CSC 330

Software Design and Development

Homework Assignment 3

Due: Monday 10/09/2023 (11:59 am)

Introduction

1. In this homework assignment, you will a sendFile function that you will incorporate in all your future work for this course.
2. This homework assignment is written with the assumption that you successfully finished the in-class exercise for reading and serving files.
3. **You may work in pairs**. If you do, make sure that both partners' names are included in the header of the .js file as well as the README file. Only one submission is required.

Grading

Part 1: Answering questions in part 1	2.5
Part 2: Server can read and send files	4
Part 2: sendFile is written as a function that is called from the call back function	4
Part 2: Your code is structured (has at least two functions in addition to the createServer callback function)	3
Part 2: File extension checked and content-type is set correctly	3
Part 3: Answers to questions from lab	1.5
Part 3: Reference and collaboration file submitted	2
Total	20

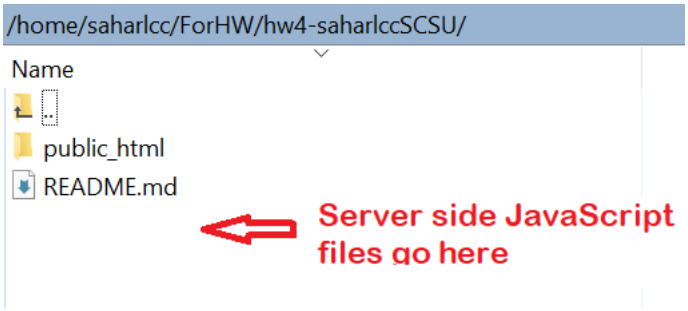
Part 1: Explore the client/server application file structure (2.5 points)

1. Access the homework link on GitHub through this link
   https://classroom.github.com/a/HljTiwz0
   Clone the repo created for you on the VM.

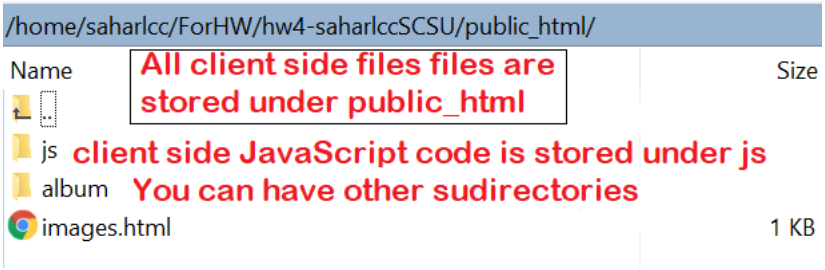
2. Navigate (i.e. cd) to the repo directory and explore it and its subdirectories (using a series of cd and ls commands. *You are a pro now!!*.

   This is a typical web application with an HTTP server directory structure.

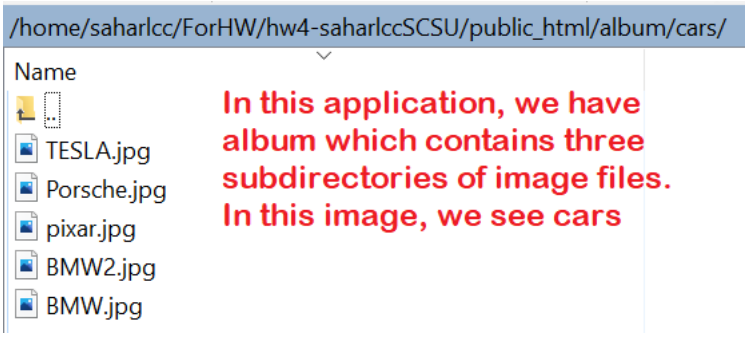
On the top level we store the server side files. This would include the entry point .js file with the callback function. A common name for this file is **app.js** (we have been calling it FirstServer.js). If you have other modules, they would also go here. If the server application is large, you might have subdirectories of modules. You will have only one file, app.js. You will create that file.



Client side files go under the public_html subdirectory



This is an example of a subdirectory of images as part of client application



3. Create a Homework 3 section in the text file you used for answers of the in-class exercise for reading and serving file. Add answers to the following questions.

