MyUniPerformance
Final Report

CIIC 4151/INSO 4151 Capstone Project

Professor: Dr. Marko Schütz-Schmuck

Team members:

Félix Dasta

Armando León

Emmanuel Hernandez

Table of contents:

* New information highlighted in this color

* New information highlighted in **this color**

1. Introduction

1.1 Background and Motivation

Given how broad the academic field is, it is highly likely that students will have doubts in multiple aspects of academic life. From the perspective of the student, these aspects include, but are not limited to:

- What courses have I taken?
- Which courses am I missing to complete my degree?
- What GPA do I currently have?
- How does my academic performance compare to that of other students in the same curriculum as me, or those who have taken the course under the same conditions as me?
- Which professors will teach the course that I want to take next semester?
- What feedback have students who have taken the course left to help me achieve my academic goals?
- I want to see the course syllabus beforehand, but I don't want to search for those individually one by one. Can I extract it directly from my curriculum?

These needs raised by students are our motivation to develop a platform that helps them to plan in a better way their academic career.

1.2 Justification

There exist similar platforms, however, they are found on separate web pages and some also carry a cost to use the same. Given that all the previous elements mentioned are related, considering a single platform to attend the needs of the students will be a good idea. This, and taking into consideration the tool that will be capable of comparing the student performance relative to the performance of their peers based on historical data, will make the product unique.

2 Problem Statement

2.1 Problem Definition

Some universities don't have tools to help students to plan their academic life in a better way. Of course, there are numerous free tools that perform tasks such as GPA calculation, but they lack functionality beyond that, such as storing courses that have been taken and displaying them in an organized manner that can allow the user to see their academic progress at a glance. There exist fewer possibilities where the user might be enrolled in a second major, or they might be undertaking a minor degree. These less common situations are not covered by free existing applications. There are also tools to see feedback from professors with whom the students would like to take a class, but these tools often do not provide specific feedback based on a certain course and section. Finally, many students would like to compare their performance relative to that of the peers in their class, to know if they are on the right track throughout their academic career. However, there is no public platform that allows us to extrapolate data anonymously from people that have taken the same courses as the student.

2.2 Target Domain

The domain that our project belongs to is that of ***academic success***, however our boundaries only extend to include ***preparation***, ***performance analysis and tracking***, and ***effective feedback resources***. By performance analysis, our project will be offering various statistical calculations that'll help the student user analyze their progress and performance compared to themselves, to past peers, across departments, and even across time with past data. In terms of tracking, the student user will have access to their department curriculum and have the capability to log previous classes taken (including their grades); in an academic sense, this will allow them to have a clearer picture of where they stand. Lastly, referring to preparation and effective feedback, our project will provide the student users with a platform to fairly rate the previous course sections that they have taken and to offer constructive criticisms and praises. This is done with the express intent to provide both students and educators with the means to engage with valid feedback to aid both student preparation and educator introspection, the end goal being to foster an environment of growth.

2.3 Proposed Innovation

The solution we propose to materialize these perspectives, features, and ideas is to provide a digital platform in the form of a website. This website will provide its users with the tools they need to fulfill and track their academic progress and

improve upon it. However, it is characterized as a digital platform primarily because a key feature is its anonymized feedback system where users can share their knowledge and experience with courses they have previously taken. Consequently, this will stimulate a positive feedback loop where the community of students can help themselves, each other, and their professors with the aim of improving the overall university experience.

3 Project Objectives

3.1 SMART Objectives

- As of February 17, 2022, we expect to have the project proposal completed. In this project proposal, the problem statement, objective, solutions, technical description and project plan will be established. Those details will aim to facilitate the development process of the project, which will be started by February 18, 2022. Technical description such as system architecture, modules, ER Diagrams, Wireframe for UI, flowchart for algorithms and engineering constraints will be taken into account for the development process.

- The main objective of this project is to develop an application that will provide a unique user system, a curriculum viewer, a performance reviewer and course details by May 6th, 2022. Our team will achieve this set goal by operating in an Agile manner that will allow us to work on pre-defined issues, which will be agreed upon by the team in a meeting. By dividing the project into stories, we will be able to create shares of equal work that will allow us to meet our goals. The sprint period we will use will be two weeks of length.

3.2 Task Completion Definition

The team will ensure that the stories created will be clearly defined. Once individual stories are defined, corresponding acceptance criteria will be created. The purpose of this is to create thorough acceptance tests based on the defined criteria, so that the task can be verified as having been successfully completed. Stories will be created using the Trello platform, given that it allows us to keep track of who has been working with which tasks, provide comments on some specific tasks (to have those for future reference), and to keep track of the amount of work which each of the members of the team has accomplished, to ensure that everyone is being productive.

3.3 Success Definition

Since most of these stories will encapsulate features of the platform, once finished we will also be able to test said features. This will allow the team to measure the success of development. Tangible goals will grant the team a sense of fulfillment once a task is finished. As we are aware of this, productive planning sessions are important to help meet the deadline established. By successfully completing this

project, we will produce an application that will allow students to further improve their academic performance and allow them to fulfill their academic goals more efficiently.

4 Solution Approach

4.1 Solution Proposed

Based on the academic characteristics that our university community requires so that they can be better equipped to have a successful college experience, it was determined that the following features will be necessary for the system-to-be:

- **Unique user registration:** We want to limit the amount of spammers in the system, given that we want the users statistics to be as accurate as possible. Hence, we will be limiting registration to users with an academic institution address, and they will have to verify their account via an email that will be sent to them.

- **Student curriculum viewer:** Student curriculum is used so that the students can benefit from access to their GPA calculation by semesters or academic year. Through predefined curriculum templates provided by different universities, students will be able to choose the curriculum that corresponds to them so that they can start entering their records. If the university is not in the system, we will allow the student to create the curriculum based on the one provided by their university. However, if the student wishes to benefit from all the app features, by using a form, students will need to notify the administrators of the page which university they wish to add to the system and their corresponding curriculum. While the students are looking at their academic curriculum, they will be able to see the details of the courses from their curriculum.

- **Academic minor feasibility:** It is likely that many students want to complement their academic major with a minor, therefore the students will be able to see the academic minors that are available at the university. When the student selects the desired minor, they will see which courses they need to complete, and have already completed, to obtain the minor degree.

- **Course details:** The details of the course will include the syllabus, the professors who have taught that course and any other relevant information. To clarify, our purpose is to provide constructive criticism, so that the student can learn from the professor's teaching tactics beforehand. The students will be able to see advice from other students in order to succeed in the course.

- **Student performance reviewer:** When students get their course grades and enter them into the system, they will be able to compare their performance relative to that of other students in the course anonymously. This will allow them to see if they are on the right track, academically, relative to their classmates. They will be able to evaluate the performance based on different criterias, such as, for example, extrapolating all the historical data anonymously of all the students who have taken that course previously or making a comparison based on the students who took that course in the same semester, since it would adapt more to the circumstances of that moment.

4.2 Gap With Existing Solutions

Currently, there is no such platform known to be developed that is capable of giving the students an insight of their performance compared to that of their classmates based on the same course sections taken. Many universities have public data uploaded of the anonymous grades given at a course section at a specific term, but it's not only about having the data, but rather, what we can do with it. Hence, with the available data, we can infer multiple aspects of what influenced the grades of that given semester, such as the course modality, the evaluation techniques, among many other factors.

However, the platform we propose provides some features that overlap with the features of other available solutions. One of them is the feature to provide feedback for an instructor. Even though our solution provides a similar feature, it differs in some aspects that definitely adds some value. Rather than providing feedback to an instructor, the feedback will be provided directly to the section where a given instructor teached at some semester. Why would this be helpful? Because as mentioned before, there are some factors that can influence the teaching techniques of a given semester. For instance, during Spring 2020, when the COVID-19 pandemic hit globally, the institutions were forced to move their courses online. This modality was new to many professors and were concerned about the fact that many were not prepared to teach in this new modality. Hence, feedback from that circumstance will be different to newer feedback, where the instructors are more adapted to this modality. By differentiating those aspects, the user will be more clear about the fact of the factors that could have influenced the instructor teaching techniques at a given term when viewing some feedback.

Last, but not least, the most similar gap between our solution and other solutions are the class tracking features. Academic organizers in the market provide the user with capabilities such as class tracking but the user must manually enter the course, so in essence it is like a note taking application. Our solution will provide the user, given that the institution they are a member of is participating in the platform, with a more intuitive solution. The user will be able to see their curriculum and keep track of which classes they have taken without having to type all of the courses manually.

4.3 Value Proposition

This project will provide users with an at-a-glance overview of their academic careers, as well as useful information about their study programs. All this information will be offered in the convenience of a single platform. We believe that providing students with information on how other students have performed in a specific course will give them an extra confidence boost to tackle and progress through their academic paths. If a user is ever interacting with a potential employer or a recruiter, and they are not able to remember information about a specific course they have taken, our easy to access curriculum viewer and progress tracker will allow them to quickly verify as well as provide them with other information, such as their GPA. Aside from benefiting students, professors will also be able to benefit from the platform. The feedback system will allow professors to see what different types of teaching styles work better for students. This extra information may allow them to more easily reach a decision on changing or modifying their teaching style. Most importantly, the user can feel sure of the fact that the feedback is valid due to the strict user registration that our application will have.

4.4 Benefits and Limitations

Due to the nature of some of the platform's features, some limitations may be encountered if there is not an established user base. One of these features would be the feedback system. If there are not enough users participating, there might be a case where a specific professor or section might not have any feedback. Another feature that could be affected by lack of an established user base would be the student ranking system. Without enough students participating, it would be difficult to produce an accurate ranking, thus affecting the functionality.

If a consistent user base can be created, then the platform would be able to work to its full potential. This would allow users to be able to take advantage of accurately working features. Users will benefit from the platform by being able to track their performance relative to their classmates, view school curriculums, quickly access and learn about what a professor's teaching style is like to better choose a section, and in the case of a professor, they would be able to have access to the constructive feedback left by students.

4.5 Commercial Potential

Although the desired model would be a completely free to use model, overhead costs such as hosting will require a monetary cost to be able to upkeep the platform. Luckily, revenue producing techniques such as ad spaces would allow us to meet such costs and keep the platform hosted. The downsides to this would be that user experience would be affected by said ads. An alternative would be to implement a premium user account, which would allow the user to pay a one time fee to remove advertisements from their viewing experience permanently or for a defined time period. This would be completely optional and would not offer any other advantage or features over non-premium users, aside from no advertisements. Another alternative would be to allow public donations to our platform. Due to the value that we believe we can offer users, a space for donations would allow the platform to run less advertisements, and in a best case scenario might allow a completely ad free experience.

4.6 Required Resources

From the perspective of the users, the required resources from them to use the application will be an electronic device with appropriate hardware and software requirements (computer, smartphone, tablet, etc.), electricity to power their electronic devices, a stable internet connection, and a higher education institutional email address. A device with appropriate hardware and software requirements is defined as a device that is capable of running a browser that supports HTML5.

From the perspective of the product owners (which will be us, the team that will be in charge of developing the application), we're going to need the same resources as the users to test the application. Apart from these resources, we're going to need certain computer programs to develop the application, such as an IDE that is capable of compiling Python code and JavaScript code, and an application to access the database tables. Certain packages from different APIs are required to

facilitate our development process and to be able to deliver our product in time as well. These packages include programming frameworks on each language such as Django Rest Framework for Python and React framework for JavaScript. In order to make our product work properly, we're going to need the following information from the universities that will be participating in the system-to-be: courses offered, historical course section data with the instructors and term given, the department from each course and instructor, and preferably, the grade statistics of each section in order to provide accurate information to the students.

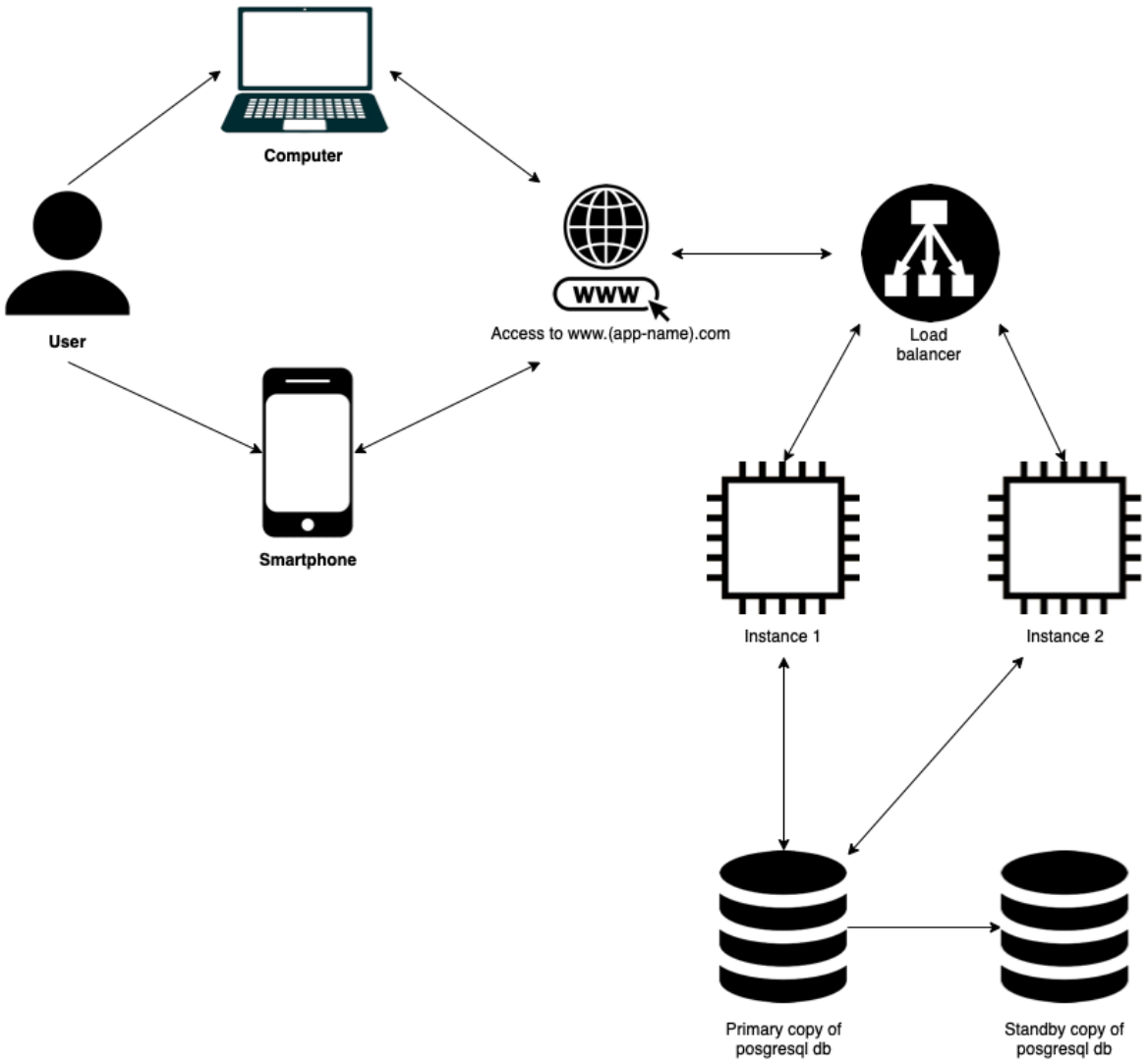
In terms of hosting, monetary resources will be required to keep our server running on the cloud with the appropriate hardware requirements that are necessary to handle the expected load. Relevant hardware requirements for hosting are established in the system architecture diagram in Section 5.1.

