SE 3316A Web Technologies Lab #3: Due Monday, Nov. 6, 5:00 pm

Deadlines:

1. Owl Submission deadline: Monday, Nov. 6, 5:00 pm.
2. Late submissions on Owl will be accepted until ~~Friday , Nov 10, 5 pm~~ Sunday, Nov 12, 11:55 pm with a 1 point penalty per day. No submission on Owl (Git log + zip), no grade. Email submissions are not accepted.
3. Demonstration deadline: Friday, Nov. 24, 5:30 pm.

Change Log

1. Removed mark for "sustained effort" and distributed it to functionality. Revised late submission dates and penalties. Oct 17, 1:40pm
2. Added penalty for no shows. Oct 30, 10:45am
3. Changed late submission deadline, Nov 10, 3pm

Objectives:

A. Design and implement a ReST API for accessing superhero characters in movies, cartoons and books and all their attributes from provided JSON files (obtained from Kaggle) as well as creating custom lists of superheroes.
B. Create a client (front-end) interface for interacting with the above API (pure HTML/CSS/JavaScript only. External libraries of any type are not allowed on client-side).
C. Use asynchronous operations for showing both static and dynamic data.
D. Using a single web-server/network endpoint for both the web application as well as ReST API.
E. Apply input sanitization techniques.
F. Learn about the basics of using Node.js and Express to create a ReST API.

In this lab, you will implement a server API for accessing information about movie superheroes along with a simple front-end for this API. JSON files containing information about superheroes are provided in the "Resources/Labs" section on Owl. Note: These files should not be included in Git. Accessing the application using a public URL is mandatory. You may use Amazon EC2 or your own server for a public URL. You may use a local installation of Node.js for development purposes.

This is a complex assignment and all aspects cannot be fully specified. There is a lot of flexibility in implementation and UI design. Please think like a user and decide the best functionality that would make it easy to use the application within the constraints specified in the rubric.

Rubric always takes precedence.

Workflow:

1. Create a new private Git repository on Github called "se3316-xxx-lab3" (all lowercase) where "xxx" is your Western email ID (without @uwo.ca part).
2. Clone that repository on your workstation/laptop to create a local working directory.
3. Create two subdirectories called "client" and "server" in the project directory. Ensure all front-end related files (html, css and js) are in "client" directory and all back-end related files are in "server" directory.
4. Add/modify the `.gitignore` file to ignore the two JSON data files.
5. Deploy on the server using Git.
6. Make frequent and meaningful commits and push your project to Github.

7. This is a relatively large lab requiring at least two weeks of steady effort. Structure your coding sessions for gradual progression. See the “Suggested Steps” in the appendix A2.

Submission Instructions:

Please carefully read the instructions and strictly follow them. Your grade depends on it.

1. Use a proper “**`.gitignore`**” file so that only the files that you edit are in your repository.
2. Make frequent commits with an appropriate commit message.
3. Ensure that you understand the principles behind your code.
4. Ensure that Github contains the latest version of your code.
5. Copy the output of command “**`git log`**” and paste that onto the submission page (Assignments section) on Owl.
6. Download your repository as a zip file from Github and submit as an attachment. Please do not zip the folder on your computer as it contains a large amount of extraneous files.
7. Submit your lab to Github Classroom after submitting on Owl (~~Link will be provided after Nov 6~~. https://classroom.github.com/a/lR6AP29K See detailed instructions).
8. Demonstrate your lab (on a public URL) before the demonstration deadline. You may set up the public URL after submission on Owl. As long as the changes only involve deployment issues, it is acceptable.

Penalties will apply for not following the naming convention or any of the submission steps.

Rubric out of 100:

See the requirements for front-end and back-end for marks allocated for implementation.

1. Using external libraries for front-end: -25
2. No input sanitization: -10 for each occurrence.
3. Code management with Git
    a. Less than 10 commits: -20
    b. Commits are not meaningful: up to -10
    c. Commit messages are not meaningful : up to -5
    d. Not using a proper **`.gitignore`** file to ignore files not created/modified by the user: -5
    e. Not using Git pull to deploy code to server: -5
4. Logistics
    a. Repository name not in required format: -5
    b. No zip file or git log on Owl: not graded
    c. Code is not attached as a zip file or it contains content that is not in the Git repository[1] or is missing content that is in the repository: -10
    d. No show for a signed up demo: -20

Preparation - Creating a Basic ReST API with Node.js and Express

Install Node.js on your workstation as well as on AWS server. Practice using “curl” and “Insomnia” to test a web API. A good introductory video tutorial for creating a ReST API: Express.js Tutorial: Build RESTful APIs with Node and Express.

Complete the week 6 tutorial (six parts) on “Web app with Node.js and Express”. For best results, type the code yourself.

[1] If you zip the file on desktop, it may often include hidden files (particularly on Mac) or the .git folder itself.

Be comfortable with the development process (See “About the course” slides) where you are able to develop and test on your workstation, deploy the latest version on AWS server as well as making changes on one environment and updating the other environment via Github.

Be careful with copy+paste since quotation marks may get mangled (“ vs " and ‘ vs ') and cause problems with the interpretation of strings in JavaScript code (or in any programming language).

Read the article RESTful API - Best practices in appendix A1.

Create a web application for interacting with the superhero data

The web application must consist of a front-end implemented using HTML, CSS and JavaScript (no external libraries or frameworks like React or Angular) and a back-end API implemented using Node.js. Both the front-end and the back-end must use a single network endpoint (protocol+host+port). Use JSON as the format for data returned from the back-end. Use of external modules (accessed from a local copy) for Node.js is allowed. However, minimizing dependencies is advised.

