BigLab2

Thursday, May 27, 2021 11:41 AM

01UDFOV/01TXYOV – Web Applications I

BigLab 2: Full Stack Todo List

During the four weeks of the second BigLab, you will develop the back-end for your web-based task manager using Node + Express. In addition, you will improve the front-end of the task manager accordingly and connect the front-end to the back-end. As for the previous BigLab, to create your repository, you must login to GitHub Classroom and accept the assignment (if needed, join your group). For more details, please have a look at the GitHub Classroom instructions. Here you can find the links for the BigLab2 repository on GitHub Classroom:

- Web Applications I: https://classroom.github.com/g/NFnVVlQH
- Applicazioni Web I [A-L]: https://classroom.github.com/g/hZPQRYna
- Applicazioni Web I [M-Z]: https://classroom.github.com/g/1mhnogjm

To better keep track of your progress, we suggest you work incrementally "week-by-week", e.g., by creating, inside your repository, a branch for each week of the BigLab.

What are we building in these weeks?

a) In the first week, we will create a **basic back-end for the task manager**. To do so, we will use the Express framework. The back-end has to implement a series of **APIs** to support the main features of the web-based task manager we developed in BigLab1: **create**, **read**, **update** and **delete** the tasks. The data will be persistently stored in an **SQLite database**.
b) In the second week, we will **update the front-end** of the web-based task manager (i.e., the final result of the BigLab1) to use some of the **APIs** designed in the previous week. In particular, we will **get the tasks** to be displayed from the server, and we will **save new tasks** on the server-side database.
c) In the third week, we will continue to **update the front-end** of the web-based task manager to **use all the APIs** designed in the first part of this BigLab. Specifically, we will use the APIs for filtering tasks, and for deleting and updating them.
d) In the fourth and last week, we will add the possibility of **having multiple users** to both our back-end and front-end applications, enabling them to **authenticate** (login and logout functionalities) and manage their own tasks. Task list page **access will then be refused** to non-authenticated users.

Evaluation Criteria & Deadlines

The points received for your work are added to the final exam score to each member of the team.
We will follow these evaluation criteria for evaluating your submission:

- The team members will receive 1 point if the submitted application is *complete*, i.e., it successfully implements *all functionalities of the 4 weeks*, i.e., points a), b), c), and d).
- The team members will receive 0.5 points if the submitted React + Express application is *partially complete*, i.e., it successfully implements the functionalities of *at least 2 weeks* (e.g., the team implemented only points a and b).

- The team members will receive 0 points otherwise.

The assignment must be submitted in the **master/main branch** before **Sunday, June 6 at 23:59 CEST** (see the the GitHub Classroom instructions for the details on the submission procedure). Remember that the last commit of your assignment must be **tagged** with "*final*".

If your **master/main branch is empty**, or you did not push a commit tagged with **final**, we assume that you decided **not to submit** the BigLab for evaluation.

The final repository structure **must** keep the client and server sub-directories of the provided template; thus, the project will need to execute correctly with the following commands:

i) for the back-end: "cd server; nodemon server.js;"
ii) for the front-end: "cd client; npm start;".

01UDFOV/01TXYOV – WEB APPLICATIONS I

BIGLAB 2A: APIS WITH EXPRESS

WHAT ARE WE BUILDING THIS WEEK?

We will create a **basic back-end for the task manager**. To do so, we will use the Express framework. The back-end has to implement a series of **APIs** to support the main features of the web-based task manager we developed in BigLab1: **create**, **read**, **update** and **delete** the tasks. The data will be persistently stored in an **SQLite database**.

STEP-BY-STEP INSTRUCTIONS:

- Design a set of APIs to support the main features of our web-based task manager. The APIs should allow the application to:
  - **retrieve** the list of all the available tasks;
  - **retrieve** a list of all the tasks that fulfill a given filter (e.g., all the important tasks, all the tasks with a given deadline, etc.);
  - **retrieve** a task, given its "id";
  - **create** a new task, by providing all relevant information – except the "id" that will be automatically assigned by the back-end;
  - **update** an existing task, by providing all relevant information (all the properties except the "id" will overwrite the current properties of the existing task. The "id" will not change after the update);
  - **mark** an existing task as completed/uncompleted;
  - **delete** an existing task, given its "id".

  Data *passed to* or *received from* the API should be in **JSON format**. You must list the designed APIs, together with a short description of the parameters and the exchanged entities, in the README.md file that is included in your repository. Be sure to identify which are the collections and elements you are representing. You might want to follow this structure for reporting each API:

  ```
  [HTTP Method] [URL, optionally with parameter(s)]
  [One-line about what this API is doing]
  [Sample request, with body (if any)]
  [Sample response, with body (if any)]
  [Error response(s), if any]
  ```

- Implement the designed HTTP APIs with Express. The tasks must be **stored persistently** in an SQLite database. To this end, you can use the database "tasks.db" that is included in your group's BigLab2 repository.
  The database contains two tables, "users" and "tasks", with each user that may have one or more tasks. For the moment, consider the information of the "tasks" table, only (the "user" column is always set to 1, the only user available). Besides the usual information (*id, description, important,*

private, deadline), the table also contains a column to mark a task as completed. The information about users will be used in the last week of this BigLab, to implement the login functionality.

