Homework 1: HTTP and Docker

In this assignment you will build the foundation for the web server you will build throughout this course. You will become familiar with the basics of HTTP and Docker.

Docker

Note: For grading, Docker and docker-compose are required for all objectives in the HW. However, it's recommended that you complete at least 1 objective before adding Docker so you have something you can run to test your Docker deployment.

Setup your web server to deploy with Docker and docker-compose. In the root directory of your submission, include a Dockerfile and a docker-compose.yml file with everything needed to create an image and run your server in a container.

- Map local port 8080 to your app in your docker-compose.yml file

Testing Procedure

The testing procedure for each objective will be followed by the course staff while grading your submission.

1. Download your submission and extract the zip file in a new directory
2. cd into to the directory containing your submission
3. Run the command "docker-compose up"
4. Navigate to http://localhost:8080 in a browser and verify that your web page loads

You should use these steps to ensure that your Dockerfile/docker-compose are set up properly to run your server. Even if you complete all the objectives, any error with your Docker/docker-compose setup may result in a score of 1/3 for all objectives. **Many students have lost significant points by rushing right before the deadline and submitting with the wrong docker configuration**. Don't be one of them.

Objective 1: HTTP Basics

Write a TCP socket server in your chosen language and implement an HTTP server with the following features:

- A path that will welcome users to your brand new site. When a user makes a GET request for the path "/hello", your server should respond with a welcoming message of your choice in a properly formatted HTTP response. The message type should be plain text (eg. content type "text/plain").
    - Example: GET /hello returns "Hello World!"
- A request for the path "/hi" should redirect to "/hello"
    - Add a Content-Length header with a value of 0 to avoid issues with Firefox

- If a request is received for any path that should not serve content, return a 404 response with a message saying that the content was not found. This can be a plain text message.
    - This can be implemented in the else of an if statement. As you add features for more paths, a 404 should be returned if a request did not match any of the implemented paths.

Testing Procedure

1. Start your server using "docker-compose up"
2. Test the path "/hello"
    a. Open a browser, with the browser console open on the network tab, and navigate to http://localhost:8080/hello
    b. Verify that the browser displays a welcoming message
    c. In the browser console, verify that response has a response code of 200, the Content-Type is text/plain, and the Content-Length contains the correct value
3. Test the path "/hi"
    a. Open a browser, with the browser console open on the network tab, and navigate to http://localhost:8080/hi
    b. Verify that the browser displays a welcoming message
    c. In the browser console, verify that 2 HTTP requests were made. The first, for "/hi" has a response code of 301, and the second for "/hello" has a response code of 200
4. Test 404
    a. Open a browser, with the browser console open on the network tab, and navigate to http://localhost:8080/<any_string> where <any_string> is chosen by the tester and is anything that does not match a path used in this assignment, and does not start with "/image" or "/users".
    b. Verify that the browser displays a message indicating that the requested content was not found
    c. In the browser console, verify that the response has a response code of 404, the Content-Type matches the type of content served, and the Content-Length contains the correct value

Objective 2: Host a Static Website (HTML/CSS/JavaScript)

Sample site download

Download and host this sample site, or a site of your own design, such that the site loads when a request is made for the root path "/".

Host index.html, style.css, and functions.js from the sample site or host your own page if you'd prefer (Your page must include, at a minimum, separate files for HTML, CSS, JavaScript, and an image and some of your text must contain non-ASCII characters).

Each file must be sent with the appropriate MIME type. Be sure to include charset=utf-8 to your Content-Type so the non-ASCII characters will be properly rendered in the browser.

You must properly compute the Content-Length of each file. Recall that this length is the number of bytes, not the length of the string. For full credit, this length must be computed programmatically. Points will be lost if the length is hard-coded for each file.

Testing Procedure

1. Start your server using "docker-compose up"
2. Open a browser and navigate to http://localhost:8080/
3. Verify that the browser displays the website
    a. The image does not have to display for this objective
4. Verify that HTML, CSS, and JavaScript were all served through separate HTTP requests
5. Check each HTTP response from your server for the correct MIME type
6. Check each HTTP response for the correct Content-Length
7. Verify in your code that the Content-Length was computed and not hardcoded
8. Verify that the "X-Content-Type-Options: nosniff" header is set on each response

Objective 3: Host Images

Host all of the images in the images directory of the sample site at the path /image/<image_name> where <image_name> is the filename of the requested image including the file extension. If you build your own site you can host any image instead of the default flamingo, but you still must host all the sample site images at this path to simplify testing. You may host more than just the sample site images.

Example: A request for the path "/image/kitten.jpg" should display this adorable image:

Testing Procedure

1. Start your server using "docker-compose up"
2. Open a browser and navigate to http://localhost:8080
    a. Verify that the image of a flamingo displays properly -or- that any image appears if you built your own site

3. Open a browser and navigate to http://localhost:8080/image/<image_name>
    a. <image_name> can be any image provided in the sample site
4. Verify that the browser displays <image_name>
5. Verify that the MIME type of your HTTP response is appropriate
6. Verify that the "X-Content-Type-Options: nosniff" header is set on the image response

Objective 4: RESTful API 1

Create a RESTful API providing CRUD (Technically CRUDL) operations for user accounts containing a username and an email. Add the following paths and functionality to your app. You must create these exact paths since the grading process will be partially automated.

