ELECTRONIC ASSIGNMENT COVERSHEET

Student Number	32674186
Surname	Panicciari
Given name	Mitchell
Email	Mitchell.panicciari@gmail.com

Unit Code	ICT375
Unit name	Advanced Web Programming
Enrollment mode	Internal / external
Date	13/April/2017
Assignment number	1
Assignment name	Individual Assignment One
Tutor	Mark Abernethy

• Submit a copy of this assignment to a plagiarism-checking service. This web-based service may retain a copy of this work for the sole purpose of subsequent plagiarism checking, but has a legal agreement with the University that it will not share or reproduce it in any form. • I have retained a copy of this assignment.
I am aware that in making this declaration, by submitting this document electronically and by using my Murdoch ID and password, it is deemed equivalent to executing this declaration with my written signature.
Optional Comments to Tutor:
Please insert this completed form into the body of the assignment you submit. Follow the instructions in the Unit Guide about how to submit your file(s) and how to name them, so that the Unit Coordinator knows whose work it is.

Introduction

This is the first assignment for ICT375 which involves developing a Web client and a server architecture-based application. The server will be implemented using Node.js all communication between the server and the client will be done using the HTTP protocol. The server and the client will both have different functionality and tasks that they must do.

The tasks that the client must be able to:

- Use HTML / JavaScript to allow for initial connection to the server through a web page
- The web page should offer a form which will have three different options which the user may choose from (these options listed further down)
- The web page should be presentable and formatted to look nice and not just have the options in the top left hand corner of the browser
- First option: The client should allow the user to enter in the details of a student which would include student id, first name, last name, age, gender and the student's degree through a web browser with a form. The details provided should be sent to the server for processing in the way of writing it to a .csv file. Each value should be comma separated and successive form entries should constitute new records in the file. The information entered is assumed to be sensible and the only form of validation is that the fields are not empty.
- Second option: The client should allow the user to enter a degree and return all students who are completing that degree and the count of students found. The user would enter in the degree they wish to search for and the server will respond by showing only the students who are completing the nominated degree in a table and showing total amount of students found via text. This requires the server to open the file with the records made in the first option and process them to show relevant data.
- Third option: The client should allow the user to select an image and have it uploaded and then displayed in the browser. The client should allow the user to choose a picture from anywhere via their browser and have it uploaded.
- After each selection by the user and response by the server index.html should be reloaded to allow for the user to make another selection. This can be done via a button or AJAX and the user should of have to press the back arrow button or type in another URL.

The tasks that the server must be able to:

- The server scripts must be modular and have separate functionality this includes having a script to start the application, a script to start the server, a script to redirect the client requests and a request handler's script to handle, process and respond to the client requests.
- When the client chooses option 1 to enter a student's details, the server should produce a form to the client using a designated request handler. If the pathname is not correct an appropriate response error should be sent to the client.
- When the client enters the details of a student the server will receive the details and have a designated request handle to parse the information and write the filed values to a file. All of the values parsed should be comma separated on one line and put into the .csv file, subsequent form submissions should be put onto a new line in the file.

- When the client chooses option 2 to view all students completing the same degree, a designated request handler will server a form to them client allowing the user to enter in the degree to search for.
- On receipt of the users input a designated request handler should parse the information to be used. The request handler will search the records in the .csv file to find any students who are completing the degree and calculate the number of those records. The server should then generate a table to display all of the students who are completing the degree and the number of records dynamically.
- When the client chooses option 3 the server should have a designated request handler to send an upload form to the client. The user can then choose an image and another request handler should upload the image.

All of the elements link together to make a working application allowing a user to insert students into a .csv file, query the .csv to display all records of students in a nominated degree and finally to upload a picture to have the server and have that picture displayed.

Description of Non-Core Modules

The only non-core module used in my solution for this assignment was “node-formidable” which was recommended by Mark Abernethy. This module is installed using the command “npm install formidable” in the command line on the server that will be serving requests. The node-formidable module extracts all of the low-level details of an incoming form allowing for it to be used.

The reasoning behind using this module is that handling incoming files and forms is about handling POST data and node-formidable has functions available for that which means there is no need to create your owns functions for receiving this data, and it makes sense to use a ready-made solution.

In order to use the module you must import it in your script using “var formidable = require(“formidable”);” then you must create a new incoming form which acts as an interim abstraction of the submitted form an example of this is “var form = new formidable.IncomingForm();” . The incoming form is then used to parse the request object from the server to obtain the fields and files that where submitted.

The function used in my code was the “form.parse(request, [callback])” function, an example from my code would be “form.parse(request, function (err, fields)” . This function parses the incoming node.js request which contains the form data and if the call back function is provided all of the fields and files are collected and passed to that function and in this case would be held in the “fields” variable allowing for it to be used.