4.7 Intellectual Property

Our platform will not make use of any intellectual property. The data that we will be leveraging for statistics is available for the public domain at no cost. Any logo or art design that the team decides to use, will be compliant with trademark and copyright laws. After researching for trademarked applications that are similar in scope to our platform, we were not able to find any that matched the criteria. Upon opting to sign up for our platform, users will have to agree to our terms and conditions, in which they will be allowing the platform to freely use any and all content they might produce while using our platform. This includes comments, reviews and any other content produced by the user. In regards to the intellectual property for the platform itself, the team is currently evaluating the available types of free use content licenses and will decide on a specific license before the culmination of the project.

5 Technical Description

5.1 Define the system architecture



In order for the user to be able to use our application, what will be required from them is a device that is capable of connecting to the internet and that has a browser to access the web. However, from the perspective of the owner, since the expectation of it is that it can be used for multiple universities, it is possible that there are many users connected simultaneously. Thus, the server will need to be capable of handling load balancing. Based on our research, AWS seems to be a viable option for us, given that the hosting cost will be proportional to the amount of use and they have servers in multiple regions around the world, making it feasible to the user so that they can connect to the closest available zone. Also, in order to improve application performance and save money, most accessed data will be stored on the server by using caching techniques (such as Redis) provided by AWS, so that it is not necessary to have to look up the information all the time in the database. When a user connects to the application and they get redirected to the website by connecting to the closest availability zone, it is highly likely that some data related to their university is already in the server cache due to the fact that users from a certain geographical location will be more interested in knowing information about a university that is closer to them.

5.2 Modules To Implement

5.2.1 User Interface Component Modules

- **Navbar:** Gives the user easy access to the 4 main views supported by the application: Dashboard, Professor Feedback, Course Feedback, and Curriculum.
- **Search:** Gives the user the choice to quickly search for a specific professor or course instead of searching for them within their curriculum.
- **Curriculum:** This component will contain the list of courses that pertain to a given semester that the user has completed or is currently enrolled in. Each course listing will have relevant information and will allow the user to directly enter that course's feedback page or it's professor's feedback page.
- **Stat Widgets:** There are various components across all views supported in the web application which require the displaying of statistics in the form of graphs and numbers; however, they all share similar attributes and structure. For example, the grades obtained by the students in a section will be represented as a pie chart.
- **Featured information:** In both feedback views there are components that show selected feedback which are either popular or recent and therefore relevant at a first glance for the user.
- **Feedback:** For both feedback views, this component will be used to represent the feedback that a student has left.

- **Filter:** Various views, notably the curriculum, make use of this family of components which are used to limit the scope of the pool of information for the ease-of-use of the user.

5.2.2 User Interface Pages Modules

- **Home Page:** The home page serves the purpose of introducing a potential user to the intended goal of our web application. There is an image carousel which has rotating views of the system as well as brief descriptions of their purpose. Additionally, there is a short demonstration video that will showcase use cases and key features of the web application. Finally, there is an information section which explains more in depth the motivation behind the app's creation as well as an introduction of our team.

- **Student Dashboard:** The main face of the application, the dashboard's purpose is to provide the user with all the information they need quick access to. This includes their academic profile information as well as various important stats such as GPA and missing credits. Right in the center is the user's current curriculum with interactive elements that display any chosen course along with some brief information about it and its professor.

- **Professor Feedback:** This page serves more as a collection of summaries which can allow a student to understand how it is that a given professor teaches. It includes a course picker, where students can see feedback for sections of specific courses that the given professor teaches. Additionally, it has a "quick facts" section which will provide information such as average student grades as well as featured feedback.

- **Course Feedback:** This page is a key feature of our platform where students can help their professors out by giving feedback about the techniques they use and what they think of the class they took. Of critical note is the course rating which allows a user to understand the difficulty level of the course compared to the past. The user can view the feedback and ratings left for all professors that have given the chosen course across all time ranges (as is feasible). Even more, the user can access the course syllabus (if available) as well as view average grades and featured feedback for each section.

- **Curriculum:** The last of the supported views, this page allows the user to access the curriculum they are currently enrolled in. Information pertaining to curriculum specifics such as department and credits is present on the bottom panel of the page. The curriculum can be filtered by year and semester and is accompanied by a side panel that shows the user which courses they've yet to take. Furthermore, these missing courses can be filtered by all sorts of categories so that future course planning is streamlined and simplified.

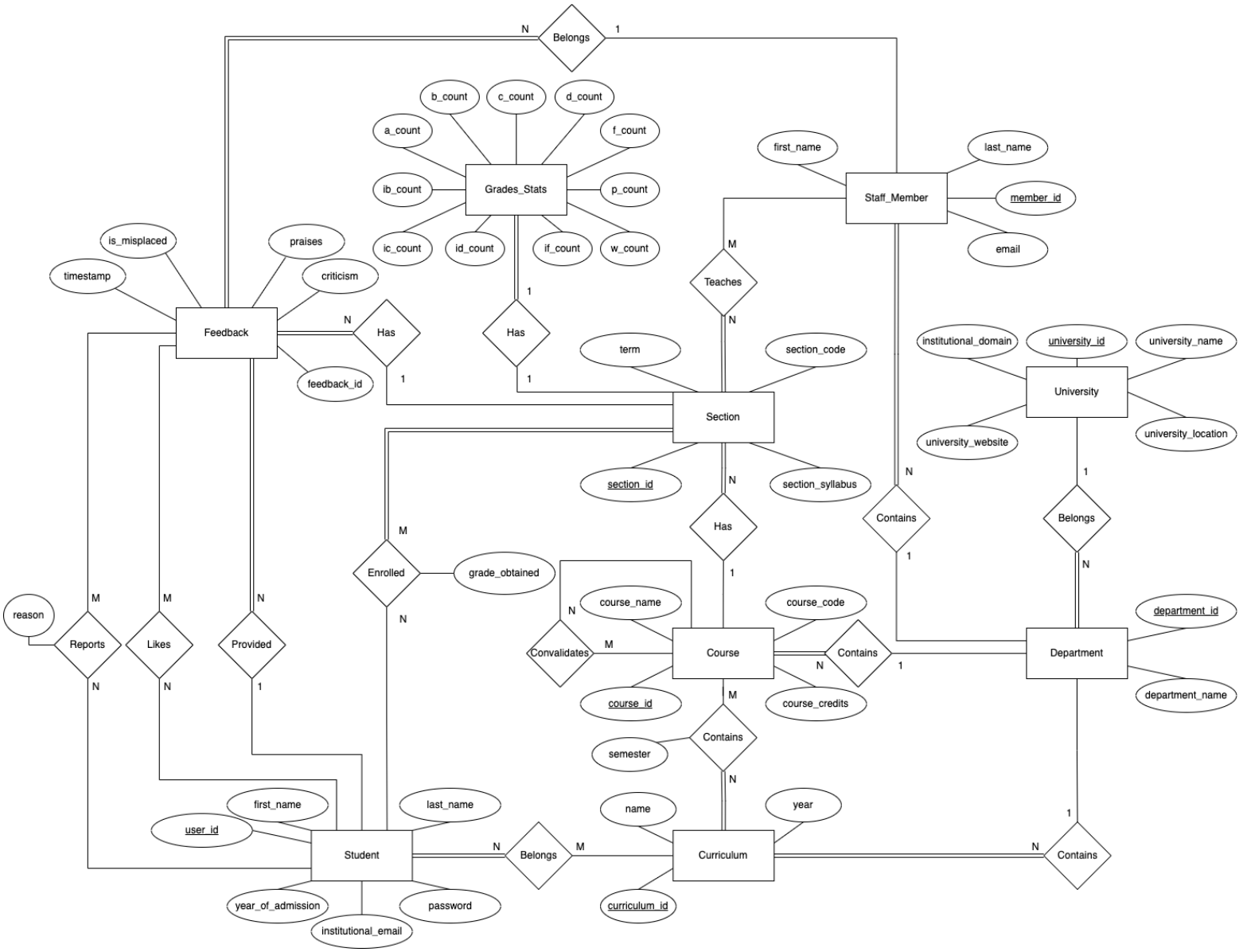
5.2.3 Object Relational Model Modules

An object relational mapper will be used to manipulate data from the database using an object-oriented paradigm. By using an ORM, this will make the code easier to update, maintain and reuse, given that we will manipulate data as objects. Also, it will shield our application from “SQL Injection Attacks”. The object parameters and relationship with other entities that will be used in the ORM can be seen in section 5.3, where the ER Diagram is being described.

5.2.4 Algorithm Utils Modules

- **Get general GPA:** This function will get all the courses that the user has registered in the database, and it will calculate the general GPA.
- **Get departamental ranking:** Retrieve every student stats that are available at the student's departament, and get ranking based on other students' that belong to the same curriculum.
- **Get average course GPA:** Based on historical data from students that have taken a course, average course GPA will be retrieved. Letter grade counts will also be retrieved from the courses.
- **Modify semester GPA:** If we want to add a course to a specific semester, the course will be added and the GPA for that semester will be modified.
- **Map course grade stats:** Course grade stats will be mapped to the course object. Grade stats will be retrieved from real grade stats data if it is available, otherwise, it will be interpolated from the registered students of the system.
- **Verify profanity on feedback:** Based on the desired feedback that will be published, checked if the text may contain some profanity. If it doesn’t pass the profanity checker filter, mark the comment as misplaced.

5.3 ER Diagram and Description



5.3.1 Relationships

1) **Student and curriculum:** It is possible that the student belongs to multiple curriculums (example: student is taking a second bachelor degree or is complementing their major degree with a minor degree), and it is possible that the academic program to which the curriculum belongs to contains multiple students (typically, that is always the case). Also, all elements from the student side must have a curriculum.

2) **Department and curriculum:** It is possible that a single department has multiple curriculums, however, every curriculum must belong to only one department.

3) **Curriculum and course:** A curriculum should contain multiple courses, hence, the courses that the curriculum must contain should be created before the curriculum. Also, a course can belong to multiple curriculums (example: calculus courses).

4) **University and department:** A university contains multiple departments and a department belongs only to one university (every department in each university is unique, each with its corresponding staff members and courses). Also, a department can't exist without the university, given that the university is responsible for having multiple departments for its function.

5) **Department and staff member:** Every department can have multiple staff members but every staff member must belong to only one department. Note that there are cases when a professor can give courses that belong to different departments, however, under the typical university rules, they are an official member of one department only (example, a professor of electrical engineering can be teaching a computer science course).

6) **Course and section:** Every course can have multiple sections, however, every section should belong to only one course.

7) **Department and course:** A department can contain multiple courses, however, each course should belong to one department only. It is possible that some newly created departments don't have any courses yet.

8) **Staff member and section:** A single staff member (professor or teaching assistant) can give multiple sections, and each section can be given by one or more staff members. Typically, the majority of sections will be given only by one instructor, but in laboratories, it is common to see that there are at least two instructors collaborating together to teach the section.

9) **Section and grade stats:** A section can have records for grade stats, and each record belongs to one section only. Grade stats can't exist without a section. Note that it might be possible that some universities don't have public records of grade stats, hence, it is not necessary for a section to have grade stats.

10) **Feedback relationship between student, section and staff member:** A student can leave multiple feedbacks, either to one section only or for more than one section. Also, a section can have multiple feedbacks, either from one student only or from more than one student. But each feedback is left by one student only and corresponds to one section only. Also, when the student leaves a

feedback for a section, if the section has more than one instructor (staff member), they can specify to which instructor they would want the feedback to correspond, given that each instructor might have different teaching tactiques as well as evaluation techniques. The student can thumbs up many feedbacks and a feedback can contain many thumbs up from different students. Thumbs up represents if the student has liked the feedback. Note that a provided feedback can be inappropriate, and it might be possible that a student provides one or more inappropriate feedback, but each inappropriate feedback belongs to one student only. Inappropriate feedback won't be shown to the public, however, those will be stored in the database so that the administrative staff can determine if the user deserves the suspension based on those comments that may contain profanity. Thus, students will be able to report inappropriate feedback. Feedbacks may be reported for many reasons, such as inappropriate content, spam attacks, inappropriate language, etc.

11) **Student and section:** A student can be enrolled in multiple sections, and a section can have multiple students, but a section can't exist without students.

12) **Relationship between two courses:** A course may be equivalent to another course, even if they belong to different departments and have different codifications, if and only if the University has determined that most of the material covered by both is the same. For example, there are different courses of introduction to computer programming at UPRM (e.g. COMP3010, CIIC3015, INGE3016) which essentially covers most of the same material. It is common that many students may have taken an equivalent course that substitutes the other course that they should have supposed to take according to their curriculum.

