COMP 2406: Fundamental of Web Applications

Carleton University
Fall 2024 Midterm Exam Solutions

October 29, 2024

Be sure to read all of the instructions below.

There are 17 questions on **2 pages** worth a total of 36 points. Answer all questions in the supplied text file template. Your submission should be titled `comp2406-midterm-username.txt`, replacing username with your MyCarletonOne username. Please do not corrupt the template as we will use scripts to divide up questions amongst graders. **Your submission should pass the supplied validator.**

This test is open book. You may use your notes, course materials, and other online resources. If you use any outside sources during the exam, you **must cite** the sources. Your citation may be informal but should be unambiguous and specific (i.e., if you refered to the textbook, indicate what chapter and page you looked at rather than just citing the textbook). You **may not** collaborate with any others on this exam, and *you may not use any LLM-based AI services.* This exam should represent your own work. **Randomized and selected interviews will be conducted after the exam. If an interview reveals that your exam is not your own work, your work will be forwarded to the Dean for disciplinary action.**

You won't be graded for spelling or grammar, just do your best to make your answers understandable.

Do not share this exam or discuss it with others who have not taken it. Some students will be taking it at other times due to accommodations. Solutions will be released once everyone has finished th exam.

All explanations should be concise and to the point (generally no more than a few sentences, sometimes much less). If you find a question is ambiguous, explain your interpretation and answer the question accordingly.

If you have questions during the exam, please go on the **lecture zoom** or **message Prof. Somayaji via Teams**. (Zoom will only work during normal class times.)

You have 80 minutes. Good luck!

1. (1) How could you make dbdemo2 listen on port 7000?

   A: Change line 331 of dbdemo.js to be "port: 7000".

2. (5) Answer the following questions about the async function `handler()` in dbdemo2:

   (a) (1) At a high level, what does the object `req` contain?

   A: the incoming web request to be processed

(b) (2) What top-level methods and properties of req are used in dbdemo2? For each, specify where it is used and what it contains.

A: Two properties and one method of req are used in dbdemo2:

- **req.method contains the method of the incoming request (GET, POST, etc) and is used in route() and handler()**
- **req.url contains the URL of the incoming request. It is used in handler(), routeGet(), and routePost().**
- **req.formData() contains the body of a POST HTTP request. It is used in addRecord()**

One point for the method & properties, half for where, and half for what it contains.

(c) (2) Does `handler()` have to be asynchronous? Explain briefly.

A: Yes it must be asynchronous because it calls await on fileData and route. The only functions that can use await are async functions. (1 for yes, 1 for the explanation)

3. (2) What are two methods of the `Deno` object that we have used in class? What do those methods do?

   A: I count five Deno methods used in Tutorial code (no new ones are introduced in the assignments). `Deno.readTextFileSync()` **and** `Deno.readTextFile()`**, used in Tutorial 1, reads a text file (synchronously and asynchronously).** `Deno.exit()`**, in T1, exits the currently running process (terminates the program).** `Deno.readFile()`**, used in Tutorials 2, 3, & 4, is used to read a file (that is served by the web server).** `Deno.serve()`**, used in Tutorials 2, 3, & 4, starts a web server.** `addSignalListener()`**, introduced in Tutorial 4, is used to set up a signal handler for a UNIX signal.**

   Note that only two of these need to be listed to get full credit. (half point per method, half point per explanation)

4. (7) Consider the following code:

```
    import { DB } from "https://deno.land/x/sqlite/mod.ts";
    var db = new DB("test.db");
```

(a) (1) What Deno permissions does this code require to run?

A: Needs `--allow-read` **and** `--allow-write`**. (Note that downloading the module doesn't require network permission because it is done by Deno not the app.) (half a point for each permission)**

(b) (2) Will this code ever cause Deno to access the network? Why?

A: Yes, when first run, to download the sqlite module. (1 for yes, 1 for the explanation)

(c) (2) Will this code ever create a file? Explain briefly.

A: Yes, if the sqlite database doesn't exist it will be created. (1 for yes, 1 for the explanation)

(d) (2) After this code is executed, can we be certain that `db` is ready for queries? Why or why not?

A: We cannot be certain the database is ready for queries. If the database did not already exist, it will be created with no data tables. As such, it isn't ready for meaningful queries. (Technically you can do queries on the sqlite built-in tables but those won't have any real information in a freshly created database.) (1 for not ready, 1 for the explanation)

5. (2) How can the `push()` method be used to store a collection of JavaScript objects? Will elements in that collection be ordered or unordered? Explain with code snippets or references to code covered in class.