The justification for the modules and functions used in my code is that it was required to process form data such as an image being uploaded and form data to be added to a .csv file, node-formidable provides us with functions that streamlines the functionality needed. Making use of ready-made solutions is good programming practice in terms of not overcomplicating code and keeping it readable as long as it is well documented what modules are being used and why.

Solution Design

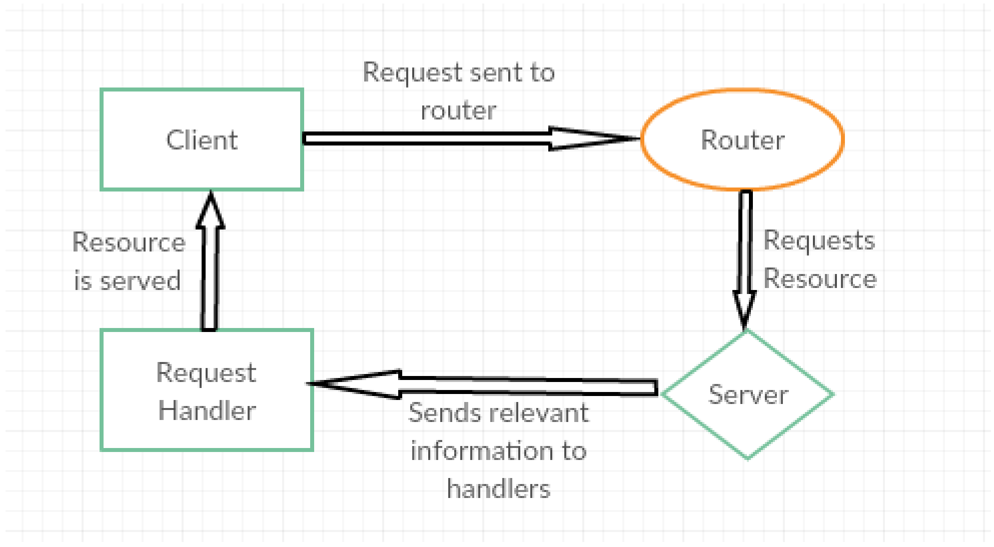
Client Side: Firstly when the /start URL is requested an HTML page is served to the user which contains 3 buttons allowing them to choose one of the three options, these being add student, search degree and to upload and view an image. For every option in my solution I added a button that brings the user back to the main menu in order for them to be able to do another option and does not force them to retype URLS or press the browser back button.

I have added some basic CSS to make my web pages presentable, by putting the elements in the centre with a heading and inside of a div with a different colour background I believe it gives the app a nice look and makes it very simple and obvious what the functions are.

Option 1: When the user chooses option 1 they are presented with a HTML page which contains a form allowing them to enter in a student's information, if they try to enter an empty field they are given an error and the form is not submitted. Once the user has entered in the student's information and press the add student button the form data is passed to the server and the user is shown another HTML page with two options one being to add another student which shows the HTML form for adding students again, and the other is to go back to the main menu.

Option 2: When the user chooses option 2 they are presented with an HTML page which has a form allowing for them to enter a degree and then search it. If the user tries to submit an empty form they will be given an error and the form is not submitted. Once the user enters a degree and presses the submit button the data is sent to the server which then displays a new HTML page to the user with all students who participate in the nominated degree in a table and shows the count of students found.

Option 3: When the user chooses option 3 they are presented with an HTML page allowing them to choose an image and then upload it. Once the user uploads a suitable image for example .png format their image is then displayed to them on a separate HTML page.



Server Side: When the client requests to enter student details the relevant request handler called reqInsertSt is called, which shows a form allowing the user to enter in the students details. When the user submits the student's details the request handler called reqAddStud is used to parse the form data using formidable to a variable which is added to the .csv file.

When the client requests to search a degree the relevant request handler called reqSearchForm is called, which shows a form allowing the user to enter a degree and search for it. When the user submits the degree the relevant request handler called reqSearchStud is used and incorporates formidable to parse the form data to a variable in order to search through the .csv file. The .csv file is read using the file structure module in node.js the whole .csv file is read and each line is then split into an array. The array is then looped through to find if the student in the rows degree matches that of the nominated degree if it is then it is added to an array. The students found are then printed to a new HTML page on a table.