5.3.2 Considerations

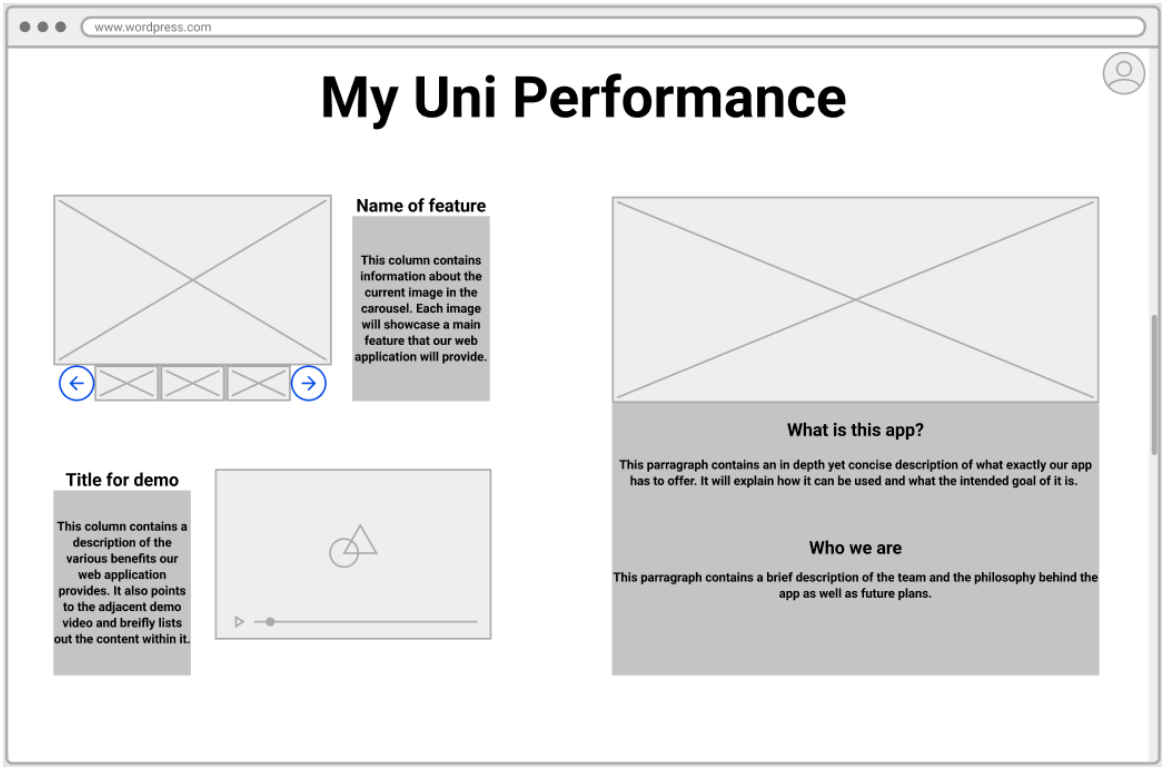
- We are aware that some courses may be affiliated with many departments. However, we discovered a tendency in which some courses appear to belong more to one department rather than the other. As a result, we're assuming that each course belongs to a single department. We're open to changing the feature in the future to allow courses with many departments to be found via various departments, especially because other institutions around the world may offer more joint programs than the universities located locally.

- Users who break our policies may be suspended. We don't have a table to keep track of the suspended users, as you can see. This is because, at the moment, user suspension is dependent on the number of times they offer incorrect feedback. If a user has given three inappropriate feedback, they will be suspended for 30 days after

the third inappropriate feedback. Our expectation is that the user will learn their lesson, but if they do not, they will be permanently banned from the system following the third thirty-day suspension. Of course, the banned users can be inferred by the amount of times they have provided inappropriate feedback, and this may be changed in the future with a table of banned users.

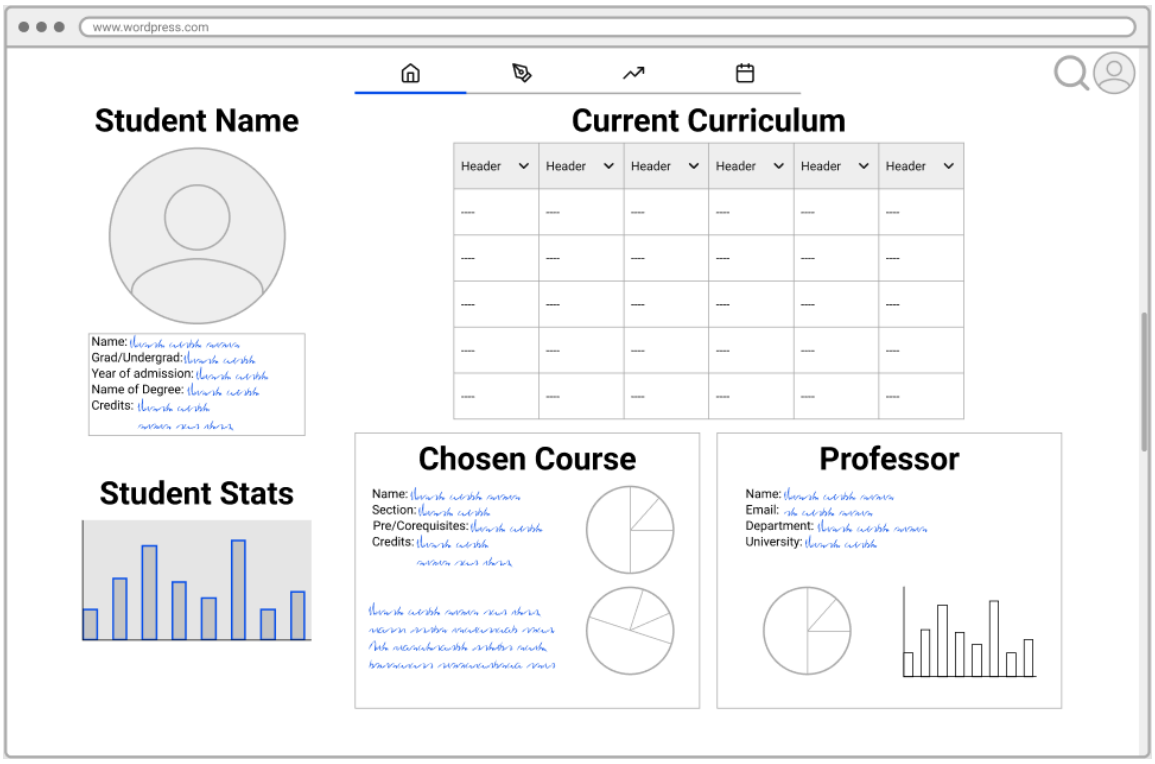
5.4 Describe Wireframe for UI

Home page



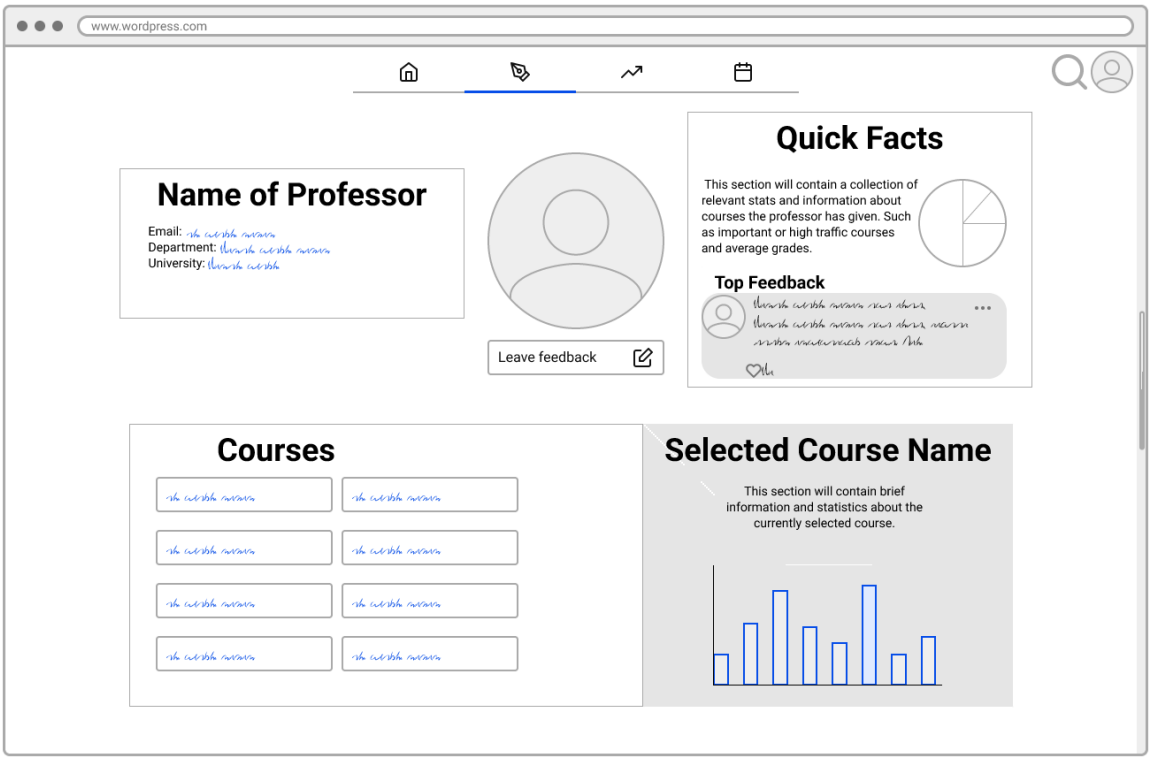
This is the application landing page, where general information about the application and its purpose can be seen.

Student dashboard



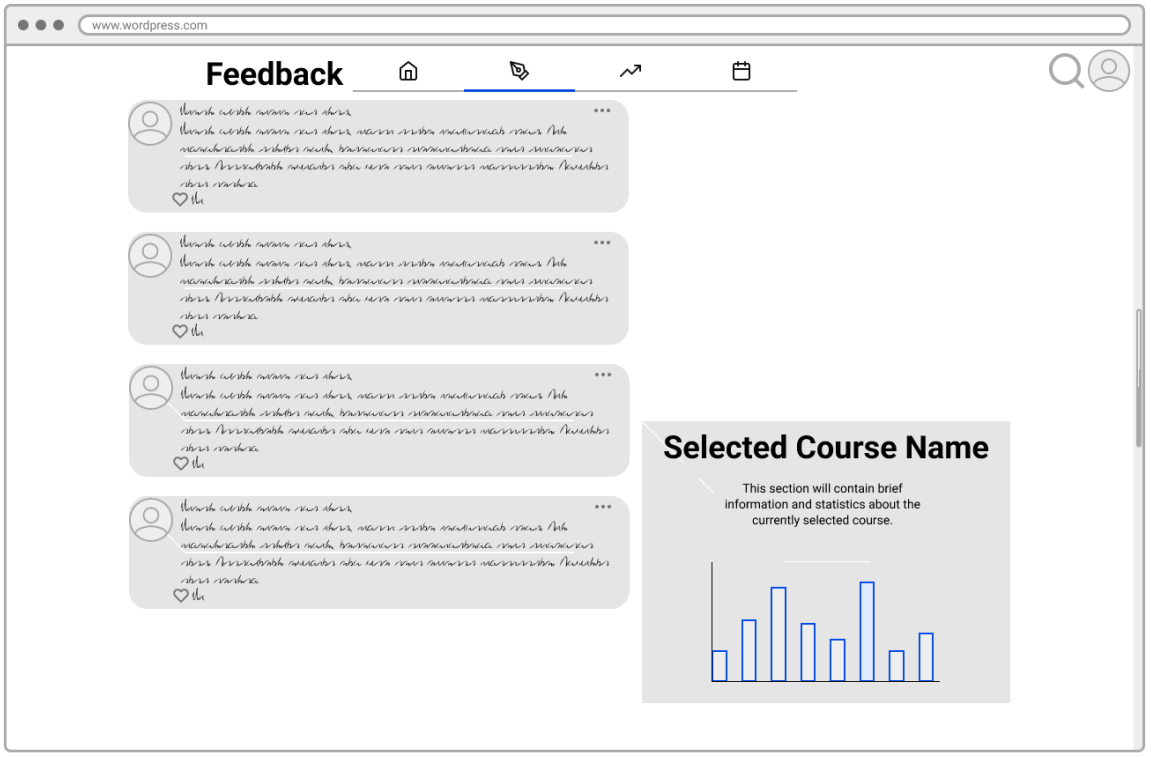
The student dashboard allows the student to have quick and easy access to all relevant information they need, such as courses taken and their statistics.

Professor feedback



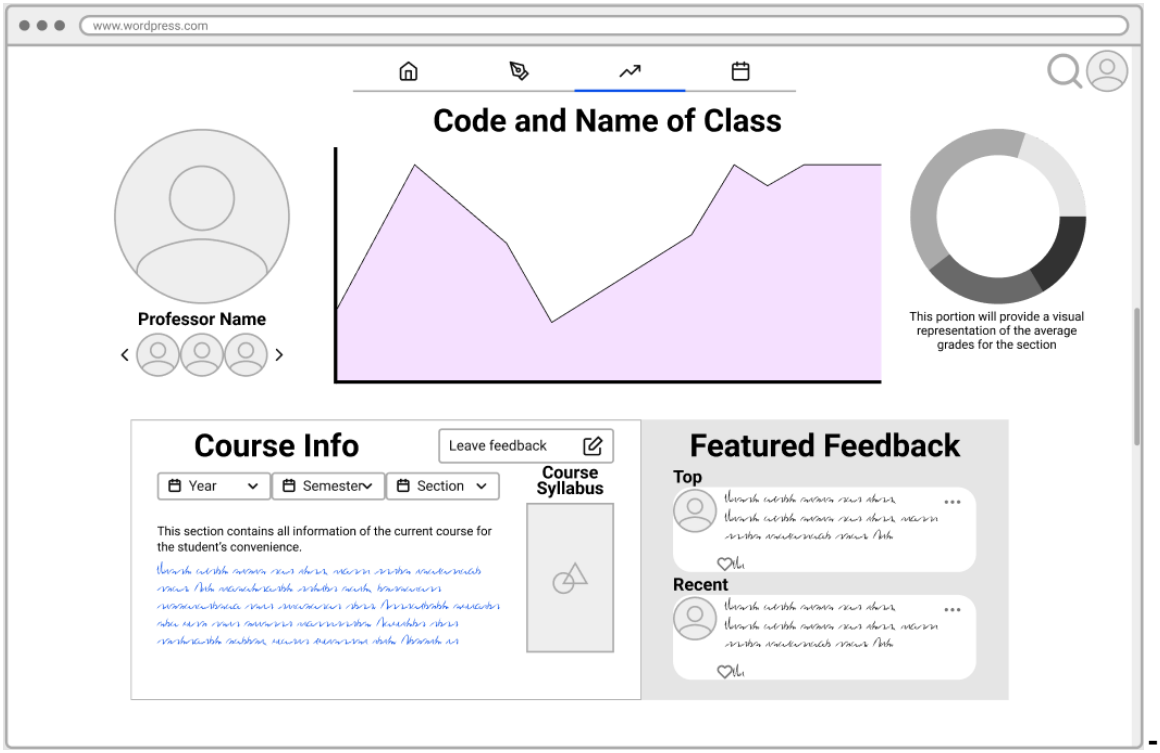
This is the professor feedback page, it allows the student to view how other students have rated a professor before, and it allows the student to leave their own feedback.

