

Web Applications I – Exam # 2 (deadline 2023-07-09 at 23:59)

“Airplane Seats”

FINAL VERSION – Modifications are reported in “red”

Design and implement a web application for booking seats on a plane. The application must meet the following requirements.

For simplicity, an airplane is composed of a set of seats arranged in a grid format with F rows and P seats per row. The number of rows and seats per row depends on the type of plane being booked. There are three types of planes: *local* ($F = 15$, $P = 4$), *regional* ($F = 20$, $P = 5$), and *international* ($F = 25$, $P = 6$).

On the main page of the website, accessible without authentication, it is possible to select a plane from the available types. The page shows the seat availability for the selected plane. The graphical implementation of the two-dimensional seat visualization is left to the student. Each seat should have a code indicating its row (1, 2, 3, ...) and position within the row (A, B, C, ...). For example, the first seat in the 10th row will have the code “10A,” while the sixth seat in the 3rd row will have “3F.” In the non-authenticated view, each seat should display its status (either occupied or available) in addition to its code. The status can be represented by colors, icons, or any other visual elements chosen by the student. The seat visualization page must also ~~continuously~~ display the number of occupied seats, available seats, and the total number of seats on the plane.

After authentication (login), a user should be able to select a plane from the available types and make a new reservation for *one or more seats* using one of the following two alternative methods:

1. Specify the number of seats to be booked, and the system will automatically assign seats (if enough seats are available). The logic for seat selection is left to the student (random, by row, etc.).
2. Directly interact with the seats using a two-dimensional visualization similar to the non-authenticated seat visualization page (more details in the next paragraph).

In the latter case, the reservation process must take into account the seats’ status (displaying them accordingly). In this case, seats can have three states:

- Occupied seats cannot be requested.
- Available seats can be requested by clicking on them. Once clicked, the seat should *immediately* be displayed as Requested on the screen.
- Requested seats, which can be released if clicked (becoming Available again).

The seat visualization should always display the number of occupied, available, requested, and total seats. These numbers should be updated every time the user performs an action on the seats.

Note that while the user is booking seats, other registered users may request and/or occupy the same seats. This is normal and should neither be prevented nor notified to the user.

Once the seat selection operations are completed using either of the two methods, the user can *confirm* or *cancel* their reservation. If, at the time of confirmation, all selected seats are still available, they

should all become occupied simultaneously. Otherwise, the reservation will be canceled, and the cancellation reason should be indicated by highlighting the seats already occupied by others on the two-dimensional seat visualization (for a duration of 5 seconds).

A logged-in user may additionally delete his/her own reservation (if it exists), thereby releasing all seats that were occupied by him/her. Deleting the reservation will make the seats available again and update the visualization accordingly. As a general constraint, an authenticated user can only make one seat reservation (including one or more seats, as before) per plane.

The organization of these specifications into different screens (and potentially different routes) is left to the student and is subject to evaluation.

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.
- The project must be implemented as a React application that interacts with an HTTP API implemented in Node+Express. The database must be stored in a SQLite file.
- 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.js` 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.
- 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. Such 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.

Database requirements

- The project database must be implemented by the student, and must be pre-loaded with *at least* 2 users who have a confirmed reservation of at least 2 seats each, on two different planes; and with at least 2 other users without any reservation.

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 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 o 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. A screenshot of the **screen for booking the seats**. The screenshot must be embedded in the README by linking the image committed in the repository.
 - b. Usernames and passwords of the users.

Submission procedure

To correctly submit the project, you must:

- **Be enrolled** in the exam call.
- **Accept the invitation** on GitHub Classroom, and correctly **associate** your GitHub username with your student ID.
- **Push the project** in 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, 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.js)
```

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.

The project will be tested under Linux: be aware that Linux is case-sensitive for file names, while Windows and macOS are not. Double-check the case of import and require() statements.