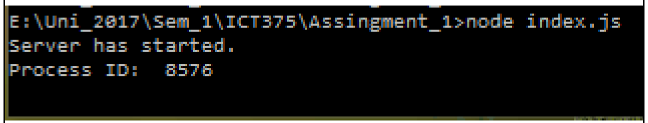
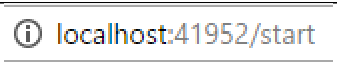
When the client requests to upload an image the relevant request handler reqUpPic is called to serve an HTML form to allow the user to select a picture. When the user clicks choose picture a window displays and allows them to navigate their directories to choose an image. Once they choose an image they can press submit where node formidable is used to upload that image to the server where it is then displayed shortly after being submitted.

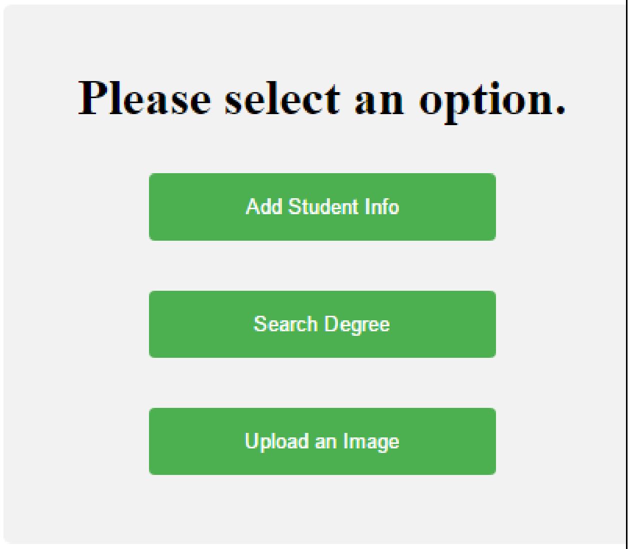
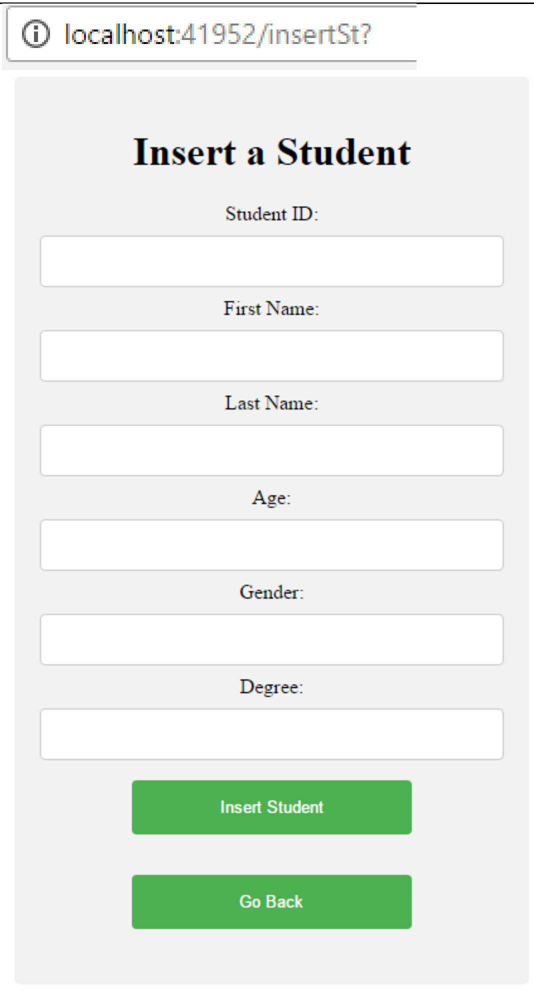
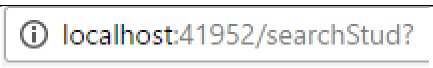
Data Structures Utilized