Professor feedback feed



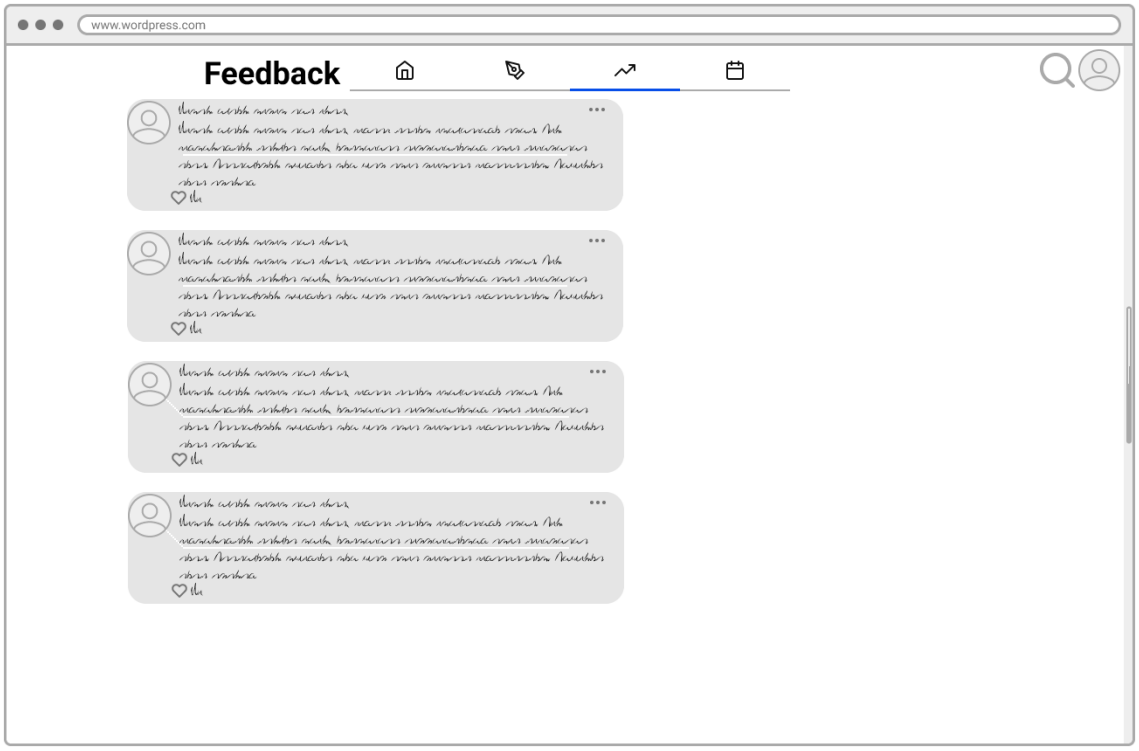
This is a view of the feedback feed for a given professor based on the sections they have teached.

Course feedback



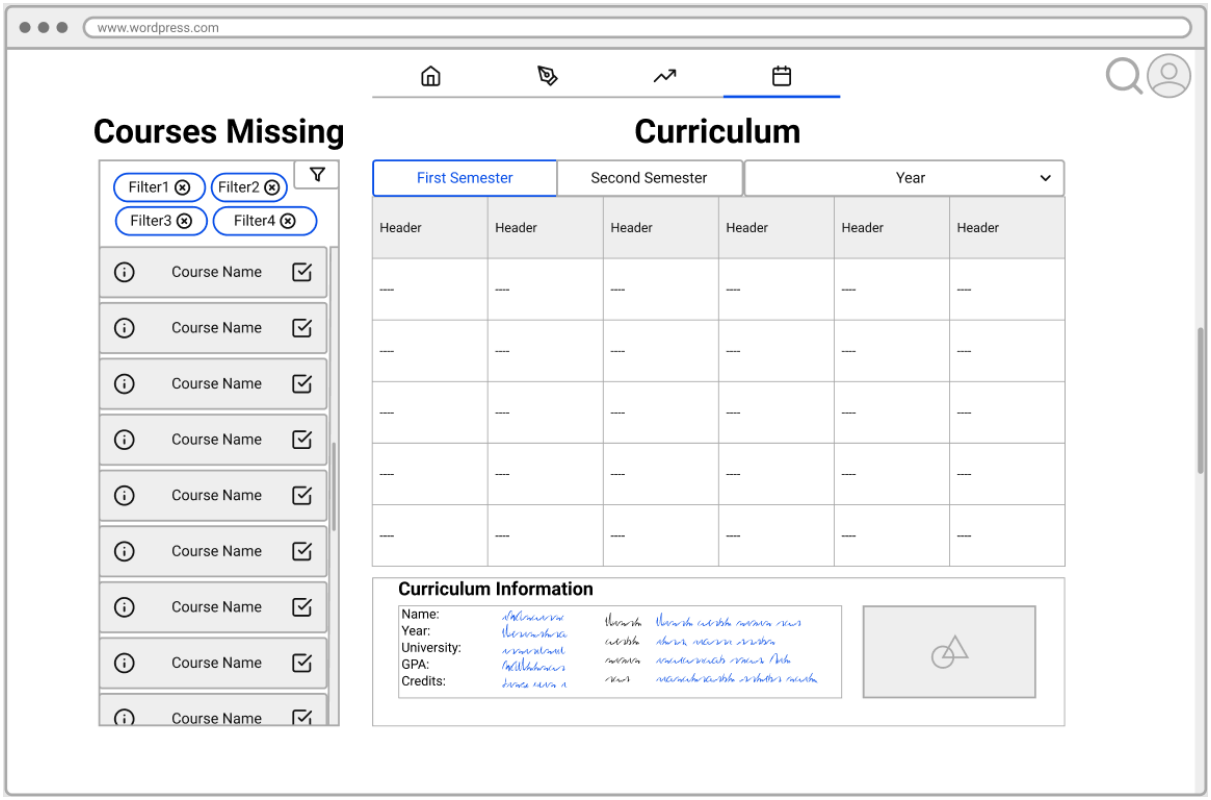
This is a view of the course feedback page, it allows the student to gauge how a course has been rated in the past and relevant statistics that pertain to it.

Course feedback feed



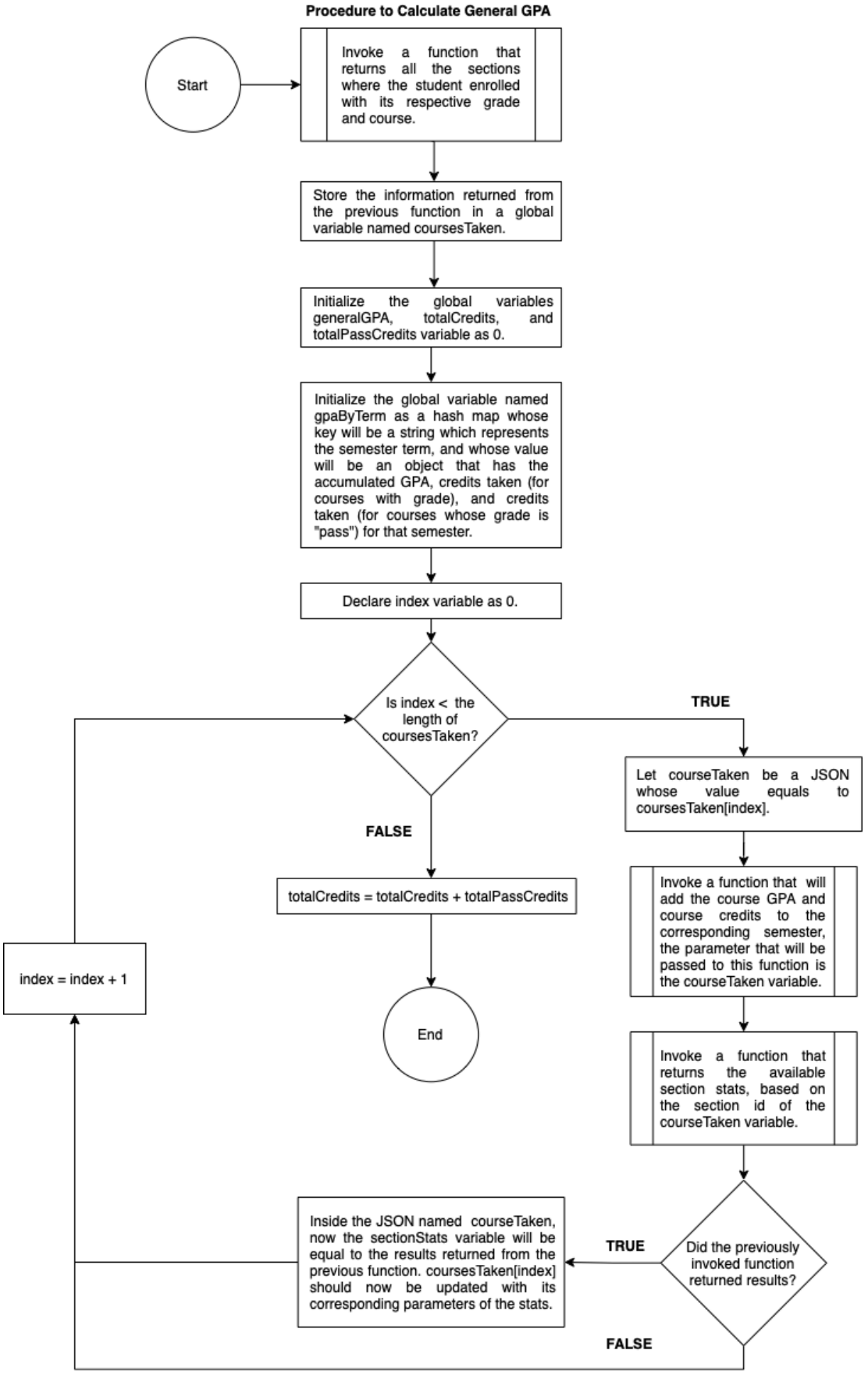
This is a view of the feedback feed for a given course.

Student curriculum



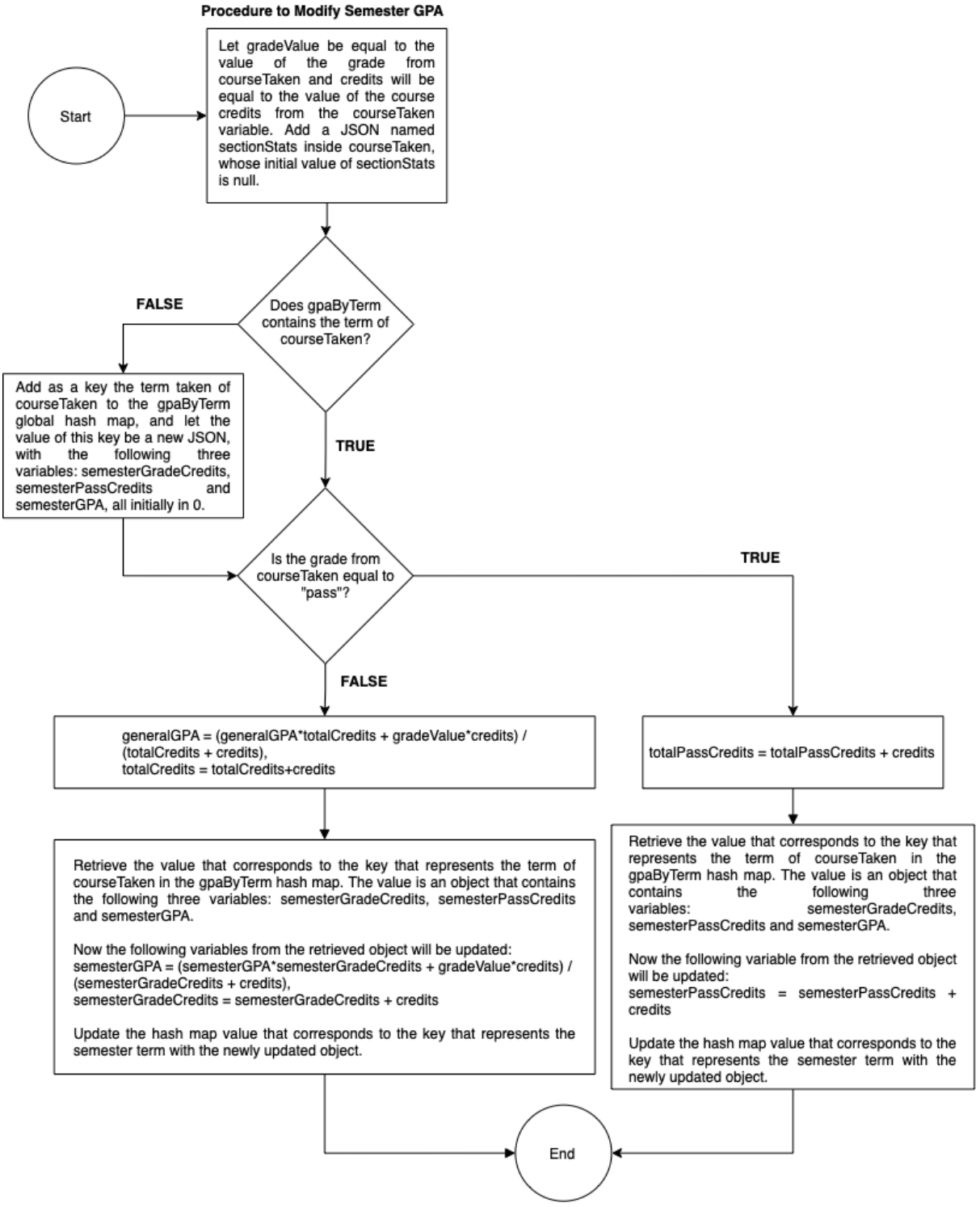
This is a view of the student curriculum page, it allows the student to see where they are academically.