Q1: What is the common name for the entry point file on the server side application (the file with the callback function)?
Q2: What is the name of the directory that holds the client side application files?
Q3: What is the name of the directory that holds the client side application JavaScript files?
Q4: List the names of the three subdirectories of images included under **album**
Q5: How many image files in total (in all subdirectories) are there in this repo?

Part 2: Build a modular HTTP file server (14 points)

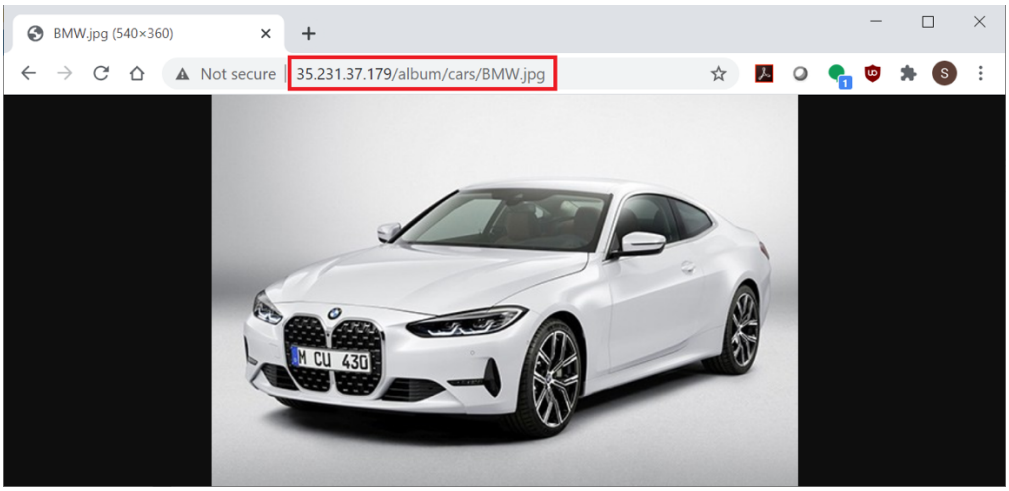
In the in-class exercise, you created an HTP file server. However, it has the following two issues:

i. The server you built in is not modular. If you put the call to readFile in a separate function you can reuse this function in all your future applications.

ii. You have to edit the code and change the content type every time you want load a file of a new type. I am sure you agree that this is a problem.

1) Start by solving issue i, above.
   a. Write a function (**sendFile**) that will be responsible for sending a file (given a file path) to the client. This function will definitely need the file path as a parameter. What other parameter(s) will it need? This function should call **fs.readFile**.
      - As a starting point, make the content type "image/jpg"
      - The function should also handle errors and send error code and message back to the client when needed.
   b. Rewrite the createServer callback function.
      - It should extract the pathname from the URL. Since all client files are under the public_html subdirectory, **it should add "public_html" as a prefix to the pathname in the URL**.
      - It should call the function **sendFile** you created in step a.
   c. Test your modular HTTP image file server.
      - Run the server
      - Connect to the browser and type the path to one of the image files as the pathname in the URL. See the image below for an example. Notice that I included the path, **relative to the public_html** directory. This image is stored in /album/cars/BMW.jpg and this whole path is under **public_html**

        The client will always reference files relative to the root directory of the client, which is public_html. The server should always add the name of the directory of client as a prefix to pathnames of files to be uploaded, as you did in b