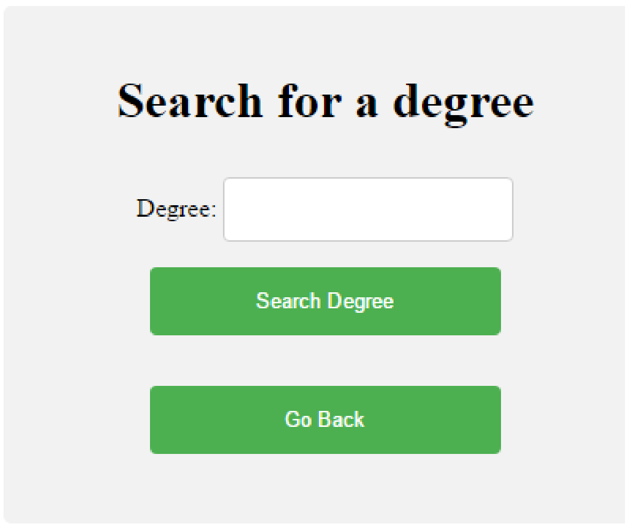
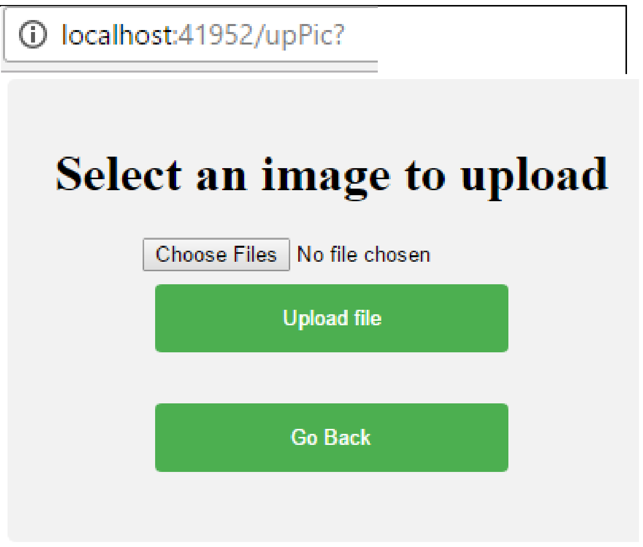
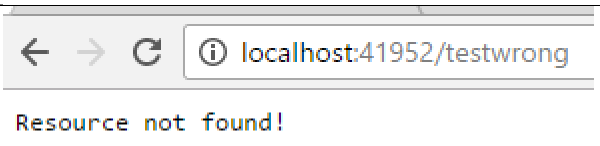
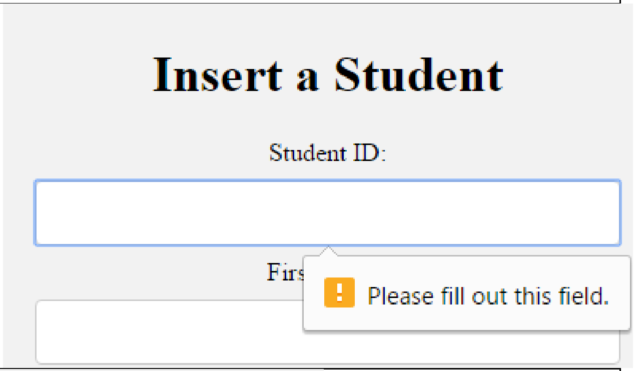
Array: I have used arrays throughout my solution as they allow for storing multiple values in a single variable. This was useful when it came to reading the .csv file and storing the records in that file, firstly the records where in an object which was then parsed to a string and then split and parsed to an array so that they could be used in my solution. I found an array was best for this application as I needed to store multiple students and have them split up in order for me to be able to print them to a table.

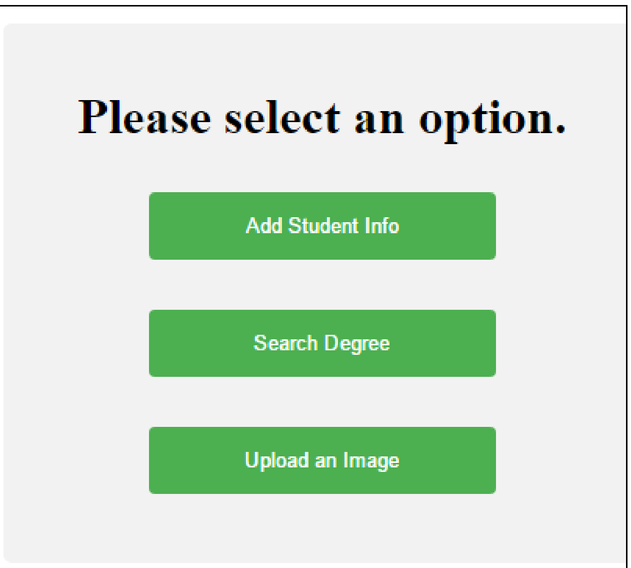
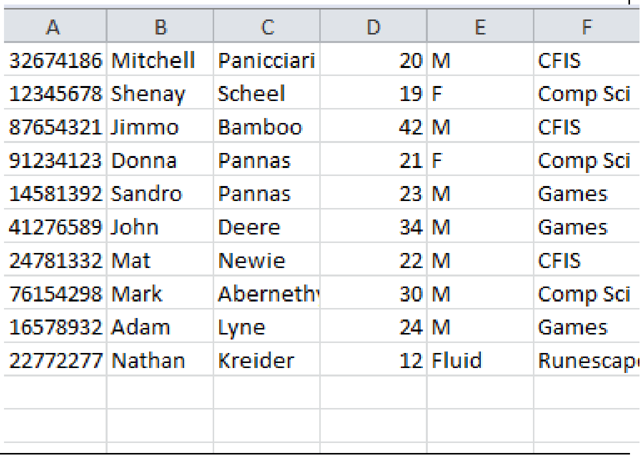
Object: When using formidable the data being parsed comes as an object which can then be used. In my solution in regards to entering a student's details I was able to use array notation to loop through the object containing the data and write it to file.

