Web Applications I – Exam # 4 (deadline 2021-01-20 at 23:59)

"Pizza"

FINAL VERSION – Additions and corrections are shown in red

Design and implement a web application for managing orders in a pizza take-away shop. Note that, for simplicity, there is no need for handling dates: it is assumed that all orders are placed in the same (unspecified) day.

The application needs to satisfy the following requirements:

- **Pizza types**
    - Pizza sizes can be Small, Medium, Large (S, M, L).
    - Each pizza can be ordered with a set of toppings: up to two for the Small one, up to three for the Medium one, and up to three for each of the two parts of a Large one. In case of a Large pizza that is not split in two parts, the total number of toppings is six.
    - Toppings are: olives, ham, bacon, mushrooms, egg, artichokes, seafood, chips, vegetables. Toppings can be combined as desired within the numeric limits stated for each size of pizza but with the following additional constraint: seafood can only be requested for the Large pizza and, if used, must be present in both parts. In addition, it is possible to ask for a pizza (of any size) with no tomato at all. Tomato does not count as a topping.
    - Several pizzas with the same size and toppings can be selected by simply modifying a suitable counter while ordering: for instance, it must be possible to order 3 S pizzas with olives and mushrooms without entering the same toppings three times but just selecting 3 as the number of pizzas.
- Every day, the shop can make a maximum number of pizzas of each size: respectively, 10 S, 8 M, 6 L. This information must be stored in the database, together with the prices of the pizzas, detailed later.
- **A generic user (authenticated or not) can freely browse the availability** of the three sizes of pizzas, as well the list of toppings that can be used on each size of pizza.
- An authenticated user can make its own pizza order by creating a list of pizzas, specifying their characteristics in an ***interactive configuration page***
    - All the user interaction for pizzas configuration must be handled client-side, except for availability checking of the pizza size, that must be done in real time ~~whenever new pizzas are requested~~ when the pizza order is complete and is being submitted, to avoid letting the user configure a number of pizzas which are actually not available.
    - While the user configures the order, the price is updated in real time at every change (addition/deletion, or modification of the toppings or the quantity). The total price is the sum of the costs of all pizzas in the order. Each pizza size (S, M, L) has a price which is stored in the database: S is 4 €, M is 6 €, L is 10 €. Seafood increases by 20% the total price of each pizza where it is present. A 10% discount on the total of the order is applied if more than 3 pizzas are included in the order.

- Then, when the user submits the order for processing to the shop, the shop checks that enough pizzas of the requested sizes are still available and confirms the order, otherwise a suitable informative message is shown (e.g., “not enough L pizzas”), and the user is sent back to the configurator which must allow to correct the order ~~just sent~~.
- 5 seconds after the order confirmation, a message must appear on screen about the fact that that the pizzas are ready to be collected for the user identified by his email. The message should stay on screen until the close button is clicked.
- Users can always check their list of past orders: the list should show a row for each order. The row shows the total number of pizzas with the total paid price. A detailed view of the order must be shown (and hidden) when the row is clicked, and show the same information that was presented in the configurator.
  Tip: an immutable configurator view can be recycled for this purpose if deemed appropriate.

Project requirements

- The application architecture and the source code should be developed by adopting the best practices in software design, in particular for single page applications using React and REST.
- The project must be realized as a React Application, interacting with a REST API implemented in Node+Express. The database should be stored in an SQLite file.
- The communication between client and server must follow the “React Development Proxy” pattern and React must run in development mode.
- The top-level directory must have a `README.md` file and contain two sub-directories (`client` and `server`). The project must be run with the following commands: “`cd server; nodemon server.js`” and “`cd client; npm start`”. A skeleton of the project directories is provided.
- The whole project must be submitted on GitHub, in the repository created by GitHub Classroom.
- The project **must not include** the `node_modules` directories. They shall be re-populated by running “`npm install`”, immediately after “`git clone`”.
- The project may use popular and commonly adopted libraries (such as `moment.js`, `react-bootstrap`, etc.), if applicable and useful. Such libraries must be correctly declared in `package.json` so that `npm install` may download them.
- User login and access should be protected with a JWT token, stored in an http-only cookie. No additional protection is required.
- The project data-base must be defined by the student and must be preloaded with at least 5 test users (the registration procedure is not required), with at least one user who made 2 orders, one who made 1 order, and a total of 10 ordered pizzas.

Contents of README.md

The README.md file must contain the following information (a template is available in the project skeleton – in general each explanation should be no longer than 1-2 lines):

1. A list of the React Application Routes, with a short description of each route’s purpose
2. A list of REST APIs offered by the server, with a short description of the parameters and the exchanged entities
3. A list of the database tables, with their purpose
4. A list of the main React components adopted in the application.

5. A screenshot of the **page for configuring a pizza** (embedding a picture stored in the repository).
6. The usernames and passwords of the test users, indicating which users ordered something and how many orders each of them did.

Submission procedure (important!)

For successfully submitting the project, you need to:

- be **enrolled** for the exam
- **push the project** to the `master` branch of the repository that GitHub Classroom generated for you. The last commit (the one that you want corrected) must be **tagged** with the `final` tag

Note: for tagging 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 a tag from the GitHub web interface (follow the link 'Create a new release').

If you want to check your submission, these are the commands that WE will use to download your project... you might want to try them in an empty directory:

```
git clone ...yourCloneURL...
cd ...yourProjectDir...
git pull origin master  # just in case the default branch is not master
git checkout -b evaluation final # check out the version tagged with
'final' and create a new branch 'evaluation'
(cd client ; npm install)
(cd server ; npm install)
```

Ensure that **all** the needed packages will be retrieved with the `npm install` commands. Be careful if some packages have been installed globally as they might appear not being needed as dependencies: you may want to test the procedure on a fresh install (e.g., a VM).

The project will be tested under Linux (beware that Linux is case-sensitive in file names, while MacOS-X and Windows are not).