- Test the realized API with a tool such as PostMan (https://www.postman.com/), ReqBin (https://reqbin.com) for Google Chrome or RestClient for Firefox (https://addons.mozilla.org/en-US/firefox/addon/restclient/).

Hints:

1. *The general specification of the second BigLab can be found at: https://github.com/polito-WA1-AW1-2021/course-materials/blob/main/labs/BigLab2/BigLab2.pdf*
2. *To visualize and edit the database, you can exploit the "DB Browser for SQLite" application, downloadable from https://sqlitebrowser.org/ or SQLiteStudio, available at https://sqlitestudio.pl/*

01UDFOV/01TXYOV – WEB APPLICATIONS I

BIGLAB 2B: APIS INTEGRATION

WHAT ARE WE BUILDING THIS WEEK?

We will **update the front-end** of the web-based task manager (i.e., the final result of the BigLab1) to use some of the **APIs** designed in the previous week. In particular, we will **get the tasks** to be displayed from the server, and we will **save new tasks** on the server-side database.

STEP-BY-STEP INSTRUCTIONS:

Modify the front-end of the web-based task manager so that:

- when the home page of the task manager loads, the available **tasks are retrieved from the server** by invoking the proper API and are then stored into the application state;
- when the user adds a new task, the **task is saved on the server-side database**, and the list of tasks is updated and displayed accordingly. To do so, invoke the proper API for adding tasks, and then retrieve the updated list of tasks from the server.

Beware: for the moment, all the other operations (i.e., filters, delete, and update of tasks) are still not linked to the server (i.e., they act on the local copy of the retrieved tasks).

Hints:

1. *The general specification of the second BigLab can be found at: https://github.com/polito-WA1-AW1-2021/course-materials/blob/main/labs/BigLab2/BigLab2.pdf*
2. *Trust the server! It shall be always up-to-date, and every operation rely on it, not on the internal state of the webapp.*

01UDFOV/01TXYOV – WEB APPLICATIONS I

BIGLAB 2C: COMPLETE APIS INTEGRATION

WHAT ARE WE BUILDING THIS WEEK?

We will continue to **update the front-end** of the web-based task manager to **use all the APIs** designed in the first part of this BigLab. In particular, we will use the APIs for filtering tasks, and for deleting and updating them.

STEP-BY-STEP INSTRUCTIONS:

Modify the front-end of the web-based task manager in the following ways.

- Minimize the "n-clients problem" shown during the lecture, by reloading the application state every time a filter change. In particular, when the user clicks on a filter, the list of **filtered tasks is retrieved from the server** by invoking the proper API.
- Make the delete operations persistent: when the user deletes a task, the **task is removed from the server-side database**, and the list of tasks is updated and displayed accordingly. To do so, invoke the proper API for deleting tasks, and then retrieve the updated list of tasks from the server.
- Make the update operations persistent: when the user updates a task, the **task is modified on the server-side database**, and the list of tasks is updated and displayed accordingly. To do so, invoke the proper API for modifying a task, and then retrieve the updated list of tasks from the server.
  Beware. A task can be updated in two different ways:
  - by updating its fields through the dedicated modal;
  - by marking it as *completed/not completed*. Tasks can be marked as *completed/not completed* by using the checkboxes that are displayed near each task in the list. A checkbox is checked if the corresponding task is completed, unchecked otherwise.

Hints:

1. *The general specification of the second BigLab can be found at:*
   https://github.com/polito-WA1-AW1-2021/course-materials/blob/main/labs/BigLab2/BigLab2.pdf
2. *Trust the server! It shall be always up-to-date, and every operation rely on it, not on the internal state of the webapp.*

01UDFOV/01TXYOV – Web Applications I

BigLab 2d: Now With Authentication

What are we building this week?

We will add the possibility of **having multiple users** to both our back-end and front-end applications, enabling them to **authenticate** (login and logout functionalities) and manage their tasks. Access to the page showing the list of tasks **will then be refused** to non-authenticated users.