Test Cases

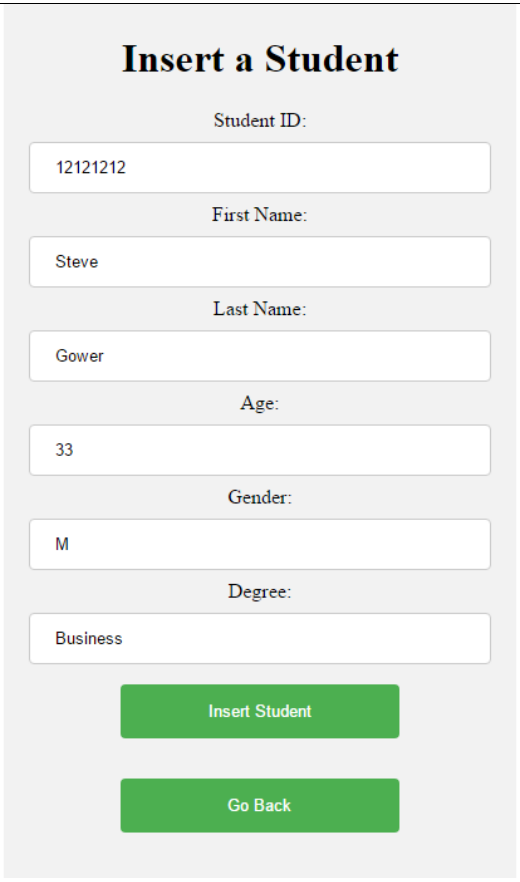
Test	Output	Evidence
Start the server with "node index.js" – Server starts and shows process ID	"Server has started, Process ID: XXXX" Server can be accessed via url	 `E:\Uni_2017\Sem_1\ICT375\Assingment_1>node index.js` `Server has started.` `Process ID:  8576`
URL is entered to show index.html	HTML form output	 localhost:41952/start

		 Please select an option. Add Student Info Search Degree Upload an Image
User chose option 1 "Add Student Info" – Form is served to the client	HTML form served	 localhost:41952/insertSt? Insert a Student Student ID: First Name: Last Name: Age: Gender: Degree: Insert Student Go Back
User chose option 2 "Search Degree" – Form is served to the client	HTML form served	 localhost:41952/searchStud?

		 Search for a degree Degree: Search Degree Go Back
User chose option 3 “Upload an Image” – Form is served to the client	HTML form served	 localhost:41952/upPic? Select an image to upload Choose Files No file chosen Upload file Go Back
User enters handler that doesn’t exist	Error message shown	 localhost:41952/testwrong `Resource not found!`
User tries to submit a form that is empty	Error message is shown form not submitted	 Insert a Student Student ID: Firs Please fill out this field.
User pressed the go back button	User is sent back to the main index.html	localhost:41952/start

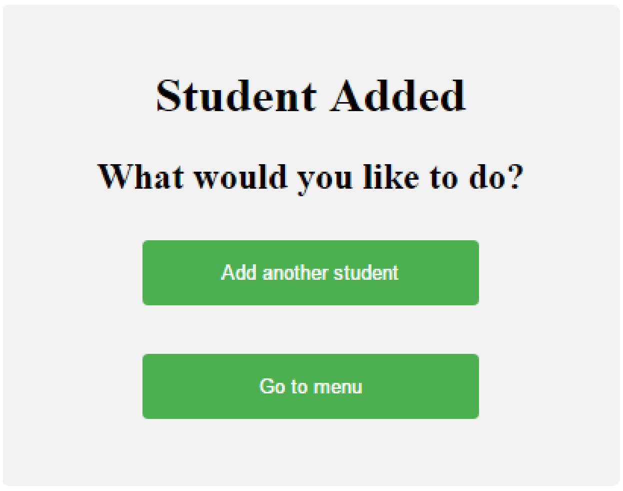
		 Please select an option. Add Student Info Search Degree Upload an Image
User presses insert student button after entering details	Student is inserted into the .csv files, user is shown a form to add another student or back to main menu	.csv before: 

A	B	C	D	E	F
32674186	Mitchell	Panicciari	20	M	CFIS
12345678	Shenay	Scheel	19	F	Comp Sci
87654321	Jimmo	Bamboo	42	M	CFIS
91234123	Donna	Pannas	21	F	Comp Sci
14581392	Sandro	Pannas	23	M	Games
41276589	John	Deere	34	M	Games
24781332	Mat	Newie	22	M	CFIS
76154298	Mark	Abernethy	30	M	Comp Sci
16578932	Adam	Lyne	24	M	Games
22772277	Nathan	Kreider	12	Fluid	Runescap



