Web Applications (2024/2025) – Sample Exam (adapted from Jun 2022)

"StudyPlan"

FINAL VERSION - edits are in red.

Design and implement a web application to manage the study plan of a university student.

The application must satisfy the following requirements.

The university offers a series of courses. Each course is characterized by a unique 7-characters code, a name, and the (integer) number of credits.

A student's study plan is a subset of the courses offered by the university. The total number of credits of the courses inserted in the study plan can range from 60 to 80 credits (extremes included) for the full-time option, or from 20 to 40 credits (extremes included) for the part-time option.

A course can have one or more constraints for its insertion in the study plan:

- A course can be *incompatible* with one or more courses. They cannot be selected together.
- A course can have one mandatory *preparatory* course, which must be already present in the study plan.
- A course can have a maximum number of *students* able to add it into the study plan.

In the homepage of the application, unauthenticated (anonymous) users see all the courses that the university offers. This list of courses must be displayed in alphabetical order (by course name). For each course, the list shows its description: the code, the name, the number of credits, the number of students that already chose the course and, if present, the maximum number of students that can select it. Each course description may be expanded/contracted by the user, to show any incompatible and/or preparatory courses (display at least their code). Many courses may be in the expanded state at the same time.

Once logged in (logged-in home page), users continue to see the same full course list. In this page, if no study plan has been created yet, the user may create an empty one, by specifying the full-time or part-time option; this empty list can be edited according to the following instructions. If a study plan has already been created and persistently saved, it is immediately displayed (in the same page) and can be edited as below.

The editing of the displayed study plan allows the following operations:

- Always display the number of credits corresponding to the courses in the study plan, and the minimum-maximum allowed values.
- Add a course from the full list to the study plan. Only courses that satisfy all constraints can be added.
- Remove a course from the study plan, if it does not violate any "preparatory" constraint (otherwise, the application should display the reason).

- If a course cannot be added, it will be marked differently in the full list, and the application should display the reason.

During the editing session, the user may "Save" the study plan in a persistent way (this will replace any possible previous version). The user may "Cancel" the current modifications, and in this case the persistent copy (if any) must not be modified.

When saving, the study plan must be validated according to the min-max number of credits.

Additionally, the user may "Delete" the entire study plan, including the persistent copy.

After each of these actions, the application will be in the logged-in home page.

The delete action on the entire study plan can be performed only if the user performed 2FA at authentication time. Note that all students have the option, at login time, to authenticate using the 2FA procedure with a TOTP. Also, any student can also choose to authenticate without using the 2FA procedure at login time, thus acting as authenticated users with username/password only.

The organization of these specifications into different screens (and potentially different routes), when not specified, is left to the student and is subject to evaluation.

For the 2FA procedure, for simplicity, use the following secret for all users that require it: `LXBSMDTMSP2I5XFXIYRGFVWSFI` (it is the same used during lectures and labs).

Project requirements

- User authentication (login and logout) and API access must be implemented with `passport.js` and **session cookies, using the mechanisms explained during the lectures**. Session information must be stored on the server, as done during the lectures. The credentials must be stored in hashed and salted form, as done during the lectures. **Failure to follow this pattern will automatically lead to exam failure.**
- The communication between client and server must follow the multiple-server pattern, by properly configuring CORS, and React must run in "development" mode with Strict Mode activated. **Failure to follow this pattern will automatically lead to exam failure.**
- 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 an SQLite file. **Failure to follow this pattern will automatically lead to exam failure.**
- 2FA verification must be implemented by using the libraries explained during the lectures, and with the secret specified in the exam text. **Failure to follow this pattern will automatically lead to exam failure.**
- The evaluation of the project will be carried out by navigating the application, **using the starting URL http://localhost:5173**. Neither the behavior of the "refresh" button, nor the manual entering of a URL (except /) will be tested, and their behavior is not specified. Also, the application should never "reload" itself as a consequence of normal user operations.
- The user registration procedure is not requested *unless specifically required in the text*.