5.5 Describe Flowcharts for algorithms



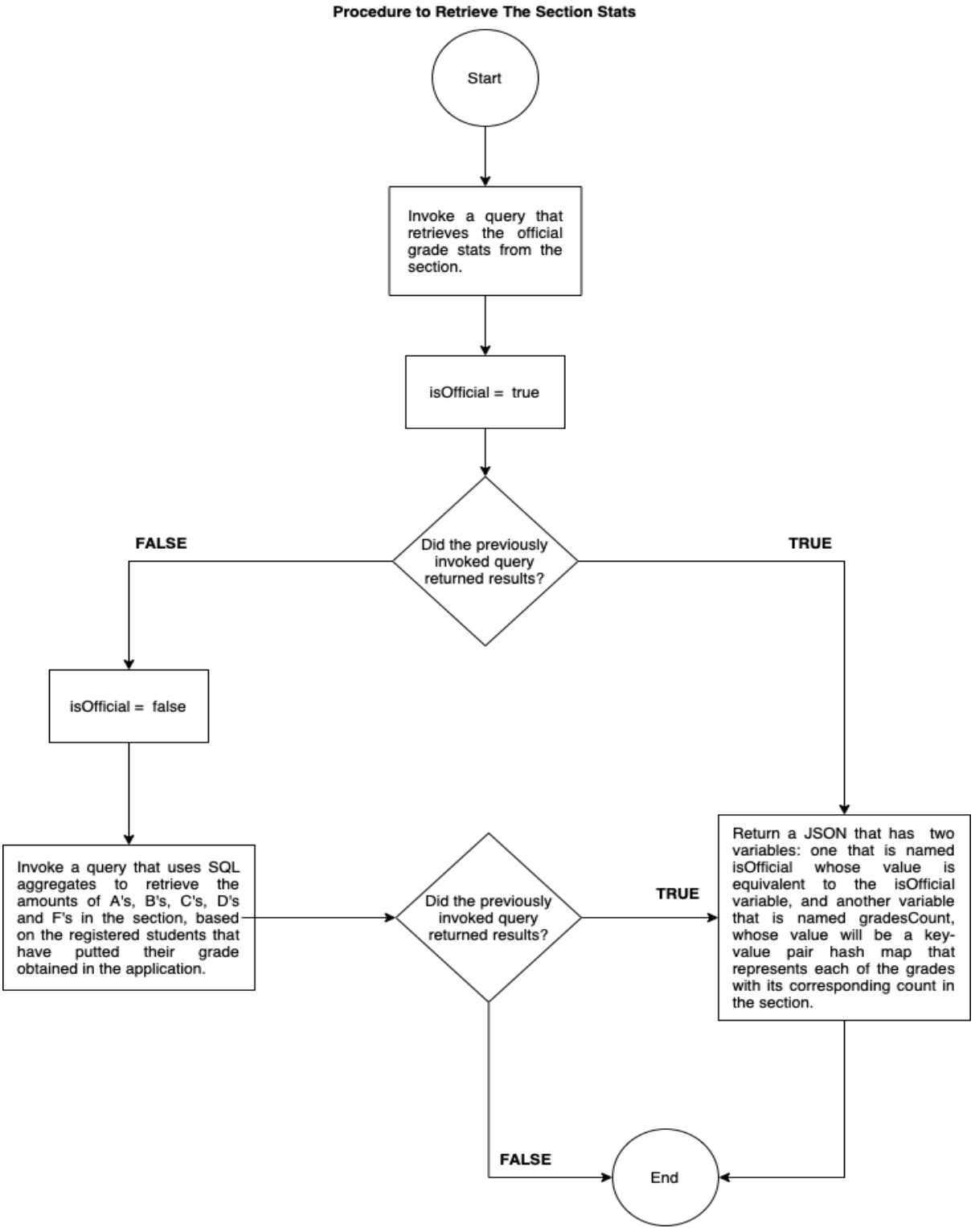
Procedure to Calculate General GPA

The previous flowchart describes the algorithm that takes into consideration all the courses that the student has registered in the system, and will calculate the student general GPA based on the amount of credits and honor points accumulated on all the courses. The courses that instead of having a traditional grade, have a “pass” grade will not be taken in consideration for the GPA calculation. Also, additional functions are invoked inside this procedure to calculate the GPA by semester, and to recollect each of the sections statistics. The functions procedures will be seen in the following two flowcharts.



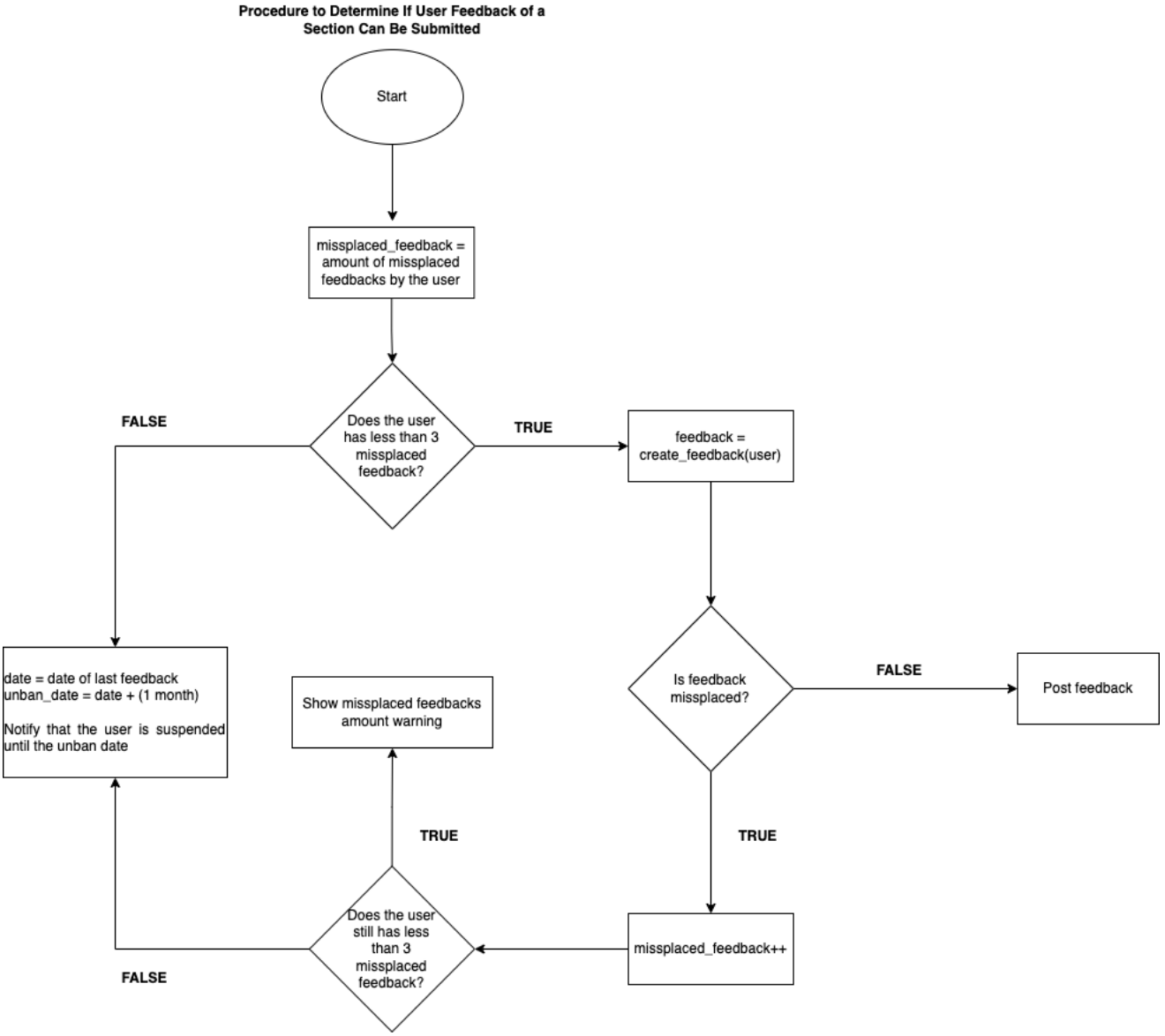
Procedure to Modify Semester GPA

The previous flowchart describes the algorithm that will modify a specific semester GPA, based on the grade obtained on the recently added course for that semester. However, if it detects that the desired course to add to a specific semester will be the first course to be added for that desired semester, then the semester GPA will be initialized as the grade point obtained for that course.



Procedure to RetrieveThe Section Stats

The previous flowchart describes the algorithm that will retrieve stats from a section. It will first see if the official section stats provided by a university is available on the database. If that is not the case, then it will see which registered users on the system have inputted their obtained grade in that specific section and it will take a sampling based on those users. However, if no official stats are available, and the user that is using the application is the only student that registered their grade obtained in that section, then no section stats will be retrieved. Those are the statistics that will be used so that the student can compare their performance with other students.



Procedure to Determine User Feedback of a Section Can Be Submitted

The previous flowchart describes the algorithm that will determine if the recently provided feedback from a user can be published. Basically, if the user has used profanity at least three times, they won't be able to provide feedback until they have passed their 30 day account suspension. However, if that is not the case, then a new feedback will be created and we will verify if the new feedback contains profanity. If not, it will be published, otherwise, they increment the amount of misplaced feedback by one and verify again, did the user now reach three misplaced feedbacks? If not, notify a warning to the user about the fact that their profane comment won't be shown to the public and that their current amount of strikes is the feedback amount, out of three. Otherwise, show that they have been awarded with a 1 month suspension. As for the latest version of the app, this 1 month suspension is not appealable but after hearing the professor's feedback on the demo presentation, for future version we decided that it will be appealable.

5.6 Identify realistic engineering constraints

We can anticipate that one of the constraints that will play a big role on this project would be the data extraction. Multiple universities will be involved, and inside those universities, there are multiple departments, professors and curriculums. Hence, thousands of records need to be added to the database in order for the application to work properly. Multiple additions can be automatized if we parse the strings that correspond to each record from multiple sources, such as universities courses, sections and grade stats. However, we don't know how to write a script that automatizes such tasks. This brings us to another point, which is the fact that the time is considered one of the biggest constraints, given that the project must be delivered by the beginning of may. In order to offset the constraint of time and data extraction, we plan to limit our scope to the UPR system in this project.

Another engineering constraint that we were able to identify is privacy. This application requires data that might be sensitive, such as the grade of the users and their institutional email. Hence, it might be hard to convince some users that we won't use their email for malicious purposes such as spamming, and the fact that we won't be sharing the course section grades they enter on our system-to-be. Emails will be strictly used for account registration and validation purposes. The grades entered will be used solely for the student's own benefit in order to take advantage of the application's features.

Application security is another important constraint given that we, the developers, might have to think about different scenarios where the security of the system-to-be can be at risk. Of course, we can think about different scenarios of how we can improve the application security (e.g. encrypting the source code to avoid unauthorized access and development of a clone app for malicious purposes, secure the backend to avoid malicious attack to the server, etc.) but thinking about all the different scenarios where a hacker can exploit the application is difficult. User passwords will be hashed using SHA-256, but their password can still be guessed if the hackers do some brute force attack. So even by using this hash function, the users can be concerned about this security vulnerability that can have an impact on their privacy.

5.7 Describe engineering standards to be used*

- ISO/IEC/IEEE 24774-2021 - Standard for Systems and software engineering — Life cycle management
  - Aims to provide a process for creating a software project life cycle.
  - Generic activities can be established in a process, such as preparing software system design and establishing designs related to each software system element.
  - Will be used due to the fact that the software is part of a larger system.
- IEEE 1012-2016 - Standard for System, Software, and Hardware Verification and Validation
  - Verification and validation processes are used to determine whether the development products of a given activity conform to the requirements of that activity and whether the product satisfies its intended use and user needs.
  - Given that we want to provide the best possible experience to the customers, our intention is to comply with this standard to make sure that the user is getting the experience they deserve.

*These standards were taken into consideration but not applied in a rigorous manner.

6 Project Plan

6.1 Describe activities and steps that lead to the completion of the project

We will be working using the Agile methodology, given that this will allow us to work on pre-defined issues established by the team in a meeting. We will be using a sprint of two weeks, however, constant communication between all the members will be an emphasis for the project success.

The project is decomposed on multiple steps:

Step 1: Problem Statement and Project Objective

- In this step, we define the problem that the project focuses to solve with the proposed objective.

Step 2: Technical Description

- A system architecture, modules, ER Diagram, Wireframe for UI, Flowcharts for algorithms, engineering constraints and engineering standards will be defined to facilitate the development process of the project.

Step 3: Technical Implementation

- Based on the proposed technical description, the database tables will be defined according to the ER Diagram. After defining the database tables, we will be working in parallel with the front-end and backend, based on the established Wireframe for the UI and the Flowcharts of

the established algorithms. Engineering constraints and standards will be kept in mind during the development. After completing the project, the system architecture will be kept in mind, especially for the deployment process given that we want the application to support load balancing and caching.

6.2 Provide a list of milestones and associated deliverables

Pre-Proposal - 01/25/2022:

- Describe project idea:
  - Motivation
  - Problem statement and challenges
  - Solution approach

Project Proposal - 02/17/2022:

- Contains the following project elements:
  - Introduction
  - Problem statement
  - Project objectives
  - Solution approach
  - Technical description
  - Project plan
  - References

First Project Implementation Progress - 03/10/2022:

- The most essential functionality of the project must be implemented

Progress Report - 03/17/2022:

- Reviews in the proposal must be addressed
- Design elements must be updated
- Preliminary performance must be discussed
- Project must have a web page and code repository

Second Project Implementation Progress - 04/29/2022:

- Bugs from the previous phase must be fixed, minor and major components must already be established in the project
- The project should already be available for public use

Final Report - 05/13/2022:

- Updated design elements
- Performance results

Final Demo - 05/13/2022:

- Fifteen minute demo will be provided via videoconference
- The following items must be available in the project web page:
  - Powerpoint presentation for the demo
  - Elevator pitch video

6.3 Define the specific role of team members

The team members of this project are Félix Dasta, Armando León and Emmanuel Hernandez, and all of them will be working as full stack developers. Each team member will be involved in the whole software development life cycle.

7 Updated Design Elements

During the development of the project, we encountered that with the ER diagram design established in the proposal, there were going to be some bugs. For example, we established before that a course section should have one instructor only and each instructor could teach one or more course sections. However, while extracting data from the historical sections of UPRM, we found out that there were course sections that had many instructors teaching it. Specifically, there were some laboratories that could have as many as three instructors. Given that we want all of the instructors of the section to have feedback, we made this change.

Another scenario we thinked about was the case were some students could have taken some course of the curriculum but under other codification (e.g. some student took ICOM4009 instead of INSO4101, given that they were from the Computer Engineering curriculum before changing to the Computer Science & Engineering or Software Engineering curriculum). So we added a relationship between two courses that could be considered the same, even if they have different codes or belong from a different department. Hence, we refined the ER diagram to reflect those changes that we made on the backend.

Also, each of the reviews of the proposal were addressed.

8 Preliminary Performance Results

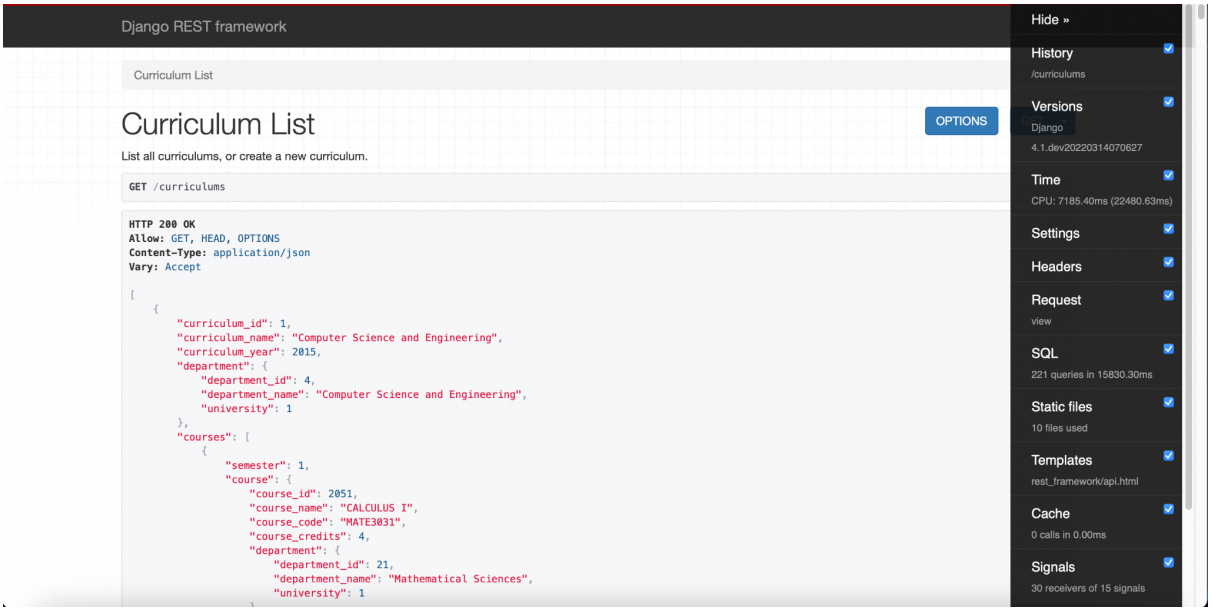
8.1 Retrieving the curriculums and its related entities

Given that there are thousands of records in the database of this project, including historical sections, courses and instructors, performance optimizations were required in order to retrieve a desired record in an acceptable time frame.