Insert a Student

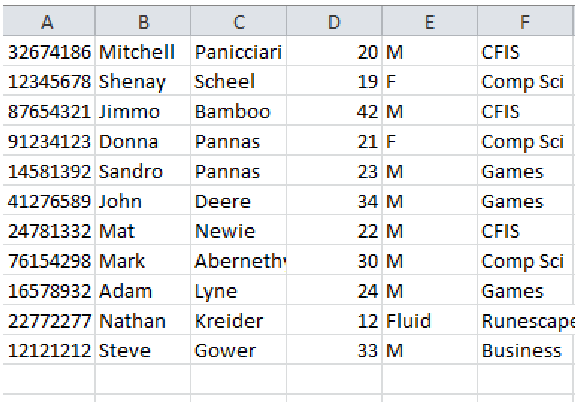
Student ID:	12121212
First Name:	Steve
Last Name:	Gower
Age:	33
Gender:	M
Degree:	Business



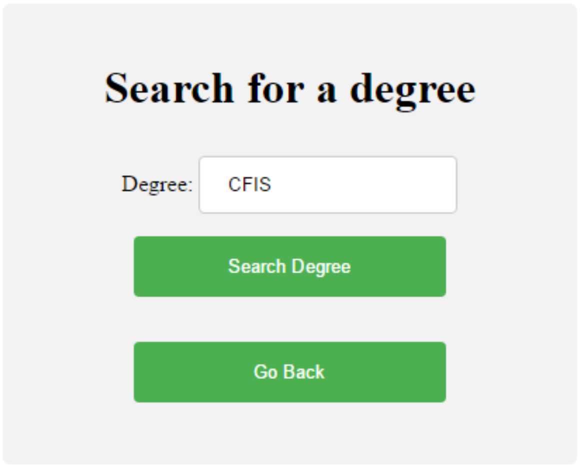
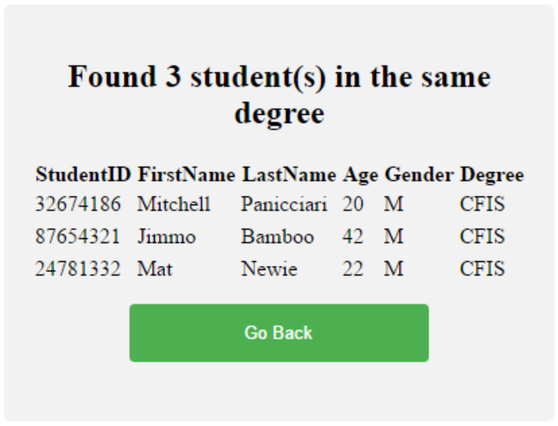
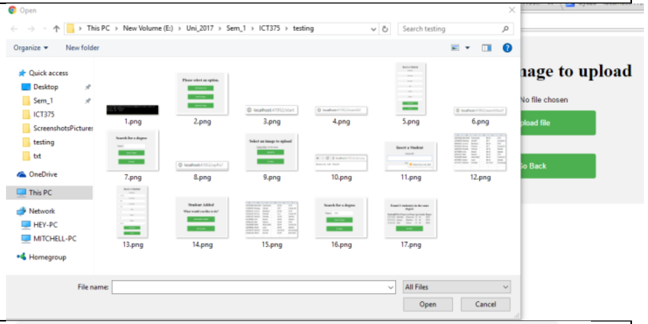
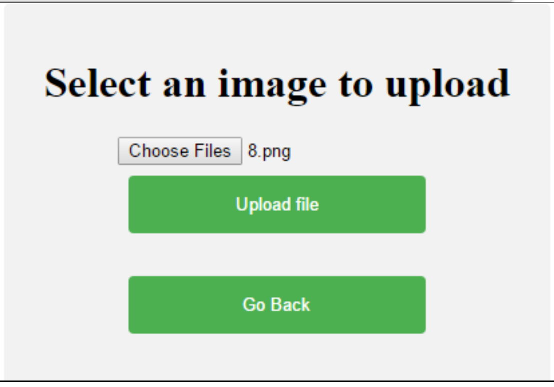
Student Added

