MURDOCH UNIVERSITY

ICT375 Advanced Web Programming

2021
(Revised 11 September 2021)

Assignment One

Due Date: See Unit Information page on the LMS.

All Students: This is an individual assignment. It should be completed independently. The assignment will be marked out of 100. Your submission must include a completed assignment cover sheet. An electronic copy of the assignment cover sheet is available on the unit LMS site.

Please note that an important objective of this unit is to develop skills in self-learning and research. The assignment question deliberately includes components that may require independent research on your part to produce an appropriate solution.

Assignment Submission: you must submit your assignment to the LMS and keep an ***identical*** copy under the home directory of your account on the ceto server on or before the submission deadline. Your LMS submission must be a single zipped file (including software components and the report) and should be submitted with the filename using the following naming convention:

unit code-assignment number-your last name-the initial of your first name-your student number. e.g. ICT375-A1-Smith-J-87654321.zip for student John Smith with student number 87654321. Note: zip file only (no WinRar or 7z, etc.).

Late Submission Penalty: Late submissions will attract a penalty of 10% per day (including weekends) of the mark achieved for the assignment. Submissions more than 10 days late will not be assessed.

Extensions: Where circumstances make it impossible to submit the assignment by the due date and time, you must contact the Unit Coordinator (via email) immediately, **and definitely before the submission date and time**. Extensions **will not** be granted if your request is made **after the due date and time**, and the above late penalty will be applied if your submission is late.

An extension will only be granted for legitimate (verifiable) reasons. Outside work

commitments, heavy study load, other assignments being due at the same time, my computer blew up, I lost my USB drive, etc., do not constitute legitimate reasons. All students have multiple assignments due around the same time; you need to effectively manage your time. Also, you should *always* keep backup copies of your work in case something goes wrong with your computer, USB drive, etc.

The Unit Coordinator will decide if an extension should be granted, and if so, the duration of the extension. If an extension is granted a copy of the unit coordinator's response email **must** be included at the beginning of your submitted report, otherwise, the late penalty may be applied.

Overview

The assignment involves the development of a web client and server architecture-based application, where the server is implemented using Node.js. The client will run inside a web browser. All communication between the client and server will be via the HTTP protocol. The application will allow the user to submit student information to the server and retrieve student information from the server.

The tasks required of the client-side and server-side programs are listed in the **Functionality** section below, and your design and implementation must meet the **Technical Requirements**. Marks will be allocated as outlined in that section, AND according to the quality of your solution in achieving the required functionality.

A report must document the design details of your solution and testing strategy and test evidence as outlined in the **Report** section and should be submitted **in PDF format**. The report should be comprehensive and clear, with sufficient test evidence to back up your claim as it constitutes the documentation required for the assignment.

Functionality (85 marks)

The completion of this section will require a good understanding of the material covered in the lectures and tutorials for Topics 1 - 4 and will require research/investigation undertaken by yourself.

The client will run inside a web browser, which by default uses HTTP protocol. The client-side program will use an HTML home page, typically named index.html, that allows the user to request the following two types of services from the server. The page should demonstrate a reasonable effort for presentation. Efforts should be made to make **all web pages** presentable and with some consistency using CSS style sheet(s).

1. **Submit student details**. Upon selection of this option, the user should be

presented with an input form to allow entry of a student's details and the degree they are completing. The user may also upload the student's photo if it is available. The form thus presented should gather the following 7 pieces of information: student id, first name, last name, age, gender, degree, and photo. The information will be sent to your server, where it (except the photo) will be appended to the file `data/students.csv` as comma-separated values. If the user also selects a photo file, the server will save the photo file into the sub-directory `photos/`, using the student id as its filename without changing the photo's extension name (such as `.jpg`, `.png`).

Successive student entries will constitute new records in this file and are to be written on new lines. Client-side validation should ensure that at least the form fields id, first name and last name are not empty. Where a field is non-empty, the value inside is in valid format (eg, the id is a number, and a name consists of letters only). If the photo is uploaded, the link to the photo should also be added to the CSV file for that student.

