Software Developer Course Assessment

Quantitative Assessment Practice #2

Course Name: Full Stack JavaScript (Node.js)

Current Week: 03 June 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.

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.

QAP 2 Overview:

Becoming intimately familiar with the core global objects will continue to be an important part of becoming a node full-stack developer. The main deliverable for this QAP is a **multi-route http server that displays html files. For each route an event will be emitted to log the server activity, this logging will be written to both disk and the console.** In its simplest the QAP would use the following four core modules; http, events, filesystem, and console. Include other core modules and npm packages if you like. **DO NOT** use npm packages that assist with the http routing (like express). The purpose of this QAP is for you to deepen your understanding of the http core object and to understand the concept of routing without using another npm package.

The steps you take to complete each task in this QAP is up to you. And don't feel you have to complete each task by itself… If you feel you can implement the whole **multi-route http server that displays html files and makes use of events** as one task, feel free. I can still use the same rubric to mark it. What has been provided here is **only a suggested approach**. As you know I am a very iterative developer, get one small thing working and then slowly add new features. Test, add features, debug, test, deploy, play, repeat!

Tasks #1: Build a multi-route http server.

Step 1: Review node.js videos, lecture materials, support sessions, teaching assistants, and readings that describe how to use the `http.createServer()` to build a multi-route web server with node.

Step 2: Enter greater than five different urls into the browser as a reminder to what is a route please review the following five urls.

1. `http://localhost:3000/about`
2. `http://localhost:3000/contact`
3. `http://localhost:3000/products`
4. `http://localhost:3000`
5. `http://localhost:3000/subscribe`

Step 3: Use the `request.url` to determine in node what url was entered into the browser.

Step 4: Use a `switch`, or other programming logic statement, with the `request.url` to determine the action to take based on the route requested.

Step 5: Add a `console.log("message goes here")` for each different `case` within the `switch` statement. This is so you know it is working.

Step 6: Confirm all this task #1 node programming can be run, without error, using the terminal.

Step 7: Don't forget to have your http `server.listen()` on the correct port.

Step 8: Once you get this working for one or two routes add a few more. Test, add features, debug, test, deploy, play, repeat!

Step 9: Continue to next task.

Task #2: Read html files from a views folder displayed to the requesting browser.

Step 1: Think about the routes you have created with the previous task. Create and author html files to be displayed for each specific route. Store all these html files into a views subfolder of your node project.

Step 2: Use the filesystem core module to provide the ability to read your html files from disk. Be sure to be reading the html files from the views subfolder.

Reminder: load the filesystem with `const fs = require('fs');`

Step 3: Once you have read the html file data read it can be passed as a parameter into the `response.write()` object of the http server. You need to use the `response.end()` to finish things off and complete the response to the server.

Step 4: Once you get this working for reading one or two html files add a few more. Test, add features, debug, test, deploy, play, repeat!

Reminder: remember to write the `response.writeHead()` before the data and include the text/html type.

Task #3: identify events to capture and write them to the console.

Step 1: identify an event, or more than one, that would make sense to capture in a working web server. Suggestions would be;

1. Capture the common http status codes and write a message to the console.
2. Capture only the warnings and errors and write a message to the console.
3. Every time a specific route was accessed and write a message to the console.
4. For every route that is not the home and write a message to the console.
5. Every time a file was successfully read and write a message to the console.
6. Every time a file is not available and write a message to the console.

Step 2. Provide a written description of the events scenario you are implementing.

Step 3. Instantiate an event emitter

```
// define/extend an EventEmitter class
const EventEmitter = require('events');
class MyEmitter extends EventEmitter {};
// initialize an new emitter object
const myEmitter = new MyEmitter();
```

Step 4: Use the `myEmitter.on()` and `myEmitter.emit()` methods of the emitter object to set the event and fire the event, respectively.

Step 5: Once you get this working for one or two events add a few more. Test, add features, debug, test, deploy, play, repeat!

With Distinction (BONUS) tasks – complete two or more.

User Story 1: As a system administrator I would like to have the system create disk files with logged events so I can review how the system has been behaving through time.

Steps: enhance the event logging to console so it also writes files to disk. Be sure to have good folder and file management. Logging to a single file is not a good idea. Log to daily files. Make sure the system creates a new file every day.

User Story 2. As a user I would like to have a route (web page) that provides useful daily information like news, weather, films, etc.

Steps: Search NPM for packages that provide daily or weekly information. Install the package and display information using response.write().

User Story 3. As a user I would like to have a simple menu displayed on every web page so I can easily access all the different routes.

Steps: Using your html and css skills create a menu system on each page for every route.

Task #4: Using Github for the Project Deliverables:

All your project files should be saved to a single github repository with an easily understood file naming convention. DO NOT include other NPM files or folders as part of your project. Do not include the node_modules folder. Your QAP will not be considered if it contains any additional files as this can make the project zip file > 30 MB in size! Use the .gitignore file to not include unnecessary files into your project.

Use git's ability to branch from the main code trunk and to merge your branch back into the trunk after each new feature has been developed. Developing an understanding of branching and merging your code is an important skill.

QAP Project Submission:

Submit the completed questionnaire with your project submission. The project submission should include the URL to your github repository for this project. This should be submitted to the assignments in the MS-Team.