Software Developer Course Assessment

Quantitative Assessment Practice #3

Course Name(s): **Full-Stack JavaScript + Database Design**

Current Session: **Winter 2024**

Introduction:

The purpose of this assessment is twofold; first, to help us understand how the class is doing in terms of the course material that we have covered during the previous couple of weeks. The main purpose of this assessment is for us to improve our approach to review and ensure that what we're currently doing is an effective teaching strategy for the students. Completion of this assessment is **mandatory - if you don't submit a solution, it will be marked as incomplete. You must complete a minimum of three of your assigned QAPs per course – otherwise you will be marked as incomplete for that course no matter how good your other grades are.** When you submit a solution, it will be marked using the accompanying assignment rubric.

Again, the goal here is to help you all in the best way that we can, so please do be honest when answering the questions related to how long it took, which resources you used, etc. And please ensure that you do your **own** work – don't just copy off a friend to get it done, earnestly do your best with it. If you can't get it completely working, give us what you have. While it will be graded, the grades will be used as a part of your overall mark. The QAP's will be evaluated less than the team based sprints. Keep in mind the QAP's are also a way for the faculty to see where everybody is, and to know which concepts, if any, we, as a class, may be struggling.

Deadline: Your instructor will determine the deadline for submission for the assessment of your solutions. Please ensure you answer all the questions outlined in the instructions portion of this document as well in your submission.

Marking: In this program core evaluation is marked with one of three possible marks: Incomplete, Pass, Pass Outstanding. For QAPs, though, where incomplete marks are more important for our own information as well as for the information of the student, we wanted to increase the resolution of our grading system. Therefore, QAPs are marked on a scale of 1-5. The details of this marking system are summarized in the table below.

Grade	Meaning
1	*Incomplete.* Student shows severe lack of understanding of the material – solution is heavily incomplete, non-functional, or completely off base of what the assignment was asking for.
2	*Partially Complete.* Students show some understanding of the material. Solution may be non-functional or partially functional, but the approach is correct, albeit with some major bugs or missing features.
3	*Mostly Complete.* Student demonstrates understanding of the major ideas of the assignment. Solution is mostly working, albeit with a few small bugs or significant edge cases which were not considered. Shows a good understanding of the correct approach, and is either nearly a feature-complete solution, or is a feature-complete solution with some bugs.
4	*Complete (Equivalent to: Pass.)* Student shows complete understanding of assigned work and implemented all necessary features. Any bugs that are present are insignificant (for example aesthetic bugs when testing the functionality of code) and do not impact the core functionality in a significant way. All necessary objectives for the assignment are completed, and the student has delivered something roughly equivalent to the canonical solution in terms of features and approach.
5	*Complete with Distinction (Equivalent to: Pass Outstanding)* The student demonstrates a clear mastery of the subject matter tested by the QAP. The solution goes above and beyond in some way, makes improvements on the canonical solution, or otherwise demonstrates the student's mastery of the subject matter in some way. A solution in this category would consider all reasonable edge cases and implement more than the necessary functionality required by the assignment.

The QAP3 covers two courses: as a reminder, this QAP3 covers both the FullStack and Database courses. Therefore, giving you more time to develop a better overall product.

Instructions:

You are allowed to complete the assessment problems below in whatever way you can but please answer the following questions/points as part of your submission:

1. How many hours did it take you to complete this assessment? (Please keep try to keep track of how many hours you have spent working on each individual part of this assessment as best you can - an estimation is fine; we just want a rough idea.)

2. What online resources you have used? (My lectures, YouTube, Stack overflow etc.)

3. Did you need to ask any of your friends in solving the problems. (If yes, please mention name of the friend. They must be amongst your class fellows.)

4. Did you need to ask questions to any of your instructors? If so, how many questions did you ask (or how many help sessions did you require)?

5. Rate (subjectively) the difficulty of each question from your own perspective, and whether you feel confident that you can solve a similar but different problem requiring some of the same techniques in the future now that you've completed this one.

Problem Background:

Understanding the Full-Stack end-to-end and the best practices in their design combined with the ability to build a RESTful API with Node, Express, EJS, and PostgreSQL is a desired technical skill. APIs are everywhere and most software technology companies providing services on the internet have many APIs, and related user interfaces, to get at, and update, their data.

The problem for this QAP3 is to build a simple user interface working with server routes and a RESTful API against a single table for a PostgreSQL database of your own. The important part of this QAP is to have your node.js scripts for **both** the UI and API access the same data access layer which retrieves, creates, updates, and deletes data from your custom database table. If you aspire to only have a **COMPLETE** grade for this QAP I suggest you keep the database layer accessing to a single table AND only implement **either** the simple UI or the API. If you aspire to **COMPLETE with DISTINCTION**, I suggest you implement both the UI and API. Reminder, to have a COMPLETE mark you must implement all five http methods of GET, POST, PUT, PATCH, and DELETE.