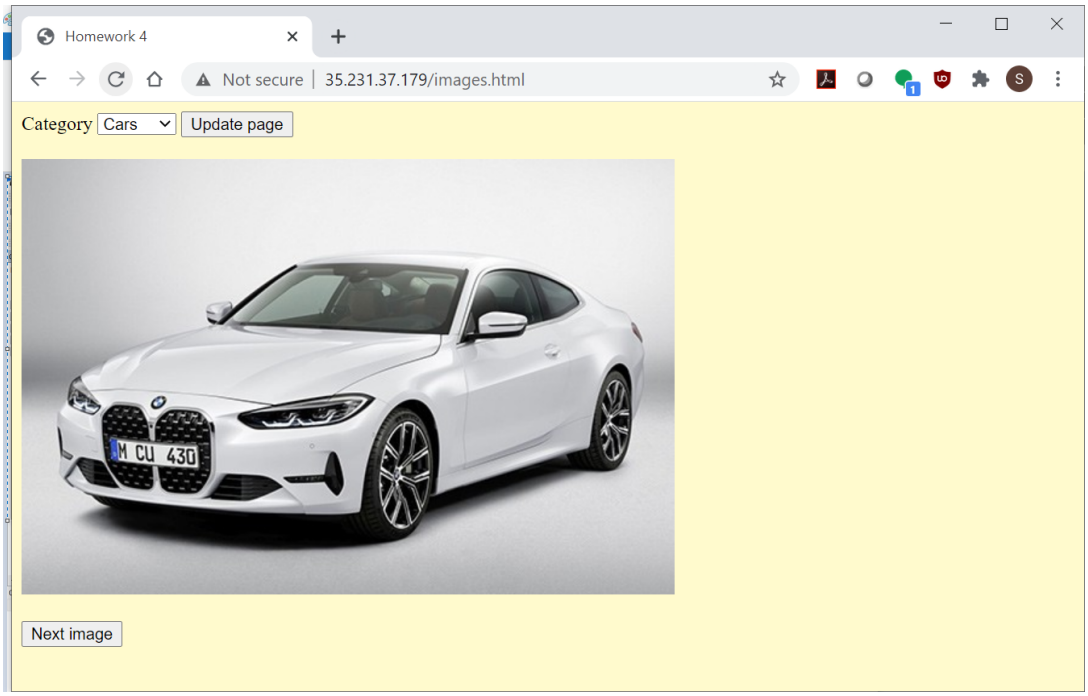
2) It's time to tackle issue ii !!. Make the **sendFile** function general and correctly sets the **content-type** in the header sent to the client, based on the extension of the file being served.

   Note: You can check the file extension using the method **extname** in the Node.js module **path** (https://nodejs.org/api/path.html#path_path_extname_path). ***Don't forget to import/require that module!***

   This page has a table of file extensions and their corresponding **content-type**. You do not need to include all of them, but make sure you cover the ones you will need for this course work (html, js, css, txt, and some image extensions).

   Consider using a switch statement instead of if statements, when implementing the function that will return the content type based on the extension.

3) Your code should be structured, instead of having one big callback function. Here are some possible suggestions to make your call more functional:
   a. Write a function that takes the file path as a parameter and returns the content-type as its return value. This function can be called from **readFile**.
   b. Write a function that is responsible for sending the response back to client. That will eliminate the repetition of calling writeHead(), write(), and end() in your code. What would this function need as input parameters? Will it have a return value?
4) Test your new general file server on the client application given to you in public_html
   a. Add a **console.log(req.url)** in the start of the call back function so you can see the requests your server is receiving
   b. Run the server
   c. Connect to the VM from the browser and include images.html in the URL pathname, as in the image below.



The webpage should load as in the image above

d. Go back to the VM, and check the URLs the server received (you added a statement to print them to the console). As expected, you received a request to load images.html. Notice that you also received requests to load the js/images.js and album/cars/BMW.jpg. These requests were initiated by the browser when it encountered the line

```
<script src="js/images.js"> </script>
```

in images.html, then the line

```
document.getElementById("currentImage").src = "album\\cars\\BMW.jpg"
```

in images.js

Open both files and check out these lines in them.

Your server is sending all these files from the VM on GCP to the browser on your laptop.

If you are able to load the webpage in the image above, then the whole web application should work. Try it out.

e. Commit your changes with a comment “Part 2” and push them to GitHub

Part 3: Additional Files (3.5 points)

1. Create another file with any references or collaborations, or the statements listed in the syllabus for that purpose. Always submit this file in future assignments.
2. Add the references and collaborations file as well as the file with answers to questions from the in-class exercise as well as questions in this assignment to the repo.
3. Commit and push these files.