "public tasks". If needed, update the server code responsible to publish any related API and to perform any database query.

OPTIONAL

If you want to go the extra mile, you can refine this exercise to enable the *user-to-user sharing* of a task. That is, a user can set one of her tasks as shared with another *specific user*, e.g., user #1 can share the task A with user #2. Such tasks will be visible in two new filters, which will replace the existing "Shared with…" option: "*Shared with me*" and "*Shared with others*".

Edit the client and the server to support the creation and the consumption of such tasks. As before, every user (registered or not), will see the publicly shared tasks in a separate page.