2. **Search for students**. Upon selection of this option, the user should be presented with a form that allows the user to search for students from the server using a search string**.** The details of matching students will be sent back to the client for display. The search must be case-insensitive and should be applied to all fields of student information except photos (eg, including id, names, degree, etc).

   This requires the server to open the CSV file `data/students.csv` and pick up those lines containing a matching field. Finally, the server should send the matching lines to the client, either in CSV format or in JSON format.

   The client should display the details of those students in a tabular form. The details of the students may include the links to their photos if available. The user can view the photo of a student by clicking the link to the photo, and the photo will be displayed in the same page.

   Your application must meet the following technical requirements:

Technical Requirements

1. As a part of the submission, you must keep a copy of your application on ceto. This copy must be identical to the one submitted to the LMS. The files of your application, including all HTML code, JavaScript code, the external style sheet(s), data and image files, must be placed under the directory `assignment1/` in your web home directory on ceto. Be sure to include the `node_modules` directory for any installed Node.js packages that your solution relies on.

   You may develop the application on your own computer, but it must be thoroughly tested and run successfully on `ceto.murdoch.edu.au` before submission.

In addition, the server should be able to run on any host, not just on the localhost or ceto. It should also serve requests from anywhere, not just from the localhost. You must use the port number assigned to you (see the Unit Information page on the LMS). Do not use port 80 or any other ports.

The server start-up script is `index.js` stored under directory `assignment1/`. The server is started by running the command `node index.js` inside the directory `assignment1/`. The client will run from a web browser.

2. The HTML files must be placed under the sub-directory `html/`. JavaScript and Node.js files must be placed under the sub-directory `js/` (but the server start-up script, usually named `index.js`, must reside in the directory `assignment1/`), stylesheets under the sub-directory `css/`, image files under the sub-directory `images/`. These sub-directories must be located directly under the application directory `assignment1/`. There should also be a `data/` sub-directory for storing the .csv file and a `photos/` sub-directory for storing students' photos. All links to these files must use **relative** pathnames (be sure to check these before submission).

3. After each search (by the user) and the appropriate response (from the server), the client-side should offer the user another choice without reloading. In other words, the user should not need to press the browser back arrow or re-load button after viewing the response to their request. You may implement the client as a Single Page Application.

4. Requests must be sent to the server asynchronously. You may handle the request and response using AJAX, jQuery or Fetch API.

5. The design of your application must be modular. You may consider the modular structure introduced in Topic 4.

6. You should aim at efficient communication between the client and the server. For example, when retrieving the matching students' details, only the raw data, either in CSV format or in JSON format, need to be sent to the client. In addition, finding the matching students should be done on the server side to reduce the amount of data sent to the client.

7. You must validate all HTML code using TotalValidator (XHTML 5), and present evidence that your code meets this standard (evidence can be provided by the inclusion of the TotalValidator output).

8. You must validate user input on the client and provide inline feedback if something is wrong. Do not use alert dialogues to provide user feedback.

9. The client interface should look neat and professional, but no need to be fancy. It is important to provide user feedback. For example, whether the student details are

submitted successfully or not, the user needs to be informed. Again, do not use alert dialogues to provide user feedback.

10. All presentation/formatting instructions must come from an **external** style sheet(s). Do not attempt to specify formatting instructions within HTML tags. You must present evidence that your CSS has been validated using the W3C online validation site.

11. JavaScript (both client- and server-side) must be in an **external** file (ie., a `.js` file), and must be modular in design.

12. Your web pages, scripts, and style sheets must be hand-coded using a text editor. For this assignment, we **will not accept** code that is automatically generated using software tools such as Adobe DreamWeaver, Microsoft Office, etc.

13. Do not use any client-side web framework such as Angular in this assignment. If unsure, please seek clarification from your lecturer. You may use jQuery.