NOTE: If you are working on another node.js project that fulfills the requirements of this QAP, ***and you have instructors' permission****, you can submit that project in place of this QAP. The important learning outcomes for this project are; it must be node.js, express, some UI framework, and a database. It must also implement all the http methods of, GET, POST, PUT, PATCH, and DELETE.*

Please work alone on this QAP. Ask for help from classmates, the instructor, the TA, or other resources. Please create an original work. **DO NOT** spend too much time spinning your wheels trying to figure something out, reach out for assistance!!!

Record a demo video of your working QAP3. Demonstrate all the features you have working for both the UI and API. Include this video with your QAP submission.

Complete the following tasks:

1. Create a simple database table in PostgreSQL to have your node.js application work with. (When you create your own database table. Be sure to include the SQL CREATE for the tables with your project submission).
2. Create and execute the SQL SELECT statements in a query window to confirm your understanding of SELECT, INSERT, UPDATE, and DELETE. Save these SQL statements for use with your Data Access Layer.
3. Create a QAP3 project folder to disk for storing all the related project files. Be sure to create folders to organize all your files into well-organized modular code.
4. Save all SQL queries to disk file(s) into a folder dedicated to SQL.
5. Using NPM packages to build a simple RESTful API and web application in Node.js using the EJS view engine, the Express middleware, and the PostgreSQL pg packages. Note: The UI and the REST API needs to GET (SELECT), POST (INSERT), PUT, PATCH (UPDATE), and DELETE data from your database. Required packages include pg, express, method-override, and ejs. Include other npm packages if you find them useful for your solution.
6. Create all node.js and ejs files like the example code created during the lectures. One file should implement the routes for the API, another the data access layer (DAL) accessing the database table you have created, and other ejs files providing the user interface.
7. Execute your Node.js application and test the application via a localhost http request. Iterate your development until your website UI and API service successfully displays the database query results and allows for all CRUD operations.
8. Be sure to save files to the respective QAP3 project folders.
9. Create a video demonstrating all the features of your QAP3 solution.

Approach:

I suggest you work on this QAP in a few phases.

Phase 1: design and create the database table you will be working with and how all the CRUD actions can be performed with the HTTP methods (GET, POST, PUT, PATCH, and DELETE). Be sure to execute the required SQL in a query window for all the actions. Save these SQL statements to disk files in an SQL directory of your project.

Phase 2: create a GitHub repository for your project. Initialize the project as you see best and clone the project for use with VSCode. Create branches for your project for each major development phase as you work through the QAP. Be sure to merge and commit your code with each working milestone of the project. Start new branches for each new feature set. As an example, get all the GET features working as a milestone / single branch. Test these, then merge and commit. Iterate.

Phase 3: using npm start a new project, install the required npm modules, and create a project folder structure that reflects a well-designed full-stack application.

Phase 4: write descriptive test cases to provide complete coverage for all your application features. These should be considered **examples** of single sentences describing each test scenario.

1. As a Martha's Good Eats customer I can see a web page listing all the [insert your table name here] from the database.
2. As a Martha's Good Eats business partner I can fetch all the [insert your table name here] from the database from a REST API. The records will be returned as json data.
3. As a Martha's Good Eats staff member I can see a web page that will allow me to add a new item to the [insert your table name here] of the database.
4. Continue with remaining test scenarios.
5. *Note: Save these test cases to disk files in a tests directory of your project.*

Phase 5: Build the index.js file which implements the express code to serve up a web server. At this time, in your routes folder, also create the js file(s) which define the required route(s).

Phase 6: in your routes file(s) create mock data arrays which implement the data formatted the same as would be returned from the data access layer. Review the lecture sample code for an example.

Phase 7. Build the html files required to display the data. Test the display of the pure HTML before beginning to embed ejs syntax into the html. Iterate and test the ejs file with the working mock data array.

Phase 8. Build the dal service layer to retrieve data from the identified database table. Replace the mock data array with the dal layer database function call. Be sure to implement promises and async / await. Continue your testing. Review the lecture sample code if required.

Phase 9. Extend the application by building out the RESTful API. Be sure the API modules access the same DAL modules as the UI part of the application.

Phase 10. Test all the features of your application using the test cases written in phase 4.

Phase 11. Iterate.

Deliverables:

1. All the CREATE, SELECT, INSERT, UPDATE, and DELETE statements for the PostgreSQL database table your project works against.
2. The node.js and ejs files used to implement all the Model-View-Controller for the UI, the REST API, and the Data Access Layer (DAL) using the respective SQL.
3. Be sure to include your package.json file
4. **A video demonstrating all the features of your QAP3 solution.**

Project Submission to GitHub:

Return your assignment github link and project demo video to the QAP3 assignments page.

All your project files should be saved to a single working directory (folder) with sub-folders and an easily understood file naming convention. All files should be added to a single Github repository using your personal github library. Include **<u>only</u>** your project files in the github repo. DO NOT include other NPM files or folders as part of your project. Do not include the **node_modules** folder. Your QAP will **<u>not</u>** be considered if it contains any additional files as this can make the project zip file > 30 MB in size!

Note: Do not include the video demo file in your github repo.