MURDOCH UNIVERSITY

ICT375 Advanced Web Programming

2020

Individual Assignment One

Due Date: See Unit LMS.

All Students: This is an individual assignment and it should be completed independently. The assignment will be marked out of 100, but will constitute 10% of your overall unit mark. 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 in order to produce an appropriate solution.

Assignment Submission: A working version of the assignment must exist on `ceto.murdoch.edu.au` under your home directory. Also, you must submit an electronic copy via LMS. Both must be available/submitted by the due date and time. N.B. the version on `ceto` must be identical to the electronic copy submitted on LMS. 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 compression 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 prior to 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, 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, etc.

The Unit Coordinator will decide if an extension should apply, 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.

Coding Standard and File Organization: The website must meet the following conditions:

1. You must validate all HTML code using TotalValidator (XHTML 5.2), and present evidence that your code meets this standard (evidence can be provided by inclusion of the XHTML5 logo, as was done in ICT286 last year).

2. 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 W3C online validation site (CSS3 standard; include logo for evidence as in point 1).

3. JavaScript (both client- and server-side) must be in an **external** file (ie. a `.js` file), and must be modular in design (i.e. using appropriately designed functions).

4. Your website must be hosted on `ceto`. The files of your website, including all scripts and external style sheet(s), should be placed under the directory `assignment1/` in your home directory. Be sure to include the `node_modules` directory for any installed Node.js packages that your solution relies on.

5. Html files must be placed under the sub-directory ./`html/` (with the possible exception of index.html, which can be placed under the directory `assignment1/`). JavaScript and Node.js files should be placed under the sub-directory ./`js/` (with the possible exception of index.js), 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 ./`tmp/` directory which is used when uploading an image file to the browser.

6. All links to the files in your website must use **relative** pathnames (be sure to check these before submission).

7. 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.

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 client will provide certain options for the user to choose from, and must initiate connection with the server. The server will listen on a port number and must process the client requests, based on the resources requested. It must then respond to the client with the requested resources. If the server is unable to respond with the requested resource, a relevant error message MUST be sent in response. The client will then display the response (the requested resource OR an error message).

Initial connection to the server will be via a web page (i.e. `index.html` or `index.js`). This means that client-side and server-side code will exist on the machine where the server is located (for the assignment this MUST be `ceto.murdoch.edu.au`; 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). You MUST use the port number given to you in lab 1 or via email; **DO NOT** use port 80 OR any other port number.

The tasks required of the client-side and server-side programs are listed in the **Functionality** section below, and **ALL** tasks are required to be attempted. 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, as outlined in the **Report** section below, and should be submitted **in PDF format**. The report should be comprehensive and clear, as it constitutes the documentation requirement for the assignment.

Functionality (70 marks)

The completion of this section will require a good understanding of material covered in the lectures and tutorials for Topics 1 - 4, and will require research/investigation undertaken on your own behalf.

Client tasks: the client-side program will constitute a minority of the marks allocated for the Functionality section (15 of the 70 marks).

The client will run inside a web browser, which by default uses HTTP protocol. The client-side program will use a HTML home page to allow initial connection with the server via a web page, typically named `index.html`. This home page may constitute a form, from which the user may choose the options listed below. Though this form is quite simple (with 3 choices and submit/reset buttons), it should demonstrate a reasonable effort for presentation. Effort should be made to make **all web pages** presentable and with some consistency. That is, CSS should be used, and page components should ***not*** be just positioned in the top left hand corner of the browser.

1. 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 form thus presented should gather the following information: student id, first name, last name, age, gender, degree. These details will be submitted to your server, where they will be written to a file as comma separated values (i.e. a '.csv' file). Successive form entries will constitute new records in this file, and are to be written on new lines. The entered information may be assumed to be sensible. However, validation should ensure that the form fields are non-empty and of the expected datatype (string or numeric).

2. Upon selection of this option, the user should be presented with a form that allows the user to request the details of **all students** who are completing the **same degree**; the user needs to enter the degree for this search. The response should display all details of only those students completing the user nominated degree, and the total number of such students. This requires the server to open the .csv file with the records, from part 1. It will then need to process the file and dynamically generate the response to the client (see server section below). The display should be in tabular form (Hint: jQuery/AJAX may help here).