You may assume that all requests are properly formatted for this objective (ie. You don't have to validate the requests).

- **Create**
    - POST /users
        - The body of the request will be a JSON object with email and username fields
        - {"email": "email@example.com", "username": "coolguy123"}
        - When this request is received, create a record in your database and assign it a unique id as an integer. You can/should use an auto-increment feature of the database to assign these ids if you're using a database with this feature
    - Respond with a 201 Created response code
        - The body of your response code will be a JSON object of the record that was created. This will be the same as what the user sent, with the addition of the id.
        - {"id":1, "email": "email@example.com", "username": "coolguy123"}
- **Retrieve - List**
    - GET /users
    - Respond with a 200 OK response code
        - The body of your response will be a JSON array containing all of the created records
        - [{"id":1, "email": "firstuser@example.com", "username": "coolguy123"}, {"id":2, "email": "cse312@example.com", "username": "cse312"}]

Testing Procedure

This testing procedure will only be used if you don't get full credit for objective 5 since the testing procedure for objective 5 tests both objective 4 and 5.

1. Start your server using "docker-compose up -d"
2. Open Postman and make requests for the various API endpoints. For each request, verify that the response code and body follows the API spec detailed above
    a. Create multiple records with POST /users
    b. Retrieve all records with GET /users

3. Restart the server using "docker-compose restart"
4. Access GET /users and verify that the records persist after a server restart

Objective 5: RESTful API 2

Add the following endpoints to your users API.

You may assume that all requests are properly formatted for this objective (ie. You don't have to validate the requests). You may assume that all {id} are well-formed integers, but they might not correlate with a record in your database.

- **Retrieve - Single**
  - GET /users/{id}
    - Where {id} is the id of the record to retrieve
    - /users/1
    - /users/2
  - If the record exists, respond with a 200 OK response code
    - The body of your response will be a JSON object containing the requested record
    - If "/users/2" is requested, return {"id":2, "email": "cse312@example.com", "username": "cse312"}
  - If there is no record for the requested id, or the record has been deleted, return a 404 Not Found
    - You may choose the message of the 404
- **Update**
  - PUT /users/{id}
    - The body of the request will be a JSON object with email and username fields
    - {"email": "email@example.com", "username": "coolguy123"}
    - When this request is received, update the record with id of {id} using the data from the body of the request
  - If the record exists, respond with a 200 OK response code
    - The body of your response will be a JSON object containing the updated record
    - {"id":2, "email": "cse312@example.com", "username": "cse312"}
  - If there is no record for the requested id, or the record has been deleted, return a 404 Not Found
    - You may choose the message of the 404
- **Delete**
  - DELETE /users/{id}
    - There is no body to this request
  - If the record exists, respond with a 204 No Content response code
    - There is no body to your response (We don't want to send deleted records)
  - If there is no record for the requested id, or the record has already been deleted, return a 404 Not Found
    - You may choose the message of the 404

Testing Procedure

This procedure tests both objectives 4 and 5.

1. Start your server using "docker-compose up"
2. Open Postman and make requests for the various API endpoints. For each request, verify that the response code and body follows the API spec detailed above
    a. Create multiple records with POST /users
    b. Retrieve all records with GET /users
    c. Find the id of one of the records and retrieve that record with GET /users/{id}
    d. Update one of the records with PUT /users/{id}, then make another GET/users request to verify the change
    e. Delete one of the records with DELETE /users/{id}, then make another GET/users/{id} request to verify that the record is no longer accessible and returns a 404
    f. Make requests to PUT /users/{id}, GET /users/{id}, and DELETE /users/{id} with invalid ids and verify that the server returns 404s
3. Restart the server using "docker-compose restart"
4. Access GET /users and verify that the records persist after a server restart

Submission

Submit all files for your server to AutoLab in a .**zip** file (A .rar or .tar file is not a .zip file!). Be sure to include:

- A Dockerfile in the root directory
- A docker-compose file in the root directory that exposes your app on port 8080
- All of the static files you need to serve (HTML/CSS/JavaScript/images)

It is **strongly** recommended that you download and test your submission after submitting. To do this, download your zip file into a new directory, unzip your zip file, enter the directory where the files were unzipped, run docker-compose up, then navigate to localhost:8080 in your browser. This simulates exactly what the TAs will do during grading.

If you have any Docker or docker-compose issues during grading, your grade for each objective may be limited to a 1/3.

Grading

Each objective is equally weighted and will be scored on a 0-3 scale as follows:

3	Clearly correct. Following the testing procedure results in all expected behavior
2	Mostly correct, but with some minor issues. Following the testing procedure does not give the exact expected results
1	Clearly incorrect, but an honest attempt was made to complete the objective. Following the testing procedure gives completely incorrect results or no results at all. This includes issues running Docker or docker-compose even if the code for the objective is correct
0	No attempt to complete the objective or violation of the assignment (Ex. Using an HTTP library) -or- a security risk was found while testing the objective

The following conversions will be used when translating grades on this 0-3 scale to percentage grades:

3	100%
2	80%
1	60%
0	0%