The back-end functionality must be provided using nouns (URLs), verbs (HTTP methods) and parameters for following actions (all searches are case insensitive). Return an error if there are no results. [**70 points total**]

1. Get all the superhero information (all fields in **superhero_info.json** file) for a given superhero ID. [**10 points**]
2. Get all the powers for a given superhero ID[**10 points**]
3. Get all available publisher names. [**5 points**]
4. Get the first ***n*** number of matching superhero IDs for a given search pattern matching a given information field [e.g. match(field, pattern, n)]. If ***n*** is not given or the number of matches is less than ***n***, then return all matches. [**10 points**]
5. Create a new list to save a list of superheroes with a given list name. Return an error if name exists. [**5 points**]
6. Save a list of superhero IDs to a given list name. Return an error if the list name does not exist. Replace existing superhero IDs with new values if the list exists. [**10 points**]
7. Get the list of superhero IDs for a given list. [**5 points**]
8. Delete a list of superheroes with a given name. Return an error if the given list doesn’t exist. [**5 points**]
9. Get a list of names, information and powers of all superheroes saved in a given list. [**10 points**]
10. Implement safeguards to prevent malicious attacks on the API. Examples include limits on size and range, preventing injection attacks etc and unintended side effects. Normal operations such as delete a list are not considered malicious.

Follow the best practices of designing a RESTful API when defining the nouns, verbs and parameters. Data may be entered in any language supported by UTF-8 and the web service must preserve the language encoding. You may use a simple file-based storage for saving lists. See the FAQ for details.

The front-end must provide the following functionality. Please feel free to shape the UI as you would like to use a similar app. [**30 points total**]

1. Ability to search superheroes by name, race, publisher or power and display results. [**5 points**]
2. Ability to create any number of favourite lists, retrieve them and display the names, information and powers of all superheroes in the list. [**10 points**]
3. Ability to sort all data (1 and 2 above) by name, race, publisher or power. [**5 points**]
4. Use asynchronous functionality to query the back-end when the user interacts with the front end.
5. Sanitize all user input so that the display must not interpret any HTML or JavaScript that might be typed on to the text area.

6. Allow any language as the list name and display it in the language it is entered. [**5 points**]
7. Use all of the back-end functions using JavaScript code on the front-end. [**5 points**]

Optional reading: **`http://xkcd.com/327/`**

Code will be checked for similarity. Please work independently. Please frequently check the FAQ below for clarifications, tips and tricks.

FAQ

1. Input sanitization

Main idea is to prevent any text input field from being used in injection attacks (HTML injection, SQL injection, JS injection etc). This happens if you store user input and then send that input back to the client to be displayed. E.g. If a user enters malicious JavaScript into a text box and that gets stored and sent back later without any filtering, it allows injecting JavaScript to your front-end.

It is very hard to sanitize user input that applies to all situations and requires a layered approach. First layer is input validation. If a field requires a number, input validation must ensure that only a number is accepted. Next layer is filtering. You may use a third-party library to strip HTML, CSS and JavaScript from user input. Third layer is how data is added to the DOM. Always ensure that you use `createTextNode()` method to show user entered data.

2. Testing ReST API

A good API tester can help debug your code easily. Browsers can only issue GET requests. Possible tools are:

- Curl - command line tool: E.g. curl --request PUT -d "name1=value1&name2=value2"
- Insomnia - Open source app.

3. Back-end storage

You may use any file-based storage mechanisms for the back-end. Examples include packages such as `lowdb`, `node-persist`, `configstore`, `flat-cache`, `conf`, `simple-store`, `key-file-storage`, `node-storage`, etc. However, you may also use a full-fledged database such as PostgreSQL, MySQL or MongoDB. When selecting external packages for Node.js, please exercise caution and minimize dependencies.

Appendix

A1. RESTful API - Best practices

https://hackernoon.com/restful-api-designing-guidelines-the-best-practices-60e1d954e7c9

A2. Data Format

Data is stored in two JSON files. Some files may be large. Suggest using small subsets for initial development if size becomes an issue.

A3. Suggested Steps

Coding targets: Oct. 20 for steps 1-6; Oct. 27 for steps 7-11; Nov. 3 for steps 12-14.

Testing: Test each back-end functionality with curl or Insomniac first. Verify full functionality (including error conditions) before implementing the corresponding front-end interface. Make commits at the end of each testing stage. For each back-end service, think about how it can be abused and implement safeguards.

1. Complete all week 6 activities including the tutorial for creating the REST API for the “parts” database.
2. Read through the entire lab handout. Read it multiple times until you have a clear picture of **what** is needed (not **how**).
3. Write a JavaScript (Node.js) function for reading the JSON files and extracting data for items 2, 3 and 4 of the back-end requirements.
4. Set up a Node.js and Express back-end for implementing item #1: “Get all the superhero information for a given superhero ID” in back-end functionality.
5. Write a simple front-end (HTML+CSS+JavaScript) to test item #1 above and serve it within the same script above by using the functionality of serving a static file in Express (see class notes).
6. Implement the back-end item #2: “Get all the powers for a given superhero ID”.
7. Add front-end functionality to test item #2.
8. Implement back-end items #2 and #3 and combine steps 6-9 as a coherent user interface (UI).
9. Implement the back-end functionality for items 4, 5 and 6.
10. Add front-end functionality for the above step and create a coherent UI.
11. Implement back-end functionality for items 7, 8, and 9.
12. Add front-end functionality for the above step.