Web Applications I – Exam # 3 (deadline 2024-09-17 at 23:59)

“Instant Lottery”

FINAL VERSION – Modifications are shown in red.

Create a web application to play an on-line version of an Instant Lottery. The game consists in a periodic drawing (extraction) of a set of numbers, where the players may bet on some of those numbers on each of these drawings. All website functionalities are reserved to logged-in users (players). Non-logged users will only see a message with the rules of the game and the login form.

More in detail: every 2 minutes, the web application shows to the users a set of **5 drawn numbers** in the 1-90 range, that are distinct and *randomly* generated by the *server*. All players connected to the web application at that moment will therefore see the same 5 numbers.

Before the drawing instant, players may place a bet on 1, 2 or 3 distinct numbers. For each drawing a player may place at most one bet. The check on the correctly guessed numbers must be done by the server, after the new drawing.

The bets are paid with some ‘points’ owned by the players: betting on one number costs 5 points, betting on 2 numbers costs 10 points, and betting on 3 numbers costs 15 points. The player may place the bet only if he/she has sufficient available points. Each player starts with an initial budget of 100 points; the budget may be increased by winning bets, only. Once the budget reaches zero, the player can no longer place any bet.

At each drawing, three cases are possible:

1. All the numbers in the bet are correct; in this case, the application shows a suitable message and the player wins twice the points.
2. None of the numbers in the bet is correct; in this case, the application shows a suitable message and no points are won.
3. Some of the numbers in the bet are correct, but not all of them; in this case, the application shows a suitable message and the player wins in proportion to the number of correct numbers. In particular, the gain is computed according to the formula *2*(spent points)*(correct numbers)/(played numbers)*.

The following table summarizes all possible cases.

Numbers in the bet	**Correct numbers**	**Gain**
1	1	10 points
2	1	10 points
2	2	20 points
3	1	10 points
3	2	20 points

3	3	30 points
1, 2 o 3	0	0 points

The web application will also provide a page with the ranking of the 3 best players, ranked by descending number of currently owned points.

The application should prevent malicious behaviors (access to private information, bets outside the correct times, cheating on scores or on the numbers in the bet, for example) through a suitable API design and all necessary checks.

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 application that interacts with an HTTP API implemented in Node+Express. The Node version must be the one used during the course (20.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 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.

Database requirements

- The student must implement the project database and be pre-loaded with at least 5 registered users. At least 3 users already placed some bets.

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 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 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 while placing the bet and the other after the drawing of the numbers**, respectively. The screenshots must be embedded in the README by linking two images committed in the 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**.
- **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.