Project 1 Documentation

Michael Carey, Cesar Gutierrez, Angel Santoyo, Nathan Storm, Ethan Stupin, Alex Tran

CPSC 449-01 13661 Fall 2023

Task 1: Define the API

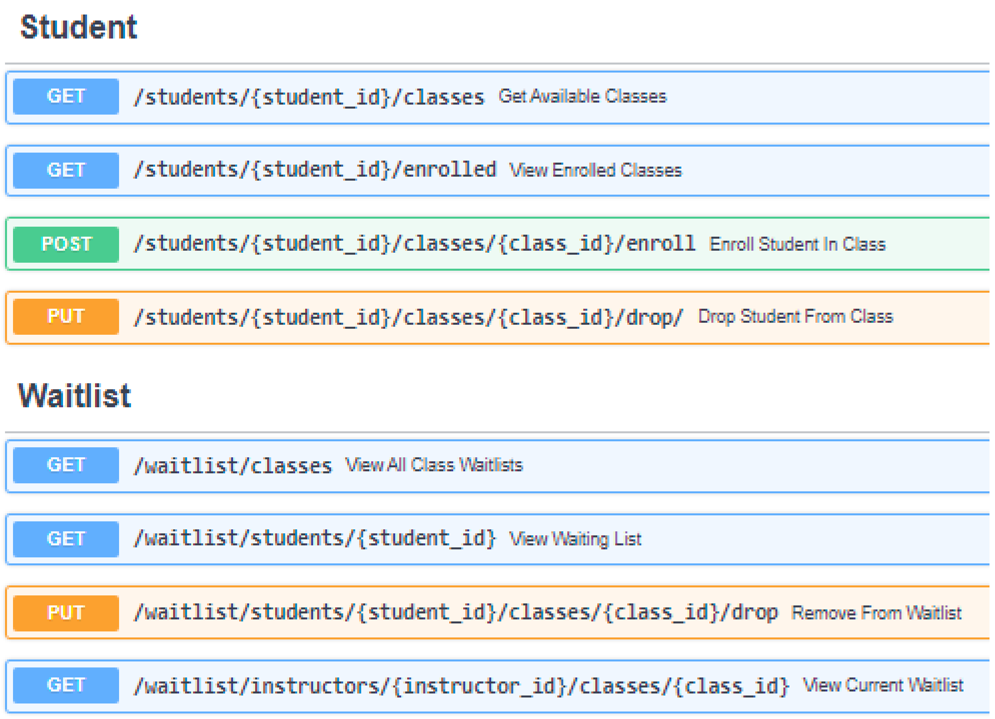
Identify what is to be done.

We had to decide as a team what endpoints that we wanted to create and how we would do it.

Document the tools to be used.

We used https://www.restapitutorial.com/lessons/restquicktips.html as a reference to help design our API.

Describe your results, including definitions, diagrams, screenshots, or code as appropriate.



Student

GET /students/{student_id}/classes Get Available Classes

GET /students/{student_id}/enrolled View Enrolled Classes

POST /students/{student_id}/classes/{class_id}/enroll Enroll Student In Class

PUT /students/{student_id}/classes/{class_id}/drop/ Drop Student From Class

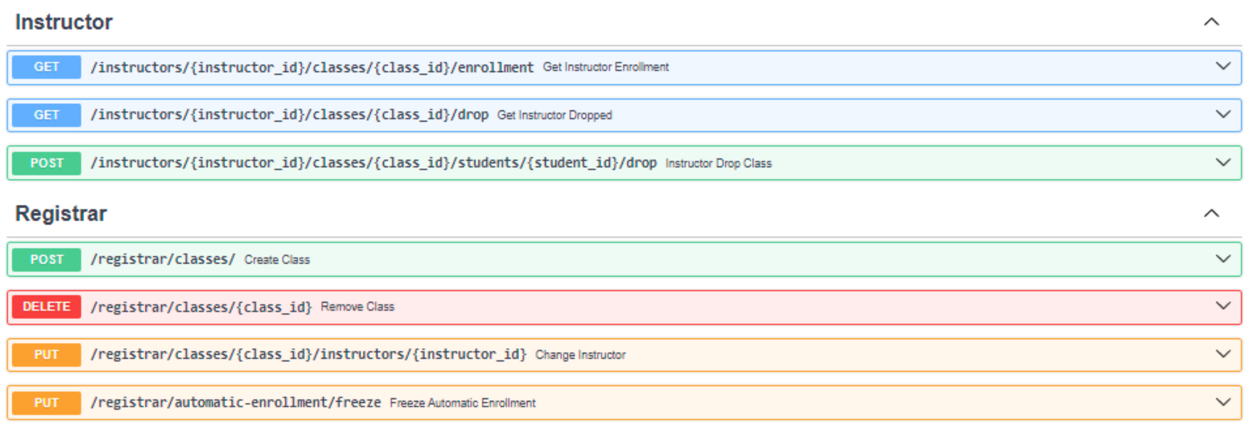
Waitlist

GET /waitlist/classes View All Class Waitlists

GET /waitlist/students/{student_id} View Waiting List

PUT /waitlist/students/{student_id}/classes/{class_id}/drop Remove From Waitlist

GET /waitlist/instructors/{instructor_id}/classes/{class_id} View Current Waitlist