3. Upon selection of this option, the user should be presented with a form that allows the user to select an image and have it displayed in the browser. The image may be a favourite picture of yourself or your family, animal, vehicle, holiday snap, etc. The image should be stored in an appropriate directory within your application (eg: under `assignment1/images/`). The image should be of manageable size dimensions (re-size it with third party software to decrease file size if necessary).

Note: after each selection (by the user) and appropriate response (from the server), the `index.html` page 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 arrow after viewing the response to their request. Therefore, if you do not use AJAX, you must implement a navigation bar (this method of navigation was covered in ICT286 last year).

Server tasks: the server application code will constitute the majority of the marks allocated for the Functionality section (55 of the 70 marks).

THE SERVER CODE MUST BE MODULAR IN DESIGN.

A DESIGN THAT IS NOT MODULAR WILL ATTRACT **VERY FEW** MARKS.

It is therefore strongly recommended that you develop separate scripts to provide the following functionality for a well designed and structured server application:

- A script to start the application.
- A script to start a server (listening on your assigned port number).
- A script to route (or re-direct) the different client requests to appropriate handlers.
- Request handler script/s to handle, process, and respond to, the different client requests.

The reasons for this approach, and the development process required to achieve it, have been discussed in the lecture notes. You have also followed this development process in tutorials 3 and 4.

The following tasks are considered the core components that your server application should achieve in response to the client requests listed above:

1. When the client requests to enter student details (as in part 1 of client tasks above), the server should serve a form to the client by utilizing a designated request handler. If the pathname is correct, point 2 below should be performed. It the pathname is incorrect, an appropriate response error message must be sent to the client for display.

2. On receipt of the entered details, a designated request handler should process the information in the POST request and write the field values to a file. The six values should be comma separated on one line (you may name the file as you wish, but by convention it should have a '.csv' extension). Subsequent form submissions should be written on new lines in this same file. The file should be stored under a directory within your application (eg: `assignment1/data/`). Any data written to this file must remain 'persistent' if for some reason the server is interrupted or stopped (i.e. once written, the data should still exist in the file after such an event).

3. When the client requests to view **all students** completing the same degree (as in part 2 of client tasks above), a designated request handler should serve a form to the client to allow the user to nominate the degree.

4. On receipt of the nominated degree, a designated request handler should process the information in the POST request. It will need to search the records (in the .csv file from point 2) to return **only** those that match the nominated degree, and calculate the number of such matches. It will need to respond with all details of this information (only those records with the same degree and the number of such records); these should be displayed in table format. AJAX may be useful to return the appropriate data for display on the client-side. If no matches are found in the search, an appropriate message must be returned to the client for display.

5. When the client requests to upload an image (as in part 3 of client tasks above), a

designated request handler should send an upload form to the client.

6. The user can then choose the image, and another request handler should upload the image (this will require the use of the `node-formidable` module). Remember to re-direct the uploading data stream to **./tmp/**, as discussed in lecture 4 (B) pages 7, 8, ,9 and 12, 13, 14.

For this assignment there should be no need to utilize any external modules, other than `node-formidable`. So unless you believe it is absolutely necessary to your solution, please do not install any other external modules on `ceto.murdoch.edu.au`. If you do believe it is necessary, you **must** document the module/s installed and justify why you believe they were necessary to your solution (see Report section, part 2 below).

When you zip up and submit your software application, please include the `node_modules` directory and its sub-directories. This will make it unnecessary for the marker to install external modules.

Remember, the server must use HTTP protocol and thus the Node.js **http** core module. As a core module, this will not be necessary to install; it already available (as are all core modules).

Please refer to page 24 of lecture notes LecturePres-01a, in regard to using some else's code.

You are advised to regularly check the Q&A file for Assignment 1 (ICT375-A1-QandA.txt) on LMS. Any questions related to the assignment from students will be answered and posted into this document. The Q&A file will post 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 (30 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 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** provide an explanation of 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. Obviously, testing via a browser will require submission of screen shots as evidence of testing.

6. A conclusion to honestly summarize what you have achieved. You should also indicate (as a sub-section) what you were **unable** to achieve in relation to the required Functionality. *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.