Apart from the infrastructure requirements that are required so that the application can run with optimal performance, optimized queries and caching techniques are also necessary in order to get a response as fast as possible.

We decided to use the Django Rest Framework due to the fact that this framework is designed to take advantage of as much hardware as possible, and we can refine the architecture to our desired business logic. Also, Django provides a powerful caching framework, and given that the information that will be retrieved from our database might be repetitive, caching would help a lot in terms of performance. However, given that there is a learning curve for the Django Rest Framework, during this phase we faced multiple performance issues using nested serializers. Before seeing the performance results, we must clarify that the database tables are hosted in Amazon Web Services and the server was being runned on our local machines.

The nested serializers and the performance results can be seen in the screenshot below:



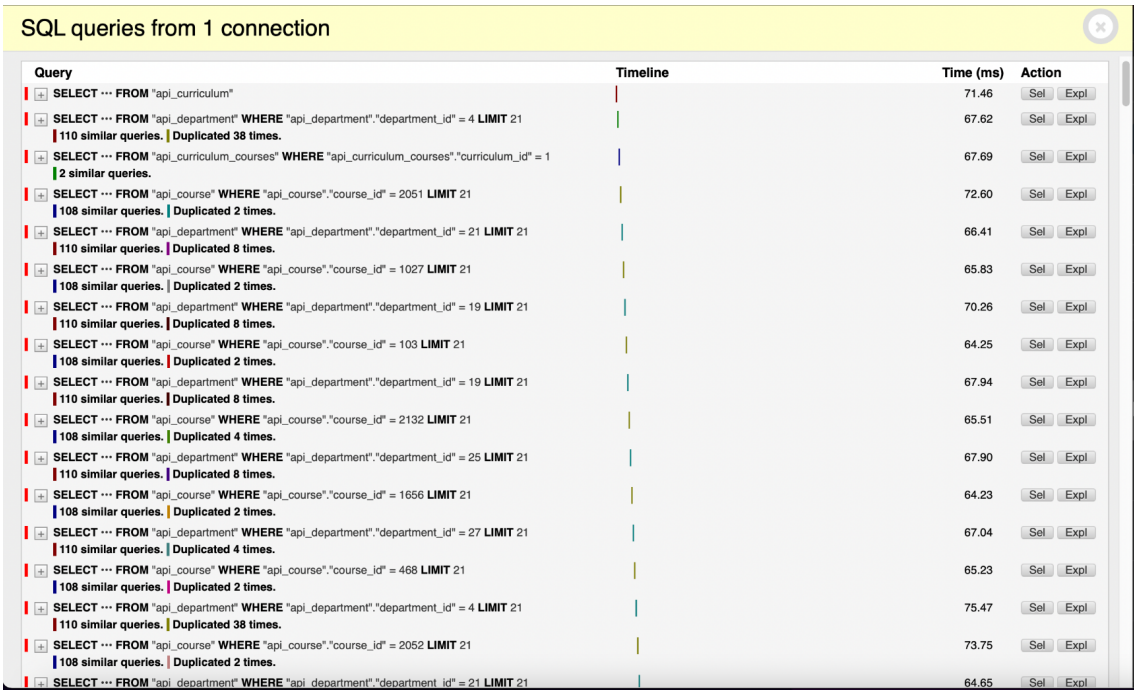
A curriculum nested serializer with poor performance results as seen on the right

As we can see, a single curriculum has multiple courses, and a course obviously belongs to a department, as defined in the ER diagram. But in order to get all of the curriculums with its respective courses and the departments of each course, if we don't tell Django to "prefetch" the related entities, we get the N+1 query issue. In this case, this means that for every curriculum, we're going to invoke a query to get each of the courses, and the department from the course, apart from the department where each curriculum belongs. In total, we got about 221 queries with just two curriculums that have 54 courses each. The overall result of all the queries invoked took an average of 15.83 seconds. The total time consumption can be seen in the right side of the above screenshot.

The 221 queries can be summarized in the following categories:

- 1 query to get all of the curriculums
- 1 query to get the department of the first curriculum
- 1 query to get the courses id's of the first curriculum
- 54 queries to get the each of the courses of the first curriculum
- 54 queries to get the department of the courses from the first curriculum
- 1 query to get the department of the second curriculum
- 1 query to get the courses id's of the second curriculum
- 54 queries to get the each of the courses of the second curriculum
- 54 queries to get the department of the courses from the second curriculum

In the next screenshot, we will see how much time it took to execute some of the separated queries:



Timeline of some of the individual queries that were invoked

The CPU resource consumption used by the previous queries will be summarized in the following table:

Resource usage

Resource	Value
User CPU time	4091.313 msec
System CPU time	3094.084 msec
Total CPU time	7185.397 msec
Elapsed time	22480.631 msec
Context switches	72 voluntary, 23008 involuntary

CPU resource usage without making improvements to the nested serializers

Obviously, this is not good. Clearly, the majority of the queries that were invoked were redundant and unnecessary. Considering the fact that we have only added two curriculums at the moment, it was also worrying. Each query requires a trip from the backend server to the database back and forth, and when repeating the same pattern multiple times, the delay caused was significant. For instance, the first query that was invoked only took about 71.48 ms to retrieve the results, but multiplying that amount of seconds times 221, we get a significant delay of almost 16 seconds. Many Django serializers tutorials didn't talk about this N+1 query issue because the average user doesn't see a huge impact in performance since they don't have too much nested data on the testing environment. Sadly, those issues get more obvious when the projects are already deployed and when the database tables get more populated.

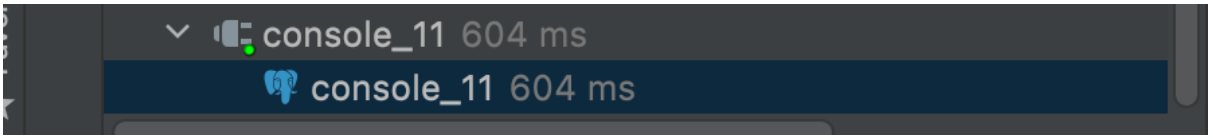
This was the problematic line of code:

```
curriculums = Curriculum.objects.all()
```

At first, the most obvious solution towards this problem was to try to make an inner join between the curriculums, course and department entity using the Django Rest Framework. Hence, our goal was to achieve something similar to this in Django:

```
SELECT * FROM api_curriculum INNER JOIN api_curriculum_courses ON
(api_curriculum_courses.curriculum_id = api_curriculum.curriculum_id)
INNER JOIN api_course ON
(api_curriculum_courses.course_id = api_course.course_id)
INNER JOIN api_department ON
(api_department.department_id = api_course.department_id)
```

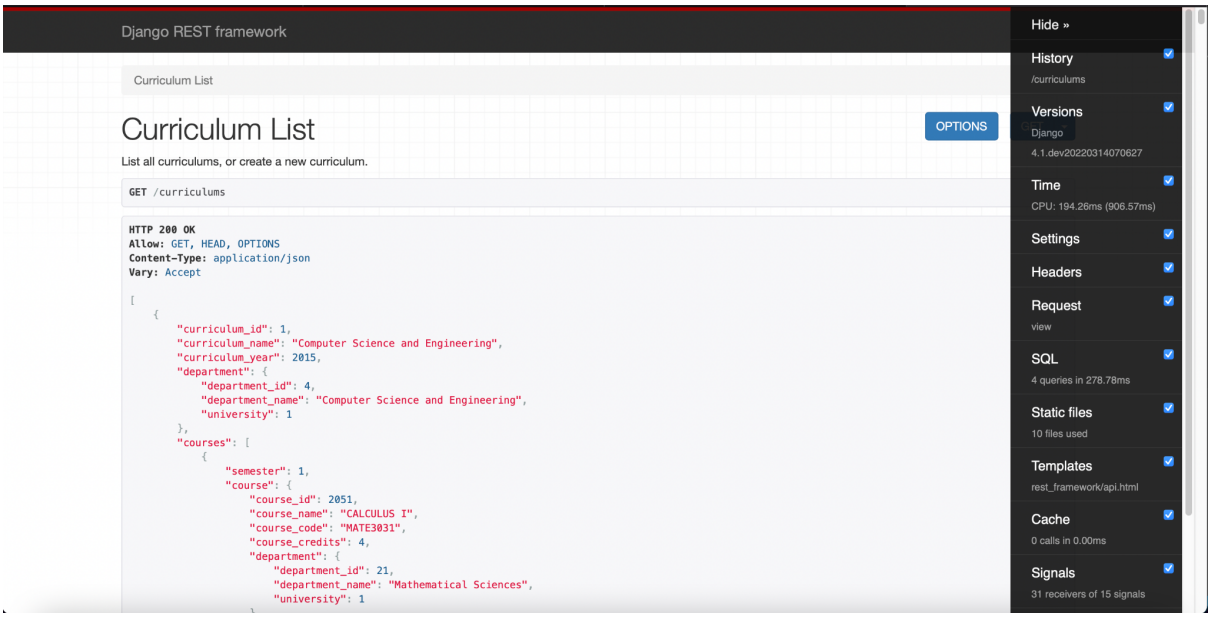
On multiple tries, the execution of this query only took between 600 ms to 700 ms. That's an improvement of over 96% compared to the previous result!



Time execution when invoking an inner join between the three tables to get all the curriculums with its courses and departments

However, instead of Django executing the previously presented query, by using *prefetch_related*, Django gets all of the items of each related object in a single query and then, the related items will be joined using Python instead of PostgreSQL.

Surprisingly, by using this technique, the performance was improved even further as seen in the screenshot below:

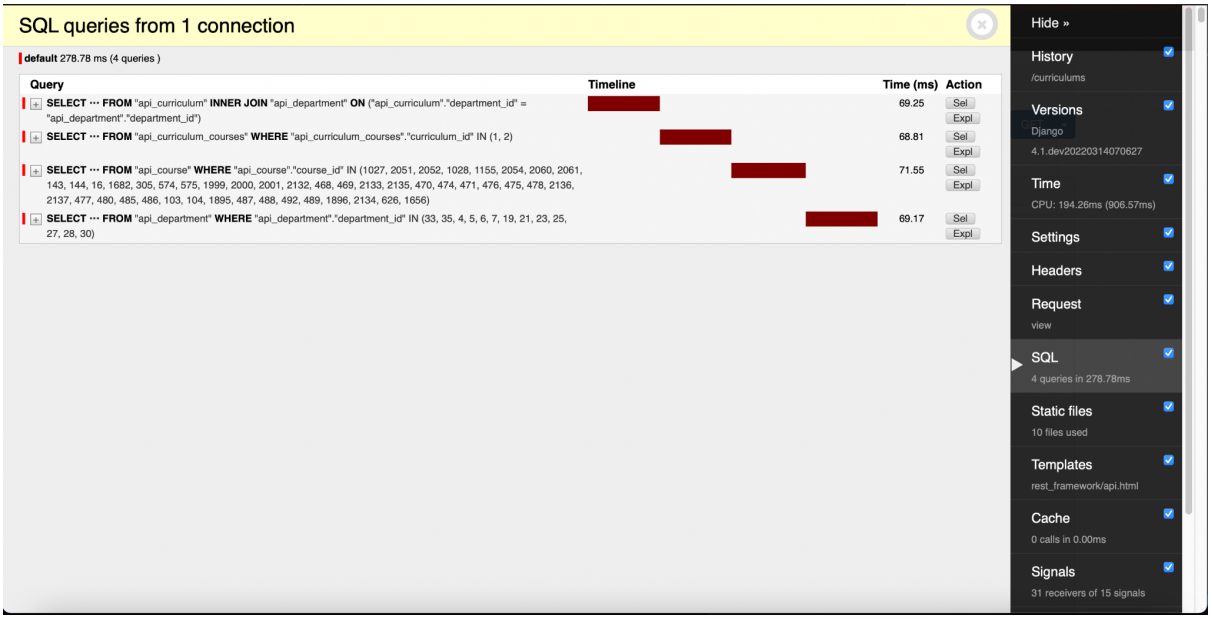


A curriculum nested serializer with great performance results as seen on the right

Observe how the amount of queries were reduced from 221 to just 4 queries. Not only that, but the overall execution time of the 4 queries was just 278.78 ms.

That's an improvement of over 98% compared to the time execution seen in the first screenshot!

In the next screenshot, we will see how much time it took to execute the 4 queries:



Timeline of some of the individual queries that were invoked

The CPU resource usage with the improved performance will be summarized in the following table:

Resource usage

Resource	Value
User CPU time	142.624 msec
System CPU time	51.640 msec
Total CPU time	194.264 msec
Elapsed time	906.565 msec
Context switches	2 voluntary, 45 involuntary

CPU resource usage by using select_related and prefetch_related

Despite the fact that each of the 4 queries extracts much more information than the redundant queries previously seen, the reality is that the execution time of each query is the same as the ones seen in the first example, which extracts less information. In this scenario, each query represents about 25% of the execution time. Also, in terms of CPU resource usage, here's a summary of how each category was improved by using the second approach:

- **User CPU time:** 96.5% less
- **System CPU time:** 98.33% less
- **Total CPU time:** 97.3% less

This is the line of code that was executed to see the performance improvement:

```
curriculums =
Curriculum.objects
.prefetch_related('curriculum_courses_set__course__department')
.select_related('department')
```

Essentially, by using *prefetch_related*, we're telling Django to prefetch the related courses that are in the many to many relationship table between the curriculum and the courses, and we also want to extract the department that corresponds to each course. By using *select_related*, we're telling Django to do an inner join between the curriculum and department entities, so that we can see the departments of each of the curriculums. Hence, *prefetch_related* is used to join the entities that have a many to many relationship or the reverse relationship (one to many relationship). *Select_related* is used to join the entities that have a many to one relationship with respect to another entity. That is, for each table that has a foreign key to another table, *select_related* is the ideal choice to make an inner join in Django.

