PG6300 Exam

The exam should NOT be done in group: each student has to write the project on his/her own. During the exam period, you are not allowed to discuss any part of this exam with any other student or person. The only exception is questions to the lecturer in class and on the discussion forum on Canvas. Failure to comply to these rules will result in an **F** grade and possible further disciplinary actions.

On average, it is expected that each student spends around 40-60 hours on this delivery, i.e., 5-8 working days. Grades will be based on what can be expected from 3rd year students working 40-60 hours on a project. Students can of course work less or more hours. The range "40-60" is just to give a high level indication of what to expect.

The students have 1 month (4 weeks to be precise) to complete the project. During such month, students will be busy also with other exams. This is known, and will be taken into account during the evaluation. There is no strict requirement stating exactly when students must start, and how to distribute how much work they should do during the 4 weeks (e.g., if constant throughout the 4 weeks, or more concentrated in the last week).

The exam consists in building a web application using a Single-Page-Application (SPA) approach (with React and React-Router), using NodeJS (serving static files, REST API and WebSockets). There must be only a *single* NodeJS instance, serving both the frontend (e.g., HTML, CSS and *bundle.js*) and the backend (business logic, REST API and WebSockets). The main goal of the exam is to show the understanding of the different technologies learned in class*. The more technologies you can use and integrate together, the better.*

You must write your project in JavaScript and JSX. You are **not** allowed to use other languages that do transpile to JavaScript (e.g., TypeScript, CoffeeScript and Kotlin) or that do compile to WebAssembly (e.g., C and Rust).

The application topic/theme of the exam will vary every year, but there is a set of requirements that stays the same, regardless of the topic/theme of the application.

The application has to be something new that you write, and not re-using existing projects (e.g., existing open-source projects on *GitHub*). If you plagiarize a whole project (or parts of it), not only you will get an **F**, but you will be subject to further disciplinary actions. You can of course re-use some existing code snippets (eg from *StackOverflow*) to achieve some special functionalities (but recall to write a code comment about it, e.g., a link to the *StackOverflow* page).

Once the deadline for the submission is passed, it is recommended (but it is NOT a requirement), to publish your solution as an open-source project (e.g., LGPL or Apache 2 license) on GitHub. This is particularly important if you want to add such project in your portfolio when applying for jobs. However, wait at least two weeks before doing it, as some students might have deadline extensions due to medical reasons.

You need to have a *readme.md* file (in Markdown notation) in the root folder of your project. To avoid having a too long file, you can have extra "*.md*" files under a "*doc*" folder, linked from the *readme.md* file.

In the documentation, you need to explain what your project does, how it is structured, how you implemented it and which different technologies you did choose to use. Think about it like a "pitch sale" in which you want to show a potential employer what you have learned in this course. This will be particularly important for when you apply for jobs once done with your degree. Furthermore, in the *readme.md* you also MUST have the following:

- If you deploy your system on a cloud provider, then give links to where you have deployed it.
- Any instruction on how to run your application.
- Any expected tool that should be run with Docker before your application can be started.
- If you have special login for users (eg, an admin), write down login/password, so it can be used. If you do not want to write it in the documentation, just provide a separated file in your delivered zip file.
- State which parts of the requirements (discussed later) you have not done.

The marking will be strongly influenced by the *quality* and *quantity* of features you can implement. For **B/A** grades, the more features you implement in your project, the better.

Note about the evaluation: when an examiner will evaluate your project, s/he will run it and manually test it. Bugs and crashes will **negatively** impact your grade.

For the deliverable, you need to **zip** (zip, not rar or tar.gz files) all of your source files. Once unzipped, an examiner should be able to build it with NPM. You can assume that an examiner has installed NPM, NodeJS and Docker. It is best if your frontend (i.e., the *bundle.js* file) is built directly with WebPack/Babel. However, using *create-react-app* is acceptable.

Your application should be started with a command like "*npm run dev*". The web app MUST be accessible at "*http://localhost:8080/*". An examiner is not supposed to install databases or other tools to make your app running. However, if you require such external tools, you can use Docker, and specify the exact command an examiner should run (best if all automated inside a *script* in *package.json*). Note: there is no requirement to use a real database. You can "fake" a database by using in memory objects (as seen in class).

This is not a course on *Web Design*, so it is not critical to have nice looking pages, or with very good UX. However, a minimal amount of CSS is expected (although no strong requirement on it) to avoid having pages that look horrible.

During the exam period, you will need to check often the discussion forum on Canvas. There is not going to be any major change to this document, but clarifications might be posted there. Clarifications made on Canvas will be binding for the evaluation and grading of your exam. If you have questions, do NOT send private emails to the lecturer, but rather post on such discussion forum.

Throughout the course, once the 1-month exam starts, in the 2 hours after the 2-hour lectures, students are allowed to discuss and show their working prototype to the lecturer. It goes without saying that

students that start early with implementing their systems will be at an advantage compared to the ones that wait until the last moment before starting.

Students are allowed to re-use and adapt any code from the main repository of the course: https://github.com/arcuri82/pg6300

However, every time a file is reused, you must have comments in the code stating that you did not write such file and/or that you extended it. For an external examiner it must be clear when s/he is looking at your original code, or code copied/adapted from the course.

Easy ways to get a straight **F**:

- Submit your delivery as a rar file instead of a zip.
- Submit a far too large zip file. Ideally it should be less than 10MB, unless you have (and document) very good reasons for a larger file (e.g., if you have a lot of images). The reason is that your delivery might be needed to be sent by email to external examiners. Zipping the content of the "*node_modules*" folders is absolutely forbidden (so far the record is from a student that thought sending a 214MB zip file with all compiled files and dependencies was a good idea…).

Easy ways to get your grade **strongly** reduced (but not necessarily an **F**):

- If you do not provide a "*readme.md*" at all.
- Skip/miss any of the instructions in this document.
- Home page is not accessible at: *http://localhost:8080/*

Necessary but **not sufficient** requirement to get at least an **E** mark is:

- Write a home page with React.
- At least 2 other React pages that can be accessed via React-Router.
- At least one page should have some "state", whose change should be triggerable from the GUI (i.e., there should be some actions for which a React component should be re-rendered and produce different HTML).

Necessary but **not sufficient** requirements to get at least a **D**:

- Create a RESTful API handling at least one GET and one POST, using JSON.
- The frontend must use such API.

Necessary but **not sufficient** requirements to get at least a **C**:

- Add WebSockets to your app, with the frontend using it.

Necessary but **not sufficient** requirements to get at least a **B**:

- You need to handle authentication/authorization, e.g., session-based via cookies.

Nice to have (but **not necessary**) for **A** grades:

- Deployment to a cloud provider (e.g., *Heroku*). (Note: this has not been shown in class)
- Add test cases (e.g., run with *Jest*), and measure coverage. (Note: in class we have not seen how to handle testing of WebSockets)
- Use *Redux* to handle the state in the frontend. (Note: this has not been shown in class)
- Connect to a real database started with Docker. (Note: this has not been shown in class).

Application Topic

The application topic for this exam is about an online, multi-player quiz game. A registered user (let's call him/her X) can start a new match and wait. Every time a new user wants to start a game, s/he will join the game of X. Once X sees that there are enough other players (at least 1) that want to play, s/he will actually start the match. No other player can join that specific match once started (they will start their own new match).

In a match, there are going to be N quizzes (e.g., N=10) on some topic (each time at random). Each quiz will have up to M answers (e.g., M=4), where only 1 is correct. Players get points based on the correct answers and how long they take to answer (the quicker, the more points). At the end of the N quizzes, players should be ranked, and the one with most points will be declared as the winner.

Add any extra feature relevant to such type of system. This is going to be very important for **B/A** grades.