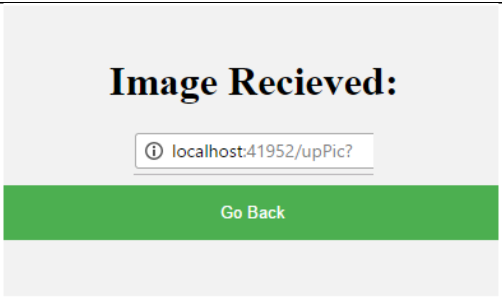
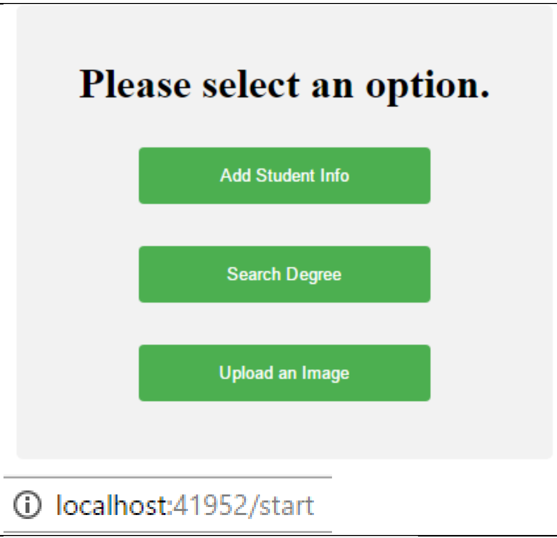
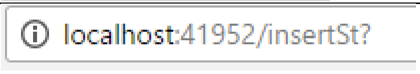
What would you like to do?

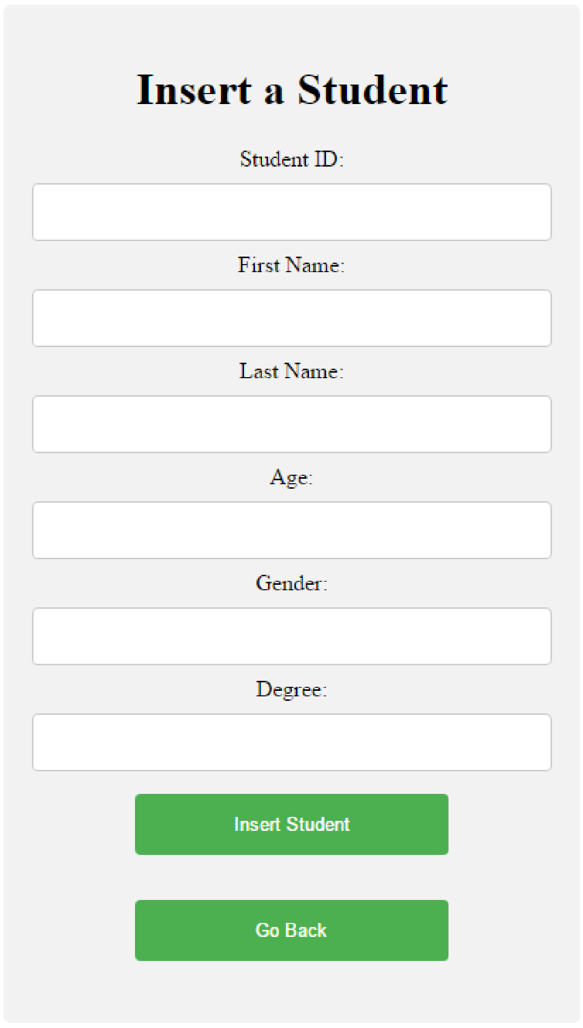
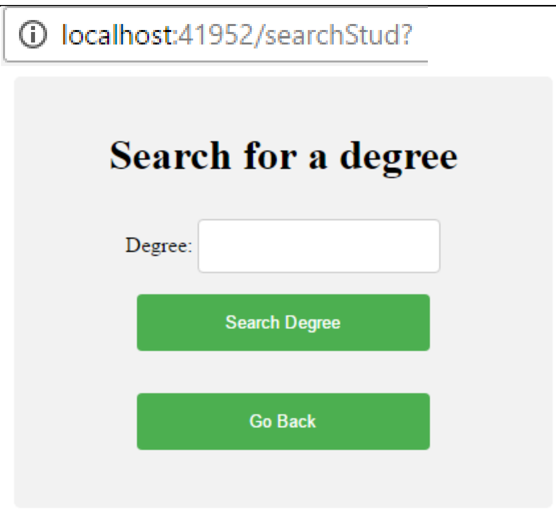
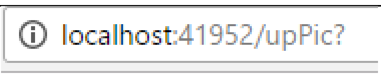
Csv after:

