`

# Fall 2024: CSE4495 : Software Testing and QA
# Assignment 1: System Test Design and Postman

**Due Date:25/ 12/ 2024 (Friday) 11:59PM (GMT+6)**

You may discuss these problems in your teams and turn in a single submission for the team (zipped archive) on elms. **Answers must be original and not copied from online sources or AI generated.** **Copies and AI generated reports will get -100% marks**

**Cover Page:** On the cover page of your assignment, include the name of the course, the date, your group name, and a list of your group members (**at most four**).

`

# Step 1 - System Test Design

In this assignment, you will design abstract test specifications for a student management system. The primary purpose of this system is to check whether students are ready to graduate from a particular degree program.

This system has a REST API that surfaces the following functions:

| Route | Method | Example URL | Description |
|---|---|---|---|
| /student | GET | http://127.0.0.1:5000/student | Get a list of all students, with their IDs. |
| /student/{student_id} | GET | http://127.0.0.1:5000/student/6 | Get data for a single student. |
| /create | POST | http://127.0.0.1:5000/create | Create a new student record in the database, if a record does not already exist for that personnummer. |
| /update/{student_id} | PUT | http://127.0.0.1:5000/update/2 | Update a student record. Note that changes to personnummer are not allowed. |
| /delete/{student_id} | DELETE | http://127.0.0.1:5000/delete/5 | Delete a student record. |
| /program | GET | http://127.0.0.1:5000/program | Get a list of all programs, with their IDs. |
| /program/{program_id} | GET | http://127.0.0.1:5000/program/1 | Get a list of required courses for a particular program. |
| /finished/{student_id}/{program_id} | GET | http://127.0.0.1:5000/finished/1/1 | Checks whether a particular student is ready to graduate from a particular program. |

The POST and PUT methods require, as input, a JSON structure representing a student. The allowed records in this structure include:

- name (string)
- personnummer (string, format YYMMDD-NNNN)
- courses_passed (a list of courses, each an ID represented by a string of format “XXXNNNN”, where “XXX” is a three letter department ID(CSE or DSE) and NNNN is a four digit course ID).

The student records stored in the app also include the field:

- student_id (integer).

`

A student_id is not needed for the create method, as it is assigned by the system.

A degree program is represented by a list of courses, where - again - each has a course_id.

All input is validated by the system. If you provide invalid or malformed input - either in the endpoint URL or the JSON bodies for the create and update methods - you should expect an appropriate error.

Note that the POST/PUT/DELETE methods do not actually make permanent changes in this example. You will get an appropriate response, but the record will not actually be created, updated, or deleted. You can use the result body to verify the results of running these functions.

Download the **app.py** file from eLMS and deploy it on your system using flask.

To deploy the system locally:

- Download and install python 3.12
- In a terminal:
  - Enter the directory where app.py is located.
  - Install the Python package flask: python -m pip install flask
  - Set the following environmental variables:
    - set FLASK_APP=app.py
    - set FLASK_ENV=development
    - (on mac/linux, “export” instead of “set”)
  - Start flask: python –m flask run or flask run
- Once the system is deployed, you can interact with the system using curl, Postman, or other utilities that can send requests to the endpoints defined above.
- See the following tutorial for an example of how to deploy this type of application: https://realpython.com/api-integration-in-python/#rest-and-python-tools-of-the-trade

**For each endpoint, except for /student/ and /program/, Identify the choices, representative values, and constraints that you would use to create test specifications.**

- **Based on the input parameters or other environmental factors under your control, identify the choices you control when testing this endpoint.**
  - These are aspects of the execution of that endpoint that you control and can affect the outcome of executing the function at that endpoint.
    - (e.g., the student_id value used as input for /student/{student_id})
- **For each choice, identify representative input values.**
  - These are the options that you can select for that choice that could change the outcome of executing the function.
    - (e.g., “a valid student_id > 0 and < the total number of students” would be a representative value for the choice “value of student_id”)
- **Bonus: For each representative value, if applicable, identify constraints**

`

- Constraints: IF, ERROR, SINGLE
- Constraints limit the combinations of representative values that will be tried when testing that endpoint.
    - IF states that Representative Value A for Choice X can only be selected if Representative Value B is chosen for Choice Y.
    - ERROR indicates that, if this representative value is chosen, an error is expected from the function.
    - SINGLE indicates that the chosen representative value should result in a normal outcome of the function, but it should be tried one time because it is an unusual value.
- (e.g., a representative value “student_id < 0” for the choice “student_id value” would receive an [ERROR] constraint because a negative student_id is invalid, regardless of the values of any other choices made for that function being tested)

Note that you do not need to create the full list of test specifications for this problem, just identify the choices, representative values, and constraints.

Next Page – Postman Instructions

`

## Step 2 - System Test Design (with Postman)

Based on your work in the previous step, you will now develop a set of concrete test cases that can be executed using the Postman tool for testing the system through its REST API.

- **If you do not have one already, create a free account at https://www.postman.com/ and download the Postman desktop agent.**
- **Deploy the system locally, following the instructions in Step 1.**
- **Open the Postman desktop agent.**
- **Use Postman to create tests for the system from Step 1, based on your choices, values, and constraints.**
    - **Create at least 15 requests/test cases, with at least one test case for each API function.**
    - **Each test case has its own input (URL + request body) and one or more assertions on the output (called "tests" in Postman). Do not simply submit 15 assertions!**
    - **Your set of test cases should test both normal functionality as well as handling of erroneous input.**
- **In your report, include the request type, URL, body, assertions ("tests"), and any other information that you used as part of your test cases. Please explain each test case - describe the goal/purpose, as well as the assertions you used to verify the behavior. You can use Zephyr scale or similar test management platforms to manage the testing process and generate parts of your report [not mandatory].**

For a starting place on testing in Postman, see https://learning.postman.com/docs/writing-scripts/test-scripts/

Also submit the .postman_collection file as well.

Note: The test cases do not have to pass! The tests should reflect how you think the system ***should*** work, so if they fail, that indicates the system is faulty (in your view).