   A: If we start with an array a, new objects "obj" can be added to it with a.push(obj). Elements in the array will be ordered by when they were pushed, with the first element being in position 0. Tutorial 3's form-demo used exactly this strategy, with the push in addRecord(). (one point for push with an array, one for ordered by when pushed)

6. (3) How can `SQL INSERT` be used to store a collection of JavaScript objects? Will elements in that collection be ordered or unordered? Explain with code snippets or references to code covered in class.

   A: We saw how to use SQL INSERT is used to store a colrlection of JavaScript objects in dbdemo (Tutorial 4) and dbdemo2 (Assignment 2), with the INSERT happening in addRecordDB(), with the fields of the object being individually assigned to specific fields in the inserted database record. Inserted records are ordered by order of insertion, although databases can be accessed in arbitrary order using indexes. (1 giving SQL INSERT syntax or referring to addRecordDB(), 1 for saying ordered by sequence of insertion.)

7. (2) What is the `slice()` method used for in the `MIMEtype()` function used in dbdemo2? What about the `toLowerCase()` method?

A: The slice method is used to extract the filename extension, while toLowerCase makes it lower case because all of the extensions in MIME_TYPES are lower case and case doesn't matter in file extensions (JPG and jpg mean the same thing). (One point for slice explanation, one for toLowerCase())

8. (2) Will a browser always follow the MIME type of a requested document? How do you know?

   A: In Assignment 1 we saw PDFs be rendered as PDFs even when simpleserver2 didn't return the appropriate MIME type (question 2) and content from simpleserver2 being rendered correctly even when everything was given an invalid MIME type (question 3). Based on these answers, we can conclude that a browser will not always follow the supplied MIME type of a requested document. (1 for saying not always, 1 for an explanation)

9. (2) In what JavaScript context is the `${}` syntax used? For what purpose has it generally been used in this class? (Be sure the purpose is specific to the programs in which it is found in this course.)

   A: This syntax is used in template literal strings (demarked like `` `this` ``) to include the output of JavaScript expressions in the string. We've used these primarily with HTML templates, starting with Tutorial 3's form-demo (in the _template functions) and every subsequent program, in order to insert values into the HTML templates. (1 for template literals/strings, 1 for HTML templates)

10. (2) Briefly describe the process for adding a static HTML document to dbdemo2. What must be changed or added, and why?

    A: All you have to do is add a .html file to the /static subdirectory. This is because dbdemo2's fileData() will look in this directory for files to serve if a path isn't defined as an explicit route in routeGet() or routePost(), both called by route(). (1 for adding a file to /static, 1 for the explanation why this works)

11. (4) How could you add a new header `X-Email`, which contains the newly added email address, to the response to `POST /add` in dbdemo2 (and no other responses)? Explain what code should be added, removed, or changed. Please outline your solution; do not specify precise JavaScript code. Your solution should require minimal changes to the program.

    A: This question was meant to refer to HTTP headers, but it is possible to interpret this as adding a tag to the HTML `<header>`.

For a HTTP header, you could add "email" as a property to the response object returned by addRecord(), with a value of obj.email. Then, in handler(), in the headers sub-object of the second argument to the new Response object returned by the function, add "X-Email: r.email" between lines 321 and 322.

For an HTML header, you could add a `<X-Email>` tag with the email address contained within it to the header template_addRecord(). However, the function would have to be changed to include the full header created by template_header with the addition of the `<X-Email>` tag and the value of obj.email.

(Points are awarded based on the clarity, accuracy, and invasiveness of the solution, with brief, clear, and minimally invasive solutions receiving 4 points. Both an HTML header and an HTTP header solution can receive full marks, with some more leniency for interpretation of the HTML header one.)

12. (4) How could you change `GET /list` in dbdemo2 so it lists the id of each record along with its contents? Explain what code should be added, removed, or changed. Please outline your solution; do not specify precise JavaScript code. Your solution should require minimal changes to the program.

    A: For this, we need to change template_listRecords() to add a column with the record id. Fortunately, this function already has the id associated with every record because getAllRecordsDB() retrieves and returns the id in the state object that is received by listRecords() and passed on to template_listRecords().

    To change the output, then, we add an ID header to table header (between lines 147 and 148) and then add the id for each state object as part of the loop outputting the table rows (between lines 172 and 173).

    More precisely, to change the output, we add a line `<th>ID</th>` between lines 147 and 148 (part of `<thead>`). We then add a `row.push(rowMarkup(r.id));` between lines 172 and 173 (the loop that outputs state as table rows).

    An acceptable solution would outline the two changes above in English without precise code, but given the changes are so simple precise code is fine.

    (one point for recognizing id is available in template_listRecords(), one for adding the header, one for adding the row, and one for general coherence of the answer)