- 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.
- Particular attention must be paid to ensure server APIs are duly protected from performing unwanted, inconsistent, and unauthorized operations.
- The root directory of the project must contain a README.md file and have the same subdirectories as the template project (`client` and `server`). The project must start by running these 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. Do not move or rename such directories. You may assume that `nodemon` is globally installed.
- The whole project must be submitted on GitHub in the repository created by GitHub Classroom specifically for this exam.
- The project **must not include** the `node_modules` directories. They will be re-created by running the "`npm ci`" command, right after "`git clone`".
- The project **must include** the `package-lock.json` files for each of the folders (client and server).
- 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` and `package-lock.json` files, so that the `npm ci` command can download and install all of them.

Database requirements

- The database schema must be designed and decided by the student, and it is part of the evaluation. Some content must be preloaded in the database.

- The database must be pre-loaded with *at least five* students, at least one with a part-time study plan, at least one with a full-time study plan. At least two courses should have reached the maximum number of enrolled students.

- The database must include, at least, the following courses:

Code	**Name**	**Credits**	**Max students**	**Incompatible with**	**Preparatory course**
02GOLOV	Architetture dei sistemi di elaborazione	12		02LSEOV	
02LSEOV	Computer architectures	12		02GOLOV	
01SQJOV	Data Science and Database Technology	8		01SQMOV 01SQLOV	
01SQMOV	Data Science e Tecnologie per le Basi di Dati	8		01SQJOV 01SQLOV	
01SQLOV	Database systems	8		01SQJOV 01SQMOV	
01OTWOV	Computer network technologies and services	6	3	02KPNOV	
02KPNOV	Tecnologie e servizi di rete	6	3	01OTWOV	
01TYMOV	Information systems security services	12		01UDUOV	
01UDUOV	Sicurezza dei sistemi informativi	12		01TYMOV	
05BIDOV	Ingegneria del software	6		04GSPOV	02GOLOV

Code	Name	Credits	Max students	Incompatible with	Preparatory course
04GSPOV	Software engineering	6		05BIDOV	02LSEOV
01UDFOV	Applicazioni Web I	6		01TXYOV	
01TXYOV	Web Applications I	6	3	01UDFOV	
01TXSOV	Web Applications II	6			01TXYOV
02GRSOV	Programmazione di sistema	6		01NYHOV	
01NYHOV	System and device programming	6	3	02GRSOV	
01SQOOV	Reti Locali e Data Center	6			
01TYDOV	Software networking	7			
03UEWOV	Challenge	5			
01URROV	Computational intelligence	6			
01OUZPD	Model based software design	4			
01URSPD	Internet Video Streaming	6	2		

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 of the exchanged objects.
    b. A list of the database tables, with their purpose, and the name of the columns.
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 developed for the project. Minor ones can be skipped.
3. Overall:
    a. A screenshot of the **logged-in home page during an editing session**. 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 (IMPORTANT!)

To correctly submit the project, you must:

- **Be enrolled** in the exam call.
- **Accept the invitation** on GitHub Classroom, using the link specific for **this** exam, and correctly **associate** your GitHub username with your student ID.
- **Push the project** in the **branch named "main"** 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 whitespaces, and it is a git 'tag', NOT a 'commit message'), otherwise the submission will not be evaluated.

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
```

NB: **the tag name is "final", all lowercase, no quotes, no whitespaces, no other characters**, and it must be associated with the commit to be evaluated.

Alternatively, you may insert the tag from GitHub's web interface (In section 'Releases' 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 in an empty 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 ci; npm run dev)
(cd server ; npm ci; nodemon index.js)
```

Make sure that all the needed packages are downloaded by the `npm ci` 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 (for instance, in an empty Virtual Machine).

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 upper/lowercase characters, especially of `import` and `require()` statements, and of any filename to be loaded by the application (e.g., the database).