SER421 Lab **REST APIs**

Activity 1: Use a REST API (20 points)

Using curl, Postman, GitHub CLI (gh), or any other REST API client, make the following calls to the GitHub REST API for public repository kgary/ser421public (our public class repo):

1. Retrieve the gitignore template for Java
2. Retrieve the set of all branches of this repo.
3. Retrieve the list of all comments on Issue 34.
4. Make a comment on Issue 34. Be sure to include your ASURITE ID in the comment body.

GitHub REST API documentation is available here.

Submission: Please put your answers in a text document, Word document, or provide a Postman collection named labRest_act1.[txt|docx|json]. If you use another tool and it has a readable output format we will accept it, ping us on Slack if you would like to doublecheck. Whatever format you provide has to give us the verb, full URL, payload (if applicable), and any headers only if they were necessary to customize. *You only have to provide the response you get back from GitHub for #4*.

Activity 2: Make a REST API using Spring (80 points)

Activity 2 is inspired by Lab 3 Activity 2, the simple survey you did in Vue. However, you are not creating an application, but rather an API that models the survey domain from that activity:

- A *survey item* is a multiple-choice, single-answer (MCSA) question with a *question stem*, a *candidate set of possible answers*, and a *single correct answer*. A *survey item* has immutable, meaning once created it stays in the same single state.
- A *survey item instance* is an instance of a *survey item* appearing on a specific survey instance (see below). A survey item instance is a copy of the *survey item* with additional fields for *answer choice* and *correct or incorrect*, and has 2 states: *Completed* or *Not Completed* based on whether an *answer choice* exists for it. *Survey item instances* are immutable once the *survey instance* upon which it appears is *Completed*.
- A *survey* is a collection of N *survey items* where 0 <= N <= 5. A *survey* has 3 states:
  1. *Created* – the *survey* exists but no *survey instances* may be started from it yet
  2. *Completed* – authoring of the *survey* is done, meaning *survey instances* may be started from it.
  3. *Deleted* – *surveys* are not physically deleted in the system, they are only marked for deletion. This is so that the provenance of *survey instances* is always maintained. New *survey instances* may not be started from deleted *surveys*, and once deleted the *survey* is immutable (can never become "un-deleted").
- A *survey instance* is a collection of *survey item instances* created as a result of starting a *survey*. A *survey instance* has 3 states:
  1. *Created* – a *survey instance* has been created by a User, but none of the *survey items* have been completed yet.
  2. *InProgress* – a *survey instance* has been started (created) by a User, and at least one of the *survey items* have been completed.
  3. *Completed* – a *survey instance* has answers for each *survey item instance*. Once completed the *survey instance* is immutable.

  I strongly encourage you to create an object model for these relationships and state diagrams so you understand them.

Your REST API will capture this 4-resource domain (we will not treat Users as a resource for this API, you can represent them simply as a String name). Provide the following API support (all response payloads should be in JSON):

1. Create a survey item.
2. Create a survey.
3. Add a survey item to a survey (note that a survey item may be on more than one survey).
4. Get the set of all surveys.
5. Get a specific survey and the set of all survey items in that survey
6. Create a survey instance of a survey for a user, with the associated set of survey item instances.
7. Accept an answer for a survey item instance on a specific survey instance.
8. Retrieve the set of all survey instances in a given state; if no state is given return all survey instances.
9. Retrieve a specific survey instance with all of the survey item instances contained with it.
10. Delete a specific survey.

8 points for the successful implementation of each of the 10 API calls. "Successful implementation" means more than "it works"; it also means you are following RESTful design principles such as:

1. It works!
2. Proper formulation of URL endpoints
3. Proper use of HTTP verbs
4. Proper navigational aspects in the response payload (payloads should be json)

5. Proper use of status codes, including error codes
6. Good documentation for each endpoint
7. (1/2 pt each) A passing test case and a failing test case. For test cases, provide a Postman collection. This should have a total of 20 HTTP requests in it, one passing case for each endpoint, and one failing case where you provide a bad message format.

Good documentation of an API endpoint (#6, describes # 2-5) is very important! If you do not give us good API documentation, then we will have no idea how to test it and you will lose more points! For documentation, you may do one of:

- Use an API documentation tool that takes inline code comments and generates web documentation. In the past I have asked for https://apidocjs.com/ to be used. *This is the preferred method*.
- Create a static HTML page with your documentation (you would hand-write this)
- Create a static text description in your README.

Repeat: in the past when I've given a REST assignment, some students would leave out complete documentation of an endpoint. We cannot and will not "guess" your endpoints, payload, and error formats, we will give you a zero. On a grade appeal providing the documentation, you will received -10 for not originally providing #6, and another -10 as a penalty!

It is your choice how to persist data behind your API – flat file system, database, or an in-memory data structure. Whatever you choose, you must bootstrap the system on startup to support at least 5 survey items, 2 surveys, and 1 survey instance.

Submission: Provide a zipfile of your source tree labRest_act2.zip. *Please do not include compiled artifacts and libraries*, these will make the zipfile too large. Provide a Readme.[md|txt] at the root of the zipfile that tells us anything we need to know.

A few students have asked for more extra credit opportunities so here are some opportunities:

Extra Credit (both EC1 & EC2 pertain to Activity 2, you have to complete Activity 2 to get EC)

EC1 (10): Response payloads are JSON. Add the ability to get response payloads in XML based on the appropriate content negotiation headers.

EC2 (15): Provide documentation for a complete Rest API. The endpoints asked for above provide only a fraction of the possible APIs in RESTful fashion for this 4-element domain. Think of what an appropriate *complete* API would look like (code not required!)

If you do EC1 and/or EC2 I suggest submitting a zipfile separate from Activity 2 named labRest_act2EC.zip to be safe!