8.2 Retrieving the sections and its related entities

Currently, we have about 33,657 sections registered in our database. The relevant related entities that we want to fetch from the section are the instructors of each section, the department of each instructor, the course that corresponds to the specific section and the department of the course. When trying to retrieve all of the sections registered in our database using the GET method on the REST API, after waiting a few minutes, we still didn't get any result. In fact, it was probably invoking some queries because this method faced the N+1 query issue as well. Given that we must fetch the related entities of the section, we can only imagine how overloaded the system should be in queries when facing this issue. We would get over 150,000 queries, which would translate to a wait time of over three hours if we assume that the average retrieval for each query is about 71 ms. So solving the N+1 query issue on the curriculums first was crucial, because that was going to help us on fixing the N+1 query issue on the sections as well.

However, given that the sections have too many records, solving the N+1 query issue wasn't enough to solve the unloadable records from the REST API. We had to implement a technique named pagination. Essentially, by using pagination, we request only a small subset of the available data from the server. So in our case, out of the 33,657 sections available, we decided to only retrieve about 50 sections at once.

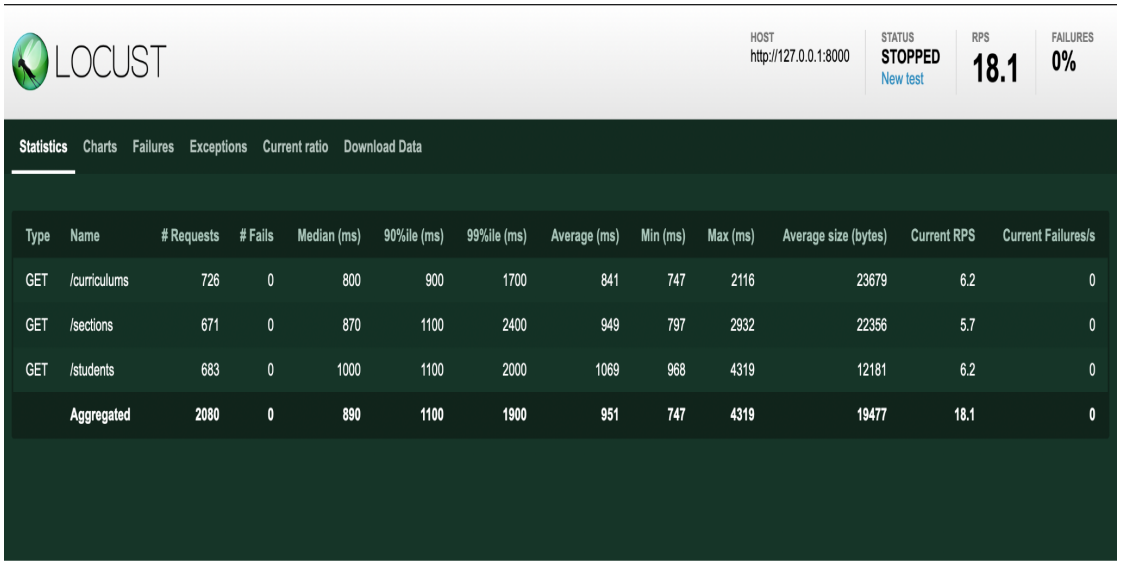
Using the pagination technique, it gave us two advantages:

1) **Improved performance:** Given that the client doesn't need to load all of the required data and the server doesn't need to send all of the data available at the moment, the performance will see an increased boost. Less memory and CPU will be used on both, the server as well as the client.

2) **Increased readability:** Like a book, we prefer each book to be divided by pages rather than having a very long piece of paper. By doing this, we can more easily recall where we last left off, instead of having to parse through a single, long paper.

8.3 Preliminary load testing results

A series of load tests were performed on a few of the endpoints the team has defined. The *Student, Curriculum* and *Section* endpoints were subjected to three different rounds of testing on their *GET* methods. The first round consisted of a simulation of 50 concurrent users. The second round consisted of a simulation of 150 concurrent users. The third round consisted of a simulation of 300 concurrent users. These tests were performed using *Locust*.

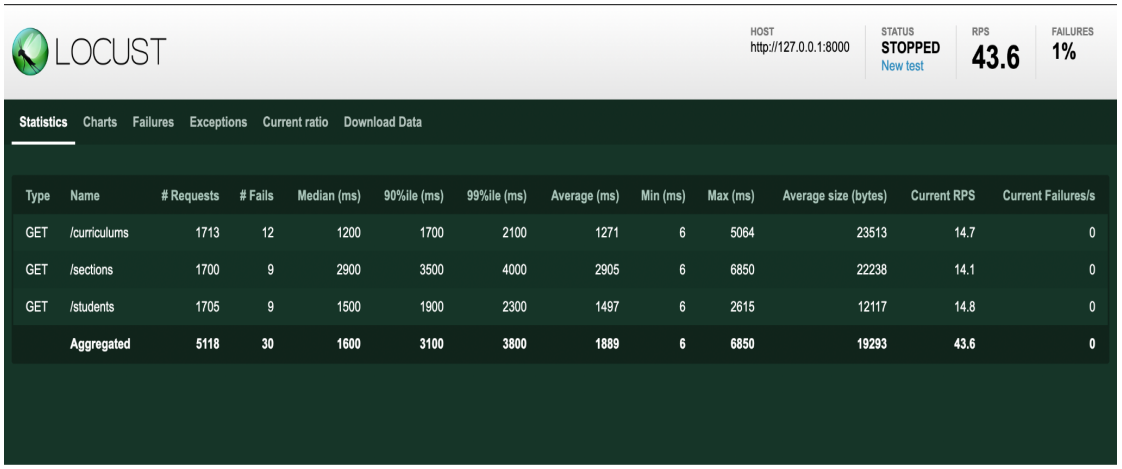


Type	Name	# Requests	# Fails	Median (ms)	90%ile (ms)	99%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s
GET	/curriculums	726	0	800	900	1700	841	747	2116	23679	6.2	0
GET	/sections	671	0	870	1100	2400	949	797	2932	22356	5.7	0
GET	/students	683	0	1000	1100	2000	1069	968	4319	12181	6.2	0
	Aggregated	**2080**	**0**	**890**	**1100**	**1900**	**951**	**747**	**4319**	**19477**	**18.1**	**0**

First test (50 Users)

The results for the first test can be seen above. This load test consisted of 50 simulated users concurrently accessing the endpoints at a loading rate of 5 users per second. The test was allowed to run until a total of approximately 2000 requests was reached. The request rate per second was approximately 18.1 *RPS*. During this test, no failure was encountered.

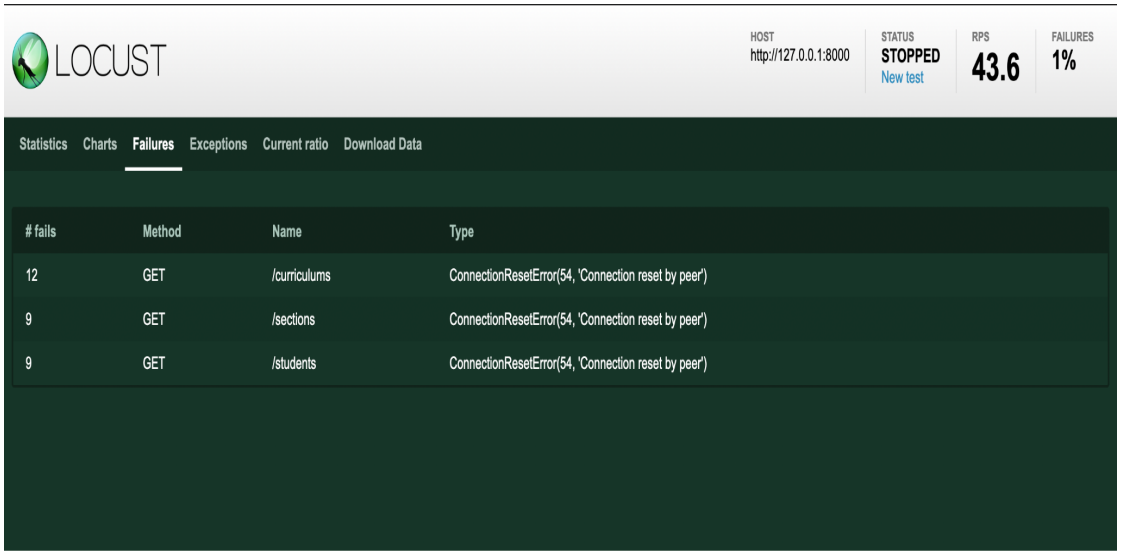
The above table shows a graph of requests in relation to failures.



Type	Name	# Requests	# Fails	Median (ms)	90%ile (ms)	99%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s
GET	/curriculums	1713	12	1200	1700	2100	1271	6	5064	23513	14.7	0
GET	/sections	1700	9	2900	3500	4000	2905	6	6850	22238	14.1	0
GET	/students	1705	9	1500	1900	2300	1497	6	2615	12117	14.8	0
	Aggregated	**5118**	**30**	**1600**	**3100**	**3800**	**1889**	**6**	**6850**	**19293**	**43.6**	**0**

Second test (150 Users)

The results for the second test can be seen above. This load test consisted of 150 simulated users concurrently accessing the endpoints at a loading rate of 15 users per second. The test was allowed to run until a total of approximately 5000 requests was reached. The request rate per second was approximately 43.6 RPS. During this test, a failure rate of 1% was reached, with a total of 30 failed requests amongst the total of 5118.

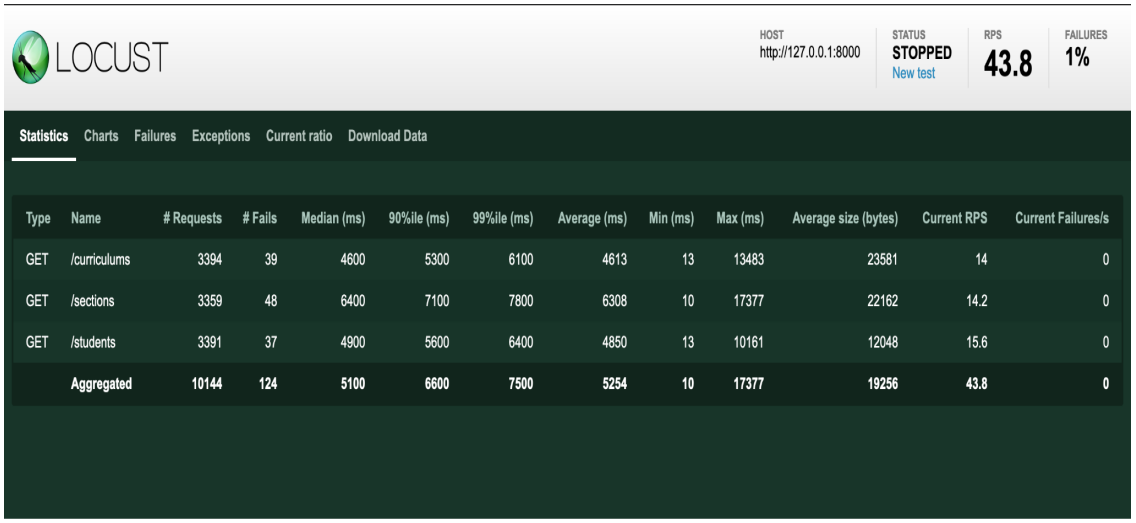


# fails	Method	Name	Type
12	GET	/curriculums	ConnectionResetError(54, 'Connection reset by peer')
9	GET	/sections	ConnectionResetError(54, 'Connection reset by peer')
9	GET	/students	ConnectionResetError(54, 'Connection reset by peer')

Second test failure summary

The above image shows a summary of the failures encountered in the load test. All failures were of type *ConnectionResetError*. This type of error is caused by a possible overload of the server or by lack of resources on the server side. It is worth noting that all 30 failed requests occurred during the first 1000 requests.

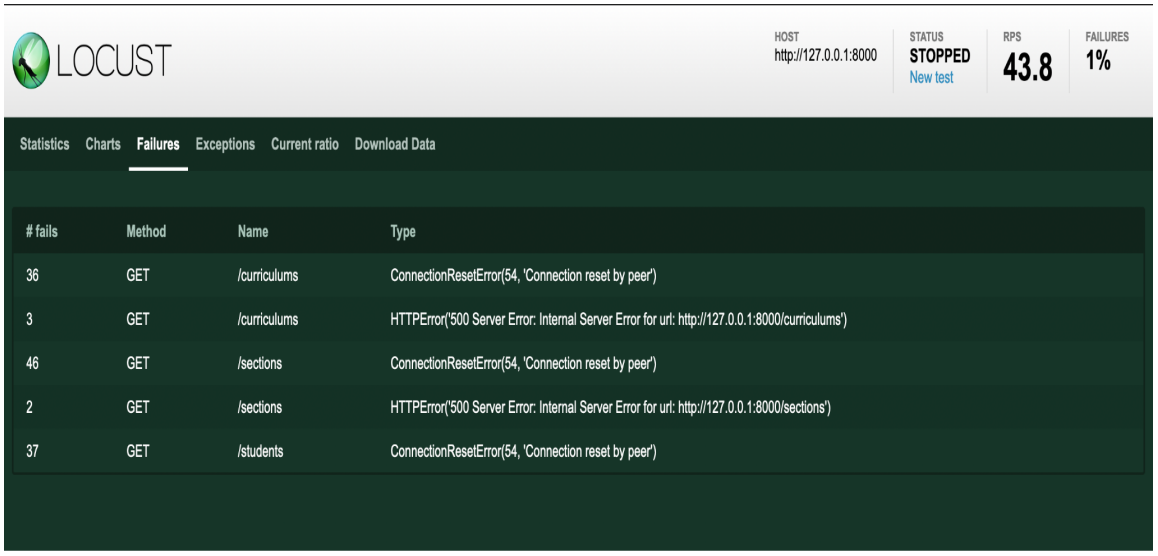
The above table shows a graph of requests in relation to failures



Type	Name	# Requests	# Fails	Median (ms)	90%ile (ms)	99%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s
GET	/curriculums	3394	39	4600	5300	6100	4613	13	13483	23581	14	0
GET	/sections	3359	48	6400	7100	7800	6308	10	17377	22162	14.2	0
GET	/students	3391	37	4900	5600	6400	4850	13	10161	12048	15.6	0
	Aggregated	**10144**	**124**	**5100**	**6600**	**7500**	**5254**	**10**	**17377**	**19256**	**43.8**	**0**

Third test (300 Users)

The results for the third test can be seen above. This load test consisted of 300 simulated users concurrently accessing the endpoints at a rate of 30 users per second. The test was allowed to run until a total of approximately 10000 requests was reached. The request rate per second was approximately 43.8 RPS. During this test, a failure rate of 1% was reached, with a total of 124 failed requests amongst the total of 10144.



