Assignment 5: REST API

1. General

- All ID's (like "5" in the endpoint examples below) are assigned by the back-end and are implementation specific. Therefore, you should not worry about trying to match the IDs that appear in either the curl-command script, the Postman configuration, or the sample output. The UI will adjust to work with whatever IDs you pass it.
- All commands return HTTP 200 (OK) unless otherwise stated.
- Any endpoint which accepts an ID (be it in the path, query string, or in the body) must return an HTTP 404 error with a meaningful message if the ID does not exist.
- Most arrays must be in sorted order so that the UI displays values in correct order.

GET /api/about

- Return a simple structure of some description of your app (you choose!) and your name.

GET /api/dump-model

- Trigger the model dump to the server's console. If you are running the server via IntelliJ, then you'll see the output inside the IntelliJ console.
- No content is returned to the client; this just outputs the debug information to the terminal.

2. Access Departments, Courses, Offerings, and Sections

GET /api/departments

- List of all departments.
- Each department has an ID (`deptId`) and a name (`name`).

GET /api/departments/5/courses

- Lists all courses for department with `deptId` 5.
- Each course has an ID (`courseId`) and a number (`catalogNumber`).
  - Note that the course number can be more than just an integer (such as 105W, or 2XX)

GET /api/departments/5/courses/123/offerings

- Lists the offerings of the course with `courseId` 123 inside department with `deptId` 5.
- Each course offering has:
  - `courseOfferingId`: The ID of this offering; assigned by the back-end.
  - `location`: String describing where the course is offered, such as "SURREY"
  - `instructors`: String representing the names of the instructors who teach the offering.
  - `year`: Integer value for the year, such as 2018.
  - `semesterCode`: Integer value for the SFU semester code, such as 1187
  - `term`: String describing the semester of the year, such as "Fall".

GET /api/departments/5/courses/123/offerings/4321

- Return the list of sections for the offering with `courseOfferingId` 4321, in the course with `courseId` 123, in the department with `deptId` 5.
  - For example, an offering in Surrey of CMPT 130 may have lecture and tutorial "sections".
- Each section expected to have:
  - `type`: String describing the section type (comes from data file).
  - `enrollmentTotal`: Integer holding the total number of students enrolled in this section.
  - `enrollmentCap`: Integer holding the capacity of the section.

3. Graph Data - OPTIONAL

GET /api/stats/students-per-semester?deptId=5

- Returns a list of data points showing how many spaces in courses were filled by students during each semester for the selected department.
  - Return one data point for each semester between the start and end of your data (see below).
  - Each returned data point is for a single semester and stores the total number of seats taken by students in lecture ("LEC") sections for courses in the given department during that semester.
  - Calculate this value for a given semester by:
    - Find the set of course offerings in the selected department for each semester of interest.
    - Then sum up the `enrollmentTotal` values for all "LEC" type sections of those offerings.
- Returned data expected to be an array of objects:
  - `semesterCode`: The SFU semester code.
  - `totalCoursesTaken`: The total number of filled seats in all courses offered by the selected department during that semester.
- The `semesterCode` must start at the first semester for which your system is given data, and must go up to the last semester for which you have data.
  - Use only the semesters 1 (Spring), 4 (Summer), and 7 (Fall).
  - Do not skip over a semester if there is no data for that semester: include it in the data set.
  - Array must be sorted by `semesterCode` (i.e., in chronological order).

4. Add New Offering / Section

POST /api/addoffering

- Add a new section to the data stored by the system.
  - This mimics the behaviour of dynamically adding data that could have been found as a row in the input data file.
- Returns HTTP 201: Created.
- POST message has the following fields:
  - `semester`: The SFU semester code, such as 1181.
  - `subjectName`: The department name, such as "CMPT"
  - `catalogNumber`: The course number, such as 213 for CMPT 213.
  - `location`: String representing the location, such as "SURREY".
  - `enrollmentCap`: Total number of filled seats in the class (number of students).
  - `component`: The component or section code, such as "LEC" or "TUT".
  - `enrollmentTotal`: Total number of seats in the class.
  - `instructor`: String for the instructor's name.
- Note that when data is added to the model the Web UI will not automatically refresh; user must manually reload the current page to have the data update.
- Adding a new section via this endpoint can add new offerings, courses, and even departments to the system.
- TIP: Do not repeat yourself! You already have code to do this task when you read data from the CSV file; reuse that same code.

5. Course Change Watchers

GET /api/watchers

- Lists all change-watchers.
  - A course change watcher ("watcher") is stored on the server and allows the user to "watch" a specific course for sections being added.
  - When created the change-watcher registers as an observer with the desired course.
  - When the course knows that it is changed (in response to reading a new row from the CSV file, or from data dynamically added via the API) it notifies the watcher.
  - The watcher maintains a list of descriptions of sections (i.e. components) being added to a course.
- Each watcher object returned by this endpoint has:
  - `id`: Watcher's ID, as assigned by the back-end.
  - `department`: JSON object for the department of the course being watched. Expected sub-fields are `deptId` and `name`.
  - `course`: JSON object for the course being watched. Expected sub-fields are `courseId` and `catalogNumber`.
  - `events`: Array of strings, showing the history of events it has observed.
    - Expected format of each event should be similar to the template:
      [date]: Added section [type] with enrollment ([total]/[cap]) to offering [term] [year]
    - For example:
      `Sun Mar 25 21:41:35 PDT 2018: Added section LEC with enrollment (89 / 90) to offering Spring 2019`
    - [total] and [cap] should be the new amount added by this change, rather than the total number. i.e., if you are adding an extra tutorial holding 23 students with a cap of 30 to an existing set of tutorials for a course offering, the event should show (23/30), not the much greater total of all tutorials for this section.

POST /api/watchers

- Create a new watcher.
- Returns HTTP 201: Created.
- Request body contents:
  - `deptId`: ID of the department in which the course is found.
  - `courseId`: ID of the course which is to be watched.

GET /api/watchers/42

- Get the list of events recorded by the watcher with ID 42.
- See `GET /api/watchers` section above for fields.
  (This is returning just one watched, `/api/watchers` returns an array of all watchers).

DELETE /api/watchers/42

- Delete the watcher with ID 42.
- Returns HTTP 204: No content.