Web Application Development

Objectives:

A. Design and implement a ReST API for the Western online timetable search page.
B. Create a client (front-end) interface for interacting with the above API (pure HTML/CSS/JavaScript only. No external libraries).
C. Use asynchronous operations for showing both static and dynamic data.
D. Using a single web-server/network endpoint for both static content 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 a time table query application for Western University along with a simple front-end for user interface. A file containing course and schedule information for 2019 has been attached to email. 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.

Workflow:

1. This is a relatively large lab. Structure your coding sessions for gradual progression. See the "Suggested Steps" in the appendix A3.

Rubric: (out of 100)

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

1. Using external libraries: -25
2. No input sanitization: -10 for each occurrence.
3. Code management with Git
    a. Less than 10 commits: -20
    b. No meaningful commits: up to -10
    c. No meaningful commit messages: up to -5
4. Logistics
    a. Repository name not in required format: -5
    b. No zip file: not graded

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.

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.

Review the data format of the file containing the timetable data in appendix A2.

Create a web application for searching the online timetable for Western

The web application must consist of a front-end implemented using HTML, CSS and JavaScript (no external libraries) 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: [**70 points total**]

1. Get all available subject codes (property name: subject) and descriptions (property name: className). [**15 points**]
2. Get all course codes (property name: catalog_nbr) for a given subject code. Return an error if the subject code doesn't exist. [**10 points**]
3. Get the timetable entry for a given subject code, a course code and an optional course component. Return an error if the subject code or course code doesn't exist. If the course component is not specified, return time table entries for all components. [**10 points**]
4. Create a new schedule (to save a list of courses) with a given schedule name. Return an error if name exists. [**5 points**]
5. Save a list of subject code, course code pairs under a given schedule name. Return an error if the schedule name does not exist. Replace existing subject-code + course-code pairs with new values and create new pairs if it doesn't exist. [**10 points**]
6. Get the list of subject code, course code pairs for a given schedule. [**5 points**]
7. Delete a schedule with a given name. Return an error if the given schedule doesn't exist. [**5 points**]
8. Get a list of schedule names and the number of courses that are saved in each schedule. [**5 points**]
9. Delete all schedules. [**5 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 schedule or delete all schedules 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 schedules. 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. A minimal user interface that uses all back-end functions. [**5 points**]
2. Ability to search the time table by subject, subject+course or subject+course+component and display results. [**10 points**]
3. Ability to create course lists (schedule), retrieve them and display the time table for those courses. [**10 points**]
4. Indicate different components of a course by different colours. [**2 points**]
5. Use asynchronous functionality to query the back-end when the user interacts with the front end.
6. Sanitize all user input so that the display must not interpret any HTML or JavaScript that might be typed on to the text area.
7. Allow any language as the schedule name and display it in the language it is entered. [**3 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 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.

4. What is a schedule?

Schedule is a named list of courses that a user can create so that they can retrieve the timetable for all the courses in that list. E.g. "all my Monday courses"

Appendix

A1. RESTful API – Best practices

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

A2. Data Format

Data is stored in a JSON file which can be read directly into a JavaScript array. Following is a small sample with two records. Full file is available in the Resources section on Owl. This sample is only for illustration. Please don't copy it for testing as formatting may have changed.

```
[
{
 "catalog_nbr": "1021B",
 "subject": "ACTURSCI",
 "className": "INTRO TO FINANCIAL SECURE SYS",
 "course_info": [
  {
   "class_nbr": 5538,
   "start_time": "8:30 AM",
   "descrlong": "",
   "end_time": "9:30 AM",
      "campus": "Main",
      "facility_ID": "PAB-106",
      "days": [
        "M",
        "W",
        "F"
      ],
      "instructors": [],
      "class_section": "001",
      "ssr_component": "LEC",
      "enrl_stat": "Not full",
      "descr": "RESTRICTED TO YR 1 STUDENTS."
    }
  ],
  "catalog_description": "The nature and cause of financial security and insecurity; public, private and employer programs and products to reduce financial insecurity, including social security, individual insurance and annuities along with employee pensions and benefits.\n\nExtra Information: 3 lecture hours."
},
{
  "catalog_nbr": 2053,
  "subject": "ACTURSCI",
  "className": "MATH FOR FINANCIAL ANALYSIS",
  "course_info": [
    {
      "class_nbr": 1592,
      "start_time": "11:30 AM",
      "descrlong": "Prerequisite(s):1.0 course or two 0.5 courses at the 1000 level or higher from Applied Mathematics, Calculus, or Mathematics.",
      "end_time": "12:30 PM",
      "campus": "Main",
      "facility_ID": "NCB-113",
      "days": [
        "M",
        "W",
        "F"
      ],
      "instructors": [],
      "class_section": "001",
      "ssr_component": "LEC",
      "enrl_stat": "Full",
      "descr": ""
    }
  ],
  "catalog_description": "Simple and compound interest, annuities, amortization, sinking funds, bonds, bond duration, depreciation, capital budgeting, probability, mortality tables, life annuities, life insurance, net premiums and expenses. Cannot be taken for credit in any module in Statistics or Actuarial Science, Financial Modelling or Statistics, other than the minor in Applied Financial Modeling.\n\nAntirequisite(s): Actuarial Science 2553A/B.\n\nExtra Information: 3 lecture hours."
}
]
```

A3. Suggested Steps

Coding targets: Oct. 29 for steps 1-6; Oct. 31 for steps 7-11; Nov. 5 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.
2. Complete the tutorial for creating the REST API for the "parts" database (will be available shortly).
3. Read through the entire lab handout. Read it multiple times until you have a clear picture of **what** is needed (not **how**).
4. Write a JavaScript (Node.js) function for parsing the course data file and extracting the subject codes and descriptions.
5. Set up a Node.js and Express back-end for implementing item #1: "Get all available subject codes and descriptions" in back-end functionality.
6. Write a simple front-end (HTML+CSS+JavaScript) to test the above and serve it within the same script above by using the functionality of serving a static file in Express (see class notes).
7. Revise code in step 4 to extract all course codes for a given subject code.
8. Implement item #2: "Get all course codes for a given subject code" in back-end functionality.
9. Add front-end functionality to test item #2. Revise it to get the subject code as a user input.
10. Combine steps 6 and 9 as a coherent user interface (UI).
11. Implement the back-end functionality for items 4, 5 and 6.

12. Add front-end functionality for the above step and create a coherent UI.
13. Implement back-end functionality for items 7, 8 and 9.
14. Add front-end functionality for the above step.