Third test failure summary

The above image shows a summary of the failures encountered in the load test. The majority of errors were of type *ConnectionResetError,* which accounted for 119 of the 124 total failed requests. This type of error is caused by a possible overload of the server or by lack of resources on the server side. The other 5 errors were of type *HTTPError* in which 3 of them were from the *Curriculums* endpoint and the other 2 from the *Sections* endpoint.

The above table shows a graph of requests in relation to failures

8.4 Next steps to take to improve performance

In order to improve the performance of the web application, we plan to implement materialized views in our database. Given that there are many repetitive queries, creating a database object to store the results of those queries will be a good idea. By using a materialized view, rather than having to execute the SQL statement itself, we're going to get the precomputed results of the query directly from the materialized view table.

Apart from creating a materialized view, we plan to cache the repetitive results on the backend server, using Django Rest Framework caching techniques. By combining both of these techniques, we hope to see even better performance. The reason why is because the only trip required to the database would be to retrieve the results from the materialized view table. In many cases, this trip won't be required if the data is cached on the backend server. However, there are some things to take into consideration. One of them is that if we don't see great performance improvement from the combination of both techniques (using materialized views + Django Rest Framework caching techniques), we might consider giving up one of the two techniques. In that case, we will stick with the technique that gives us faster results. The reason why we might consider giving up one of the two techniques if the performance improvement isn't too much is due to space complexity. By using both, we're occupying more space on the database tables and on the backend server. So the performance improvement must be significant enough to justify having both techniques.

We will be using these caching techniques on data retrieval that might be repetitive, such as the curriculums, courses, sections and instructors.

9 Project Repository and Website

Project repository: https://github.com/felixdasta/MyUniPerformance
Project website: https://myuniperformance.herokuapp.com/

10 Final Report Addendum

10.1 Automated Data Acquisition

Due to the data driven nature of this project, it is no surprise that large amounts of data are needed to maintain the up-to-date functionality. This raises the issue of having to plan the yearly acquisition of new data released by *OPIMI*. This yearly refresh process would be very time consuming, if done manually,, but there exists libraries that would facilitate the automation of this process. A possible solution to this problem would be to utilize a Python web scraping script that would be executed by a yearly scheduled batch job, upon release of new data by *OPIMI*. This script would download all the necessary .csv's from the *OPIMI* website. Once the data has been acquisitioned in .csv format, it would then enter the next step of inserting the bulk data into the projects database tables. This step could be done with Python's *Bulk Insert* utility, which is a function that inserts data from a .csv file

into a database table. Automating this process would allow us to extend the lifespan of the application beyond the scope that the team might be able to maintain it.

10.2 User Data Privacy Concerns

To help secure the user information, currently the only strategy implemented is

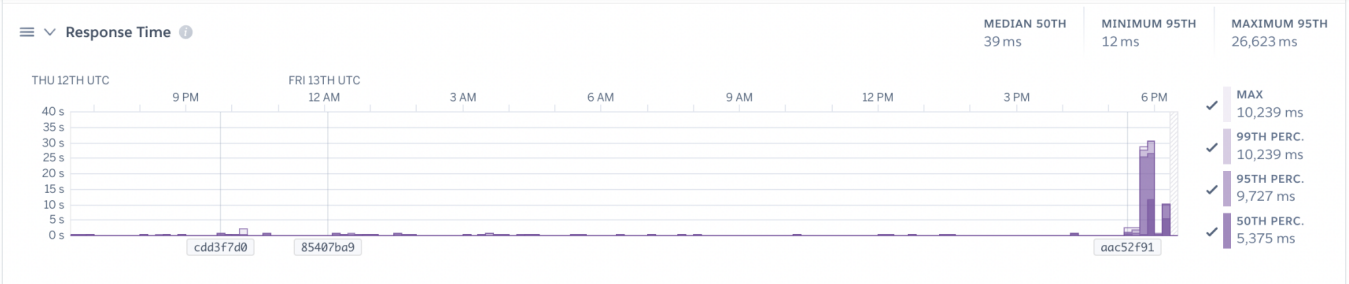
the encryption of the user password which is secured with the SHA-256 algorithm. However, a good short term strategy to strengthen our system would be to use an extension for our PostgreSQL database called “PostgreSQL Anonymizer". This would let us anonymize the user data by defining a schema-level strategy or using functions to target identifiable user data. This anonymization can be achieved with techniques such as pseudonymization, or dynamic masking and then validate the results by evaluating the k-anonimity factor from the queried data and data dumps. The k-anonimity factor can also be calculated with this extension and will let us know how secure our user data will be.

10.3 Final Performance Results

10.3.1 Addressing Performance Results

Regarding past query results which were included in the *Progress Report* phase of the document, the reason why we had concluded that a query response time of ~300ms was not fast enough is due to the fact that it would not be a seemingly instantaneous query execution. Query response time guidelines agree that by having a query execution time of less than 0.1s or 100ms, you can give the user an experience of instantaneous system reaction. Although a query execution time of ~300ms is by no means bad, we wanted to reduce this response time as much as possible. We were able to achieve this, and the solution is further explained below.

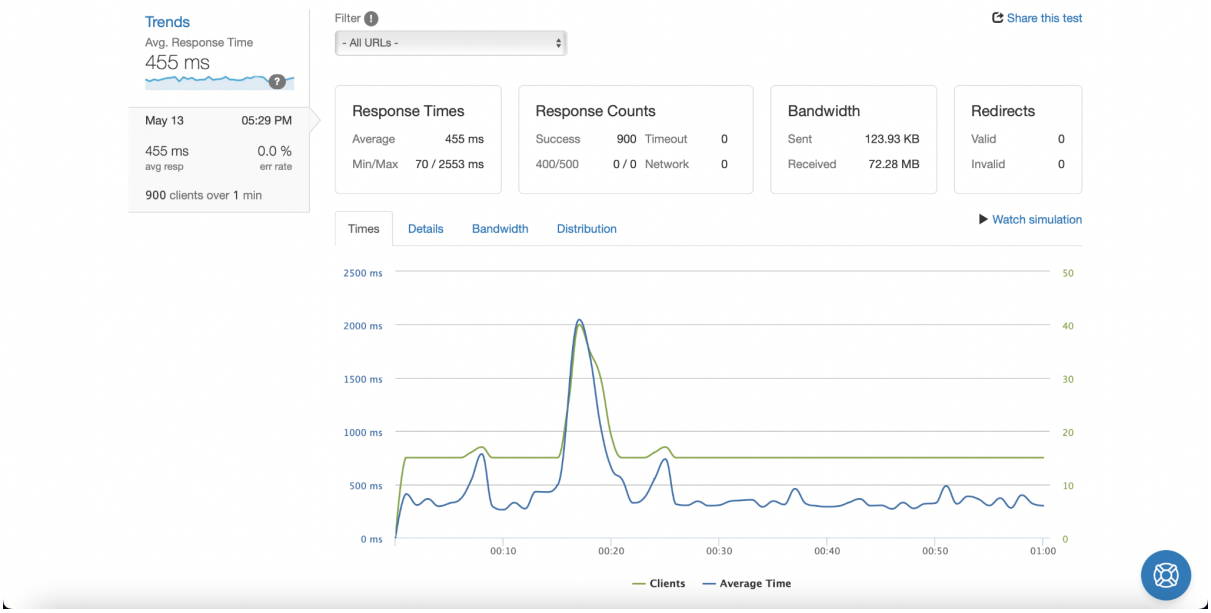
By installing the server on Heroku, we were able to minimize the backend server response time. After further investigation, we discovered that because the database was hosted on the mainland of the United States, there was a substantial delay when running queries on our local devices. The latency between the database and the server was significantly lower because the deployed backend server on heroku was situated in the United States. The median, minimum, and maximum latency between the heroku server and the database server are shown in the diagram below.



10.3.2 Load Testing Result

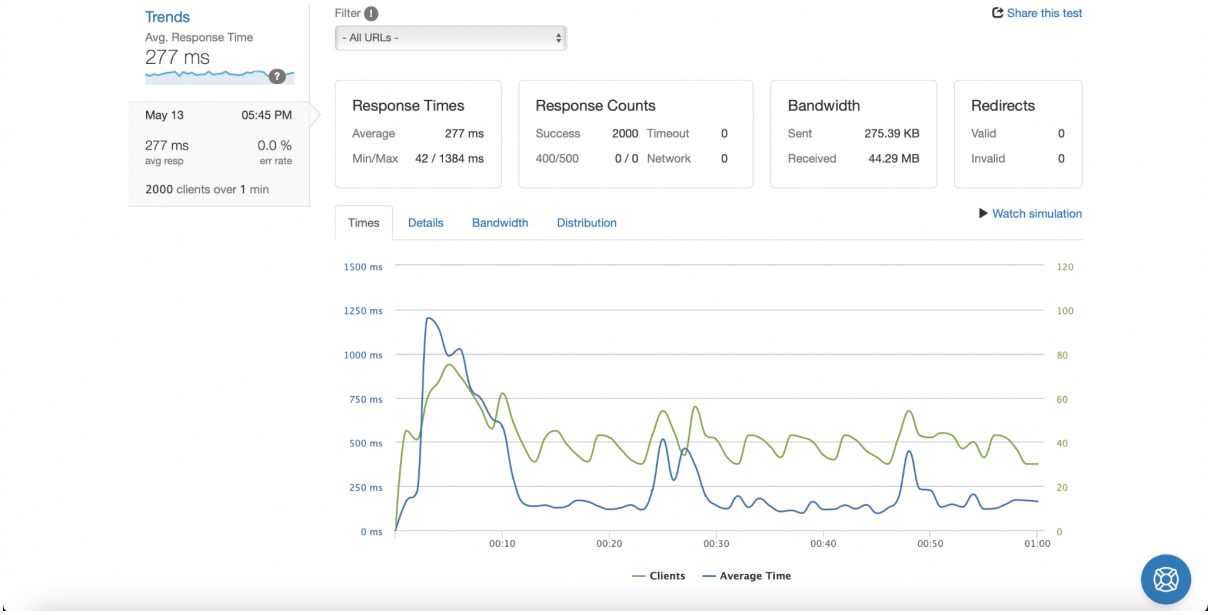
a) Students Load Testing

Assuming a load of 900 clients per second, our heroku server was capable of returning the responses to the client with a response time of 455 milliseconds with a rate of 0.0%, which falls under the category of an average server response time. Considering the fact that we're using the Free Heroku Server, that seems to be good because right now, the only University that we support on the system is the University of Puerto Rico. Students is one of the routes that returns the greatest amount of content so making a load test on this api route made sense. On the bottom, we'll be showcasing the results:



b) Sections Load Testing

Sections is another commonly used api route on our backend server. However, its response isn't as heavily loaded as the student's api response. Hence the load testing of sections was capable of withstanding 2000 clients per minutes with a lower average response time, as seen on the following figure:

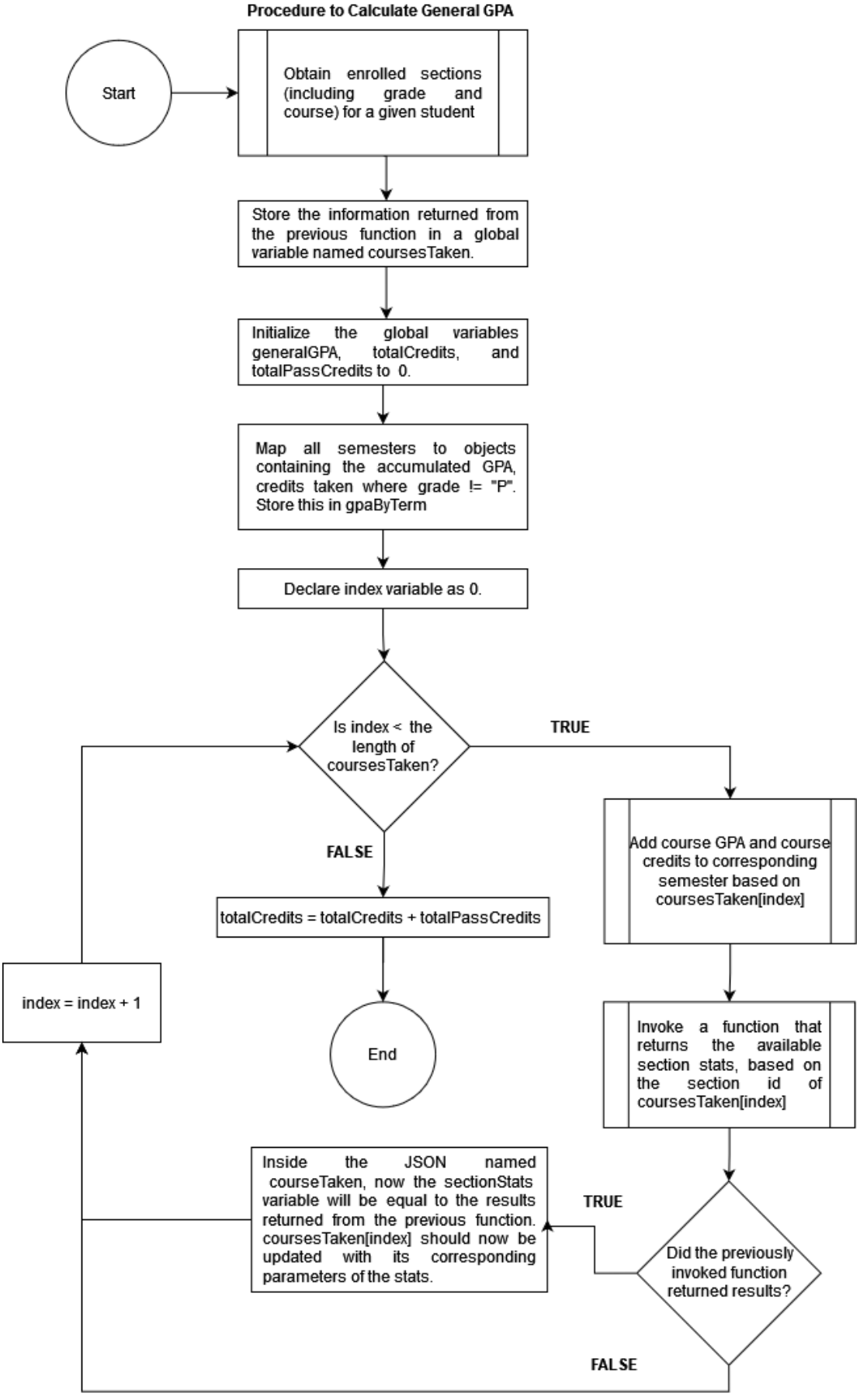