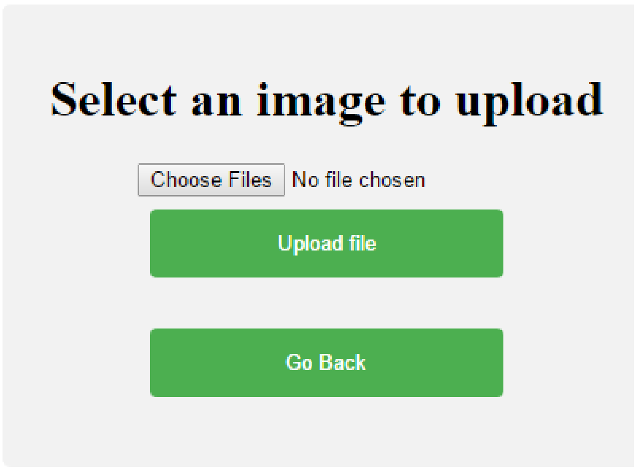


A	B	C	D	E	F
32674186	Mitchell	Panicciari	20	M	CFIS
12345678	Shenay	Scheel	19	F	Comp Sci
87654321	Jimmo	Bamboo	42	M	CFIS
91234123	Donna	Pannas	21	F	Comp Sci
14581392	Sandro	Pannas	23	M	Games
41276589	John	Deere	34	M	Games
24781332	Mat	Newie	22	M	CFIS
76154298	Mark	Abernethy	30	M	Comp Sci
16578932	Adam	Lyne	24	M	Games
22772277	Nathan	Kreider	12	Fluid	Runescape
12121212	Steve	Gower	33	M	Business

User presses “add another student” button after inserting a student	User is served the html form for inserting students into the .csv	
User presses “search degree” after entering a degree	Table is served to the client showing all students found in the degree	 **Search for a degree** Degree: CFIS  **Found 3 student(s) in the same degree** StudentID FirstName LastName Age Gender Degree 32674186 Mitchell Panicciari 20 M CFIS 87654321 Jimmo Bamboo 42 M CFIS 24781332 Mat Newie 22 M CFIS
User presses “choose file” button on image upload	Window pops up allowing user to choose an image	 Open This PC › New Volume (E:) › Uni_2017 › Sem_1 › ICT375 › testing Search testing Organize New folder Quick access Desktop Sem_1 ICT375 ScreenshotsPictures testing txt OneDrive This PC Network HEY-PC MITCHELL-PC Homegroup 1.png 2.png 3.png 4.png 5.png 6.png 7.png 8.png 9.png 10.png 11.png 12.png 13.png 14.png 15.png 16.png 17.png File name: All Files Open Cancel age to upload No file chosen pload file o Back
User chooses a .png image and clicks open	Form shows the name of the image chosen	 **Select an image to upload** Choose Files 8.png

User presses the "upload file" button	User is shown the image	 Image Recieved: localhost:41952/upPic?
URL http://localhost:41952/start is entered	Index.html is shown	 Please select an option. localhost:41952/start
URL http://localhost:41952/insertSt Is entered	Form to add student is shown	 localhost:41952/insertSt?

		 Insert a Student Student ID: First Name: Last Name: Age: Gender: Degree: Insert Student Go Back
URL http://localhost:41952/searchStud entered	Form for searching degree shown	 localhost:41952/searchStud? Search for a degree Degree: Search Degree Go Back
URL http://localhost:41952/upPic is entered	Form for choosing an image is shown	 localhost:41952/upPic?

		 Select an image to upload

Conclusion

I was able to achieve everything asked for in the assignment, the client is able to insert students into a form which is then added into a .csv file, users are then able to query a degree and be shown each student whose degree matches the search, and finally the user is able to upload a picture and be able to view that picture.

In parts of my request handlers which would be serving static HTML pages instead of stuffing the request handler with html being put into a variable and then written with response.write, I made use of the file system or FS module within nodejs allowing me to read a html file which would contain the response for the request handler and then have that html put into a variable and then written to the client using response.write. I believe in doing this it makes my code easier to read and less cluttered with information that is not directly related to the request handler. It greatly reduced the amount of lines my requestHandlers.js file would have had.

In my request handler for searching for students in the .csv and then displaying them in a form, I was having issues with displaying each record on a new table row after the first inserted student. I incorporated the use of the modulus operator to check if 6 values had been added into the table if there had been 6 entered then start a new table row. As my variable holding the students found was an array being split by comma this was a way I was able to get around one student being entered into each row of the tables and then consecutive students being added onto the end of the last student.

Although not requested I was unable to implement a way of searching for a degree and ignoring case. For example if a user tries to search for “cfis” it would still include students who are doing the degree “CFIS”. As of my current solution searching “cfis” won’t show students in “CFIS”.

To conclude I think my solution works as it is intended and does it in a good manner. I would have tried to implement AJAX in some areas but I was unsuccessful in the implementation of it, for example having the students found to be in a certain degree added to an existing element on an html file.