14. Do not use any Node.js web framework such as Express in this assignment. If unsure, please seek clarification from your lecturer.

15. For this assignment, there should be no need to utilize any external modules, other than `node-formidable`. So, unless you believe it is necessary to your solution, please do not install and use any other external modules. Seek clarification from your lecturer if unsure. If you have the permission from your lecturer to use any additional modules, you must document the module/s installed and justify why you believe they were necessary to your solution (see Report section, part 2 below).

16. Please refer to pages 16-17 of the introduction to the unit in Topic 1 about using some else's code.

17. Before you zip up your software application, please copy the `node_modules` directory and your report to directory `assignment1`. Then zip up `assignment1` directory and submit the zip file to the LMS.

Note: You consult the Q&A file for Assignment 1 (ICT375-A1-QandA.txt) on LMS. However, please be aware that the assignment has changed this year, therefore not everything documented in Q&A applies to this assignment. Any questions related to the assignment from students will be answered and posted in this document. The Q&A file will post the most recent additions at the top of the document. Therefore, older posts will be further down in the document. Any questions that are received but have been already addressed in the Q&dA file, will be answered by reference to an earlier post. So please read the Q&A file before sending emails.

Report (15 marks)

The first page of the report document should include the unit code and unit name, the assignment number, your name and student number. The body of the report document requires the inclusion of the sections listed below. The sections may be either numbered or given a descriptive section heading (but must be in the order given below):

1. An introduction providing an overview of the assignment, including any assumptions being made in your solution (assumptions will always be made, but **cannot contradict** the assignment requirements).

2. A full description of each of the **non-core** Node.js modules that your server application employs. N.B. *core modules* are Node.js modules such as http, fs, etc. *Non-core modules* are any other Node.js modules (installed via npm), AND also those you have developed yourself for your solution. Talk about what each function (within a module) does and how it achieves its purpose in your assignment. Provide a realistic justification for the functionality of each of the functions and modules. If you utilize any non-core Node.js modules, other than `node-formidable`, you must list them in this section, explain their usage, and provide justification for using them.

3. A detailed description of the overall design of your solution. That is, how all of the functions and modules work together to provide the required functionality. This description should include both the client-side and server-side parts of the solution. As part of this section, include *appropriate* diagrams to help clarify your design description (eg: UML, structural organization diagram, data flow diagram, state transition diagram, etc.). Your description should reference your diagrams.

4. A description of the data structures that your solution utilizes. Remember, an array is a data structure, as is an object. You **must** explain how the data structure/s were utilized, and a justification for why the data structure/s were utilized in the way they were.

5. A thorough testing strategy AND evidence of such testing. You should test the application as a whole, by demonstrating that each client request is successfully completed (or an appropriate error response, sent by the server, is displayed in the browser). You should also test each URL individually (listed in `index.js` and catered to by the request handlers). This can be done by specifying a URL in the browser (or using '`curl`'; this utility is installed on `ceto.murdoch.edu.au`). You should test for errors in the input URL, with the browser (or `curl`) displaying the appropriate error response sent by the server. Testing via a browser will require the submission of screenshots as evidence of testing.

   All features claimed below must be supported by test evidence. Each feature test should include the purpose of the test, the outcome of the test and evidence that the feature works as required. Of the 15 marks allocated to the

report, 10 of which are allocated to feature testing.

6. A conclusion to honestly summarize what you have achieved against each requirement. You should also indicate (as a sub-section) what you were **unable** to achieve concerning the required Functionality and Technical Requirements. *False claims will cost marks*. You can also highlight any points that you consider demonstrate good design, clever pieces of code, etc.

In your discussion be precise with your terminology, particularly with the distinction between modules and functions, arrays and associative arrays, parameters and arguments, objects and other code components.

Breakdown of marks

- (35 marks) submission of student details
- (25 marks) search for students
- (15 marks) display of student photo
- (10 marks) overall quality of the application
- (15 marks) report

Total: 100 marks