Step-by-step instructions:

- In the front-end, add a new page (with a dedicated route) with a form, which will be used to log in a user. The page should be well structured in terms of needed components and appropriate states. The login form will have two *mandatory* fields: **email** and **password**, both **validated** properly by the client.
- Implement the **login** process by exploiting the **Passport** authentication middleware in express, as shown in the lectures, and the "users" table of the provided database. You are free to add new tasks and users inside your database.
  Beware: *do not store* **plain text passwords** *in the database*! Use **bcrypt** (see the hints) to generate a hash of the passwords before saving them. For testing the application, list the usernames and the plain text passwords of the example users in the project README.md file included in the repository.
- When the login process **fails**, the front-end should display a suitable error message (e.g., "*Incorrect username and/or password*" or similar) and continue to show the login form. Instead, when the login is **successful**, the application redirects the user to her task list page, showing a message like: "*Welcome, {name_of_the_user}*".
- Implement the **logout** functionality, again by exploiting the **Passport** authentication middleware. When the user is logged out, redirect her to the login form.
- Identify the routes that need to be authenticated (e.g., those to get the list of tasks) and *protect* them accordingly. Non-authenticated users must not see any task, meanwhile authenticated users must see only their tasks.

Hints:

1. *The general specification of the second BigLab can be found at:* *https://github.com/polito-WA1-AW1-2021/course-materials/blob/main/labs/BigLab2/BigLab2.pdf*
2. *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.
3. *You can install the bcrypt node module with npm. Documentation is available at* *https://github.com/kelektiv/node.bcrypt.js*

01UDFOV/01TXYOV – WEB APPLICATIONS I

DEPLOYMENT ON THE CLOUD

WHAT ARE WE BUILDING THIS WEEK?

In this lab, we will deploy the web-based task manager developed during BigLab2 on the cloud. Specifically, you will launch the app on a *Platform as a Service* cloud platform called **Heroku**. In this manner, the web-based task manager you have implemented will be accessible for everyone on the Web. The following are the instructions you should follow to register on Heroku, configure the deployment environment, and launch the application.

SETTING UP THE HEROKU CLI

- Create a Heroku account (each member of the group). To do so, you must go to their site and create a free account (https://signup.heroku.com/login).
- Download and install the Heroku Command Line Interface (CLI); it enables you to create and manage your Heroku apps directly from the terminal. The simplest way to install the Heroku CLI is by running the command **npm install -g heroku**[1].
- Once you installed the CLI, run in the terminal the **heroku login** command. You will have to press *any key* (and then press "enter") to go to your web browser to complete the login. The CLI will then log you in automatically.

SETTING UP YOUR PROJECT

- Create a new branch in your Big Lab 2 repository (e.g., *deploy*) and pull it.
- Re-arrange the structure of your BigLab2 project. Move out all the files within the *server* folder to the *root* of your project, then delete the empty *server* folder.
- In the **package.json** file that you have just moved in the *root*, add the following line inside the scripts **"heroku-postbuild": "cd client && npm install && npm run build"**.
  Your file should look like:

```
"scripts": {
  "start": "node server.js",
  "heroku-postbuild": "cd client && npm install && npm run build",
  "test": "echo \"Error: no test specified\" && exit 1"
},
```

- Inside the client **package.json** remove the line related to the proxy.

[1] However, feel free to try other installation methods indicated in the official documentation (https://devcenter.heroku.com/articles/heroku-cli)

- To access properly the client folder from the server, it is necessary to install the *path* module[2].To do so, open another terminal and install the **path** module (**npm install path**), then import it in server.js (i.e., const path = require('path').
- In the same file, modify the **port** constant by setting:
  const PORT = process.env.PORT || 3001;
- Additionally, indicate Express how to serve static files by adding:
  app.use(express.static("./client/build"));
- Finally, **below all the endpoint definitions**, add a new endpoint to declare that any request that does not match any other endpoints send back the client React application index.html file. The modifications in your **server.js** file should look like the snippet below.

```
const path = require('path');
const port = process.env.PORT || 3001;

app.use(express.json());
app.use(express.static("./client/build"));

[…]

app.get('*', (req, res) => {
    res.redirect('index.html');
});
```

DEPLOYING THE APPLICATION ON HEROKU

- Open the terminal, step up in the root of your project, and execute the following commands:

```
heroku create
git add --all
git commit -m "Preparing deploy to Heroku"
git push heroku <your_new_branch>:<heroku_main_branch>
```

- To determine whether the <heroku_main_branch> is master or main, run the command **git branch --all**. It shows both the local and the remote branches of your repository. The starred branch (*) is your current branch. You should see also a remote branch related to Heroku called "main" or "master".
- After executing them, Heroku will install all the required node modules and will create and deploy an instance with your web-based task manager. Run the command **heroku open** to open the application in the browser.

Hints:

1. *You can find the deployment of a complete working sample application at: https://github.com/mars/heroku-cra-node*
2. *You can check your logs for details from the Heroku CLI with the command:* **heroku logs --tail**

[2] The path module provides utilities for working with file and directory paths.