PG6301 – Exam – November 2022

Exam concept

The exam for PG6301 is a practical test where you should demonstrate your mastery of the technologies and concepts that have been thought in the course: React, Express, Azure, Mongodb, Jest, and Web sockets.

The fact that this is a practical exam means that you are evaluated on the functionality of the running code that you deliver, rather than your theoretical mastery of the subjects.

Please read the document in detail.
NOTE: the task specific requirements found below are also compulsory.
NOTE: you are allowed to use code you wrote, code that you make specific reference to, and code from the git repositories associated with the course, as relevant.

Suggestion - Test your application

I suggest that check your application on Azure and Github. Use another machine if possible. When you evaluate your application you can do as follows:

1. Go to the Github page of the solution
2. Check the Github Action actions build and verify that you can see the coverage
3. Check that Github links to the deployed app on Azure
4. Verify that you can log in to the app
5. Verify that you can add data
6. Verify that you can see the data you added

Reusing code, especially `package.json`-files, `index.html` and `index.jsx`, login code, and `server.js` is allowed (and indeed a good idea).

Checklist of technologies you should include in your application

* [ ] Some form of Login and access control
* [ ] Jest tests
  * [ ] Snapshot tests
  * [ ] Simulate + jest.fn
  * [ ] Supertest
* [ ] Github Actions with coverage report
* [ ] Deployment to cloud (in this case, Azure)
* [ ] Mongodb
* [ ] Navigating in the application using React Router (remember Express Middleware)
* [ ] Reading data from the server (remember error handling)
* [ ] Writing data to the server
* [ ] Websockets

Practicalities

* The exam is **individual**. During the exam period, you are not allowed to discuss any part of this exam with any other student or person, not even the lecturer (i.e., do not ask questions or clarification on the exam). In case of ambiguities in these instructions, do your best effort to address them (and possibly explain your decisions in the readme file).
* Structure and formatting:
    * Your project should have a README.md file, that contains information about how to start/run the code, what tasks have been accomplished, and discusses what assumptions were made.
    * Your code should clearly separate the server from client
    * While you are free to use libraries you find useful you **are not** allowed to replace the libraries we discussed in class. So you **cannot** replace React with other libraries, MongoDB with other databases, or go for other backend solutions than Express.

Submission contents:
    * PDF file – a pdf version of the README.md file described above
    * .zip archive – you can obtain this from the github repo. You can (and it is recommended that you do) use the repo associated with the github classroom: https://classroom.github.com/a/qa5zyqOy
Otherwise, you should also provide a link to the github repository you have used, where continuous integration is setup (see requirement **R2**).

Grading

Grading is from A to F. In order to get a grade, you must complete all the requirements associated with that grade, as well as all the requirements for lower grades. For example, to get a C, a submission must fulfil all the requirements listed under **R3** and topic specific requirements under **T3** along with all the previous requirements (so, **R2**, **R1**, **T2**, and **T1** as well).

Below you will find the general requirements. In addition to these, a list of topic or application specific requirements has also been added.

* **R1** Requirements **necessary**, but not **sufficient**, for an **E**
    * Write a homepage with React
    * Have at least 2 other React pages that can be accessed via React-Router
    * At least one page should have some "state", with a change that should be triggered from the interface.
    * From each page, be able to navigate back (either to previous page or to homepage) without using the browser "Back" button.

* **R2** Requirements **necessary**, but not **sufficient**, for a **D**
    * Create a RESTful API that handles at least one GET, one POST, one PUT, and one DELETE calls and uses JSON for data transfer.
    * The frontend must use that RESTful API (for example, using fetch).
    * All endpoints must be listed in README.md
    * The solution should use continuous integration (in this case GitHub actions). Your code should be uploaded onto a github repository, and on every push to the master branch, the CI script should run the tests associated with your project.

* **R3** Requirements **necessary**, but not **sufficient**, for an **C**
   * handle authentication/authorization, session-based via cookies
   * Frontend should have a login page (Register will depend on the topic)
   * A logged-in user should get a personalized welcome message
   * On every page, there should be an option to logout

* **R4** Requirements **necessary**, but not **sufficient**, for an **B**
   * Each REST endpoint MUST handle authentication (401), and possibly authorization (403) checks. If an endpoint is supposed to be "open" to everyone, explicitly add a code-comment for it in its Express handler.
   * Create a test class called security-test.js, where each endpoint is tested for when it returns 401 and 403 (if applicable, i.e., if they can return such codes).
   * In the "readme.md" file, where you list the endpoints (recall R2), for each endpoint list the security tests written for it.

* **R5** Requirements **necessary**, but not **sufficient**, for an **A**
   * Your solution should include continuous deployment to Azure.
   * In the eventuality of you finishing all of the above requirements, and only then, if you have extra time left you should add new functionalities/features to your project.
     Those extra functionalities need to be briefly discussed/listed in the "readme.md" file (e.g., as bullet points). Note: in the marking, examiners will ignore new functionalities that are not listed in the readme document. What type of functionalities to add is completely up to you.

Testing

Your submission should have tests for both frontend and backend. You should measure and report code coverage, and there are requirements on coverage for each grade. The tests should be written with Jest.

The requirements for code coverage are as follows:
   * For an E grade – minimum of 50%
   * For an D grade – minimum of 60%
   * For an C grade – minimum of 70%
   * For an B grade and above – minimum of 80%

Remember, when measuring and reporting test coverage, that some files should be excluded (coverage files, some generated files, etc.)

Links to github repositories for this course:

L1	- https://github.com/pg6301-fall2022/livecode_l1
L2	- https://github.com/pg6301-fall2022/l2_livecode
L3	- https://github.com/pg6301-fall2022/l3_livecode
L4	- https://github.com/pg6301-fall2022/l4_offline
	- https://github.com/pg6301-fall2022/l4_livecode
L5	- https://github.com/pg6301-fall2022/l5_precoded
	- https://github.com/pg6301-fall2022/l5_livecode
L6	- https://github.com/pg6301-fall2022/l6_offline

	- https://github.com/pg6301-fall2022/l6_livecode
L7	- https://github.com/pg6301-fall2022/l7_offline
L8 - Start	- https://github.com/pg6301-fall2022/l8_offline
- Offline	- https://github.com/pg6301-fall2022/l8_offline/tree/offline2
- Livecode	- https://github.com/pg6301-fall2022/l8_offline/tree/livecode
L9 - Start	- https://github.com/pg6301-fall2022/l8_offline/tree/l9_startpoint
- Offline	- https://github.com/pg6301-fall2022/l8_offline/tree/l9_offline
- Livecode	- https://github.com/pg6301-fall2022/l8_offline/tree/l9_livecode
L10	- https://github.com/pg6301-fall2022/l10_offline
L11	- https://github.com/pg6301-fall2022/l8_offline/tree/l11_offline

Task and Task specific requirements

The task specific requirements (marked below with T) are compulsory. In order to get a grade, you must complete both the associated general requirements **AND** the task specific requirements.

Task

You are to make a website and support server for a catering business. The idea is that the business organises and prepares food for events.

They have a catalogue of possible dishes (that is accessible online by potential customers) that show all the dishes that they can provide, along with any relevant information (for example, information relating to allergies, suitability for vegetarian/vegan diets, and ingredients).

A (potential) customer can access the website, show all available dishes, sort/filter them by their preference, and place an order for some combination of dishes, for a specified date/place. A customer can only place an order after making an account.

An admin (or some employee of the company) can add or remove dishes (but, obviously, only while successfully logged in).

Task specific requirements

* **T1** Task requirements **necessary**, but not **sufficient**, for an **E**
   * A visitor to the page should be able to see the menu of potential dishes being offered
   * When the application starts in developer mode, you must have some basic test data, representing a valid menu. If you cannot setup the REST API (requirement for grade D, see requirement **R2**), then hardcode a menu in the frontend.

* **T2** Task requirements **necessary**, but not **sufficient**, for an **D**
   * A customer should be able to create a customer account, but there should be no registration page for admin users (those are for employees, can be assumed to be added via some other mechanism). Your database should include some predefined admin accounts for testing purposes.

* **T3** Task requirements **necessary**, but not **sufficient**, for an **C**
   * An admin should be able to log in, add and remove items to the menu, edit existing menu items (for example, update the ingredients of a menu item).

* **T4** Task requirements **necessary**, but not **sufficient**, for an **B**
   * The server should have separate API paths for customers getting data and admins adding/removing items from the menu. Checks to ensure that only admins can add/remove items from the menu should be enforced.

* **T5** Task requirements **necessary**, but not **sufficient**, for an **A**
   * A chat system (based on WebSockets) should be provided, where a customer can discuss further details with one of the employees in real time.
   * A customer can create an order specifying a time/place where they want a delivery, and how many portions of each dish they select. This order can be placed if the customer is logged

in, and will be stored in a separate database (separate table/collection, not different technology).

You are free (and even encouraged) to add any extra features for this type of system that you think is relevant (once you are done with the requirements above). This will help greatly with getting A/B grades.

Make sure to clearly state in the Readme.md file which requirements you have completed, and which (if any) extra features you have added.

Lykke til and Happy Programming!