Instructor

GET `/instructors/{instructor_id}/classes/{class_id}/enrollment` Get Instructor Enrollment

GET `/instructors/{instructor_id}/classes/{class_id}/drop` Get Instructor Dropped

POST `/instructors/{instructor_id}/classes/{class_id}/students/{student_id}/drop` Instructor Drop Class

Registrar

POST `/registrar/classes/` Create Class

DELETE `/registrar/classes/{class_id}` Remove Class

PUT `/registrar/classes/{class_id}/instructors/{instructor_id}` Change Instructor

PUT `/registrar/automatic-enrollment/freeze` Freeze Automatic Enrollment

Student API's

1. Get available classes endpoint: **/students/{student_id}/classes**

   REQ: Enter in your student id number as a parameter.
   RES: You will receive the list of classes available to you.

2. View enrolled classes endpoint: **/students/{student_id}/enrolled**

   REQ: Pass in your student id number as a parameter.
   RES: You will receive the list of classes you are enrolled in.

3. Enroll student in class endpoint: **/students/{student_id}/classes/{class_id}/enroll**

   REQ: Pass in your student id number and class_id of the class you wish to enroll in as a parameter
   RES: You will receive confirmation that you have been enrolled with a 200 status code

4. Drop student from a class endpoint: **/students/{student_id}/classes/{class_id}/drop**

   REQ: Pass in your student id and class id number as a parameter.
   RES: You will receive confirmation that you have been dropped with a 200 status code

Waitlist API's

5. View all class waitlists endpoint: **/waitlist/classes**

   REQ: Nothing required in the request body
   RES: A list of a dictionaries, where each dictionary contains a class id for the waitlisted class, the course code and section number for that class, name of

class, the instructor for the class, and lastly the total amount of people on the waitlist for that class

6. View the waitlist for a specific student endpoint: **/waitlist/students/{student_id}**

   REQ: A student id number
   RES: A 200 status code and a list of dictionaries for that student, each dictionary showing the student's waitlist placement number for a specific class id the student is waitlisted in. If the student id does not have any waitlist enrollment then a 200 status code and a message saying "The student is not on a waitlist"

7. Remove a student from the Waitlist endpoint: **/waitlist/students/{student_id}/classes/{class_id}/drop**

   REQ: A student id number and a class id for which the student is waitlisted in
   RES: A 200 status code message saying, "Student removed from the waiting list"

8. View current waitlist for a specific instructor endpoint: **/waitlist/instructor/{instructor_id}/classes/{class_id}**

   REQ: A instructor id and a class id that the instructor teaches
   RES: A 200 status code

Instructor API's

9. Get instructor enrollment list endpoint: **/instructor/{instructor_id}/classes/{class_id}/enrollment**

   REQ: You will pass in the instructor id as a parameter and the class id.
   RES: The response will be a list of students enrolled in the instructors class id that was inputted.

10. Get instructor dropped enrollment endpoint: **/instructors/{instructor_id}/classes/{class_id}/drop**

    REQ: You will pass in the instructor id and class id as the parameter
    RES: Should receive a list of students that have dropped the class

11. Administratively drop student endpoint: **/instructor/{instructor_id}/classes/{class_id}/students/{student_id}/drop**

    REQ: You will pass in the instructor id, class id, and student id.
    RES: Should receive a 200 when the student is dropped by the instructor from the class.

Registrar API's

12. Create a class endpoint: **/registrar/classes/**

    REQ: A dictionary that contains a unique integer class id number, string class name, string course code, a integer section number, integer current enrollment, integer max enrollment, integer department id number and lastly a integer instructor id number

    RES: A json dictionary that is a copy of what was in the request information

13. Remove a class endpoint: **/registrar/classes/{class_id}**

    REQ: A class id of the class the Registrar wants removed
    RES: A 200 status code and a string message saying, "Class removed successfully"

14. Change a instructor for a class endpoint: **/registrar/classes/{class_id}/instructors/{instructor_id}**

    REQ: A class id of the class, and the new instructor id to replace the old instructor id for that class
    RES: A 200 status code message, and a string message saying, "Instructor changed successfully". If the class does not exist, a 404 status code with a string message saying, "Class not found"

15. Freeze automatic enrollment endpoint: **/registrar/automatic-enrollment/freeze**

    REQ: No parameters required
    RES: A 200 status code and a string message saying, "Automatic enrollment frozen successfully"

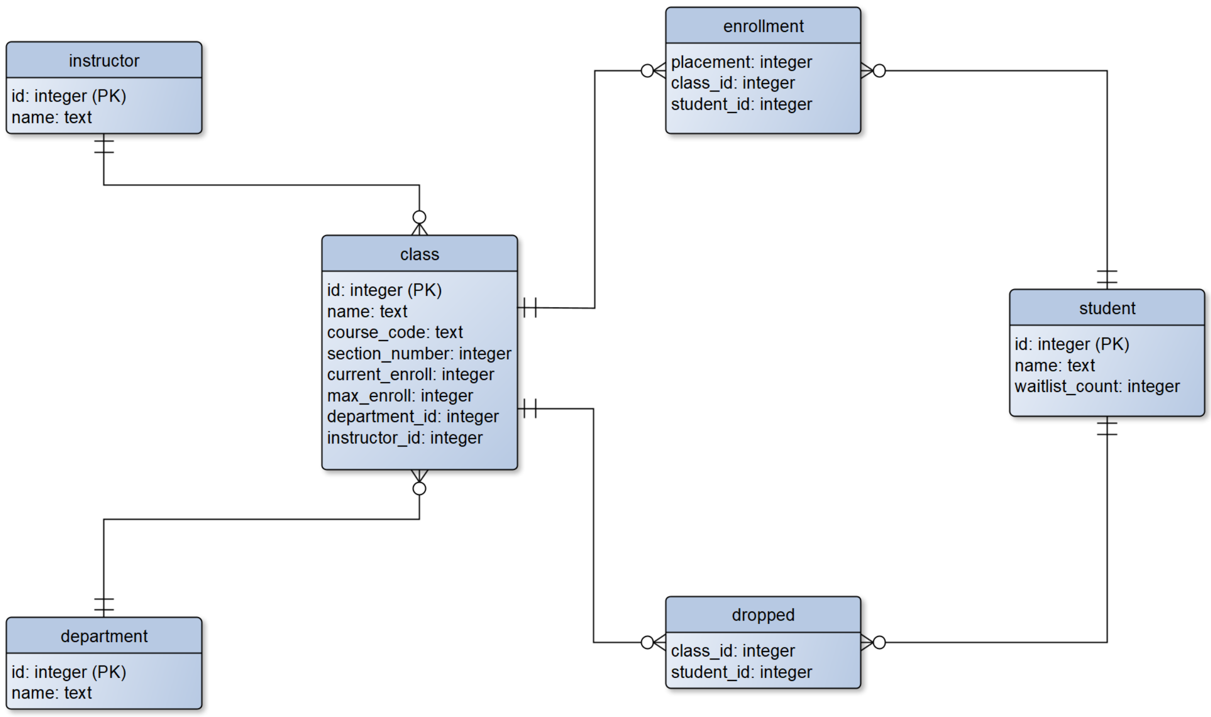
Task 2: Design the database

Identify what is to be done:

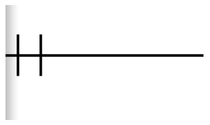
We will need to design a database that will be able to hold all the students, classes, professors, etc.

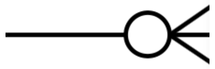
Describe your results, including definitions, diagrams, screenshots, or code as appropriate.

After some discussion, we decided to go with the design listed below.



For reference:


represents a **one and only one** relationship.


represents a **zero to many** relationship

We also wrote a populate.py script to populate our database with mock data.

Comment on the results, including references you consulted, variations you considered, or issues you encountered.

Our initial database design did not have the dropped table. It was only when we started implementing the api endpoints did we realize we needed to include a new table to keep track of which students dropped which classes.

Another thing to note is that our database separates both instructors and students into different tables, even though they are both technically users. It works out for the current implementation, but for project 2 we would most likely need to combine the tables into a new user table.

Task 3: Implement the service

Identify what is to be done.

We need to take the API design listed above and build our endpoints in python using FastAPI. We then need to write the logic to query the database information required for each endpoint and return it.

Document the tools to be used.

Python programming language, FastAPI, Foreman,

Describe your results, including definitions, diagrams, screenshots, or code as appropriate.

We were able to implement all the endpoints that we wanted to create from Task 1 and connect them to the database we created in Task 2.

Comment on the results, including references you consulted, variations you considered, or issues you encountered

When we realized we needed to keep track of students that have dropped the class, we tested out various implementations to achieve this. Initially we had the information be stored in a list of ‘dropped’ classes. But this implementation started to require more code than was worth it. We ended up deciding it would be easier to implement our current solution, which was to simply add in another table in the database that keeps track of students who drop classes.

We also messed around with various implementations to handle waitlists, but ultimately decided on our final implementation, which was to have an attribute in the ‘enrollment’ table, placement, keep track of a student's placement in a class. If the student's placement ever reached above the max_enroll attribute, then we knew they were on the waitlist for said class. It also allowed us to order those in the waitlist, as their placement in the class was dependent on when they enrolled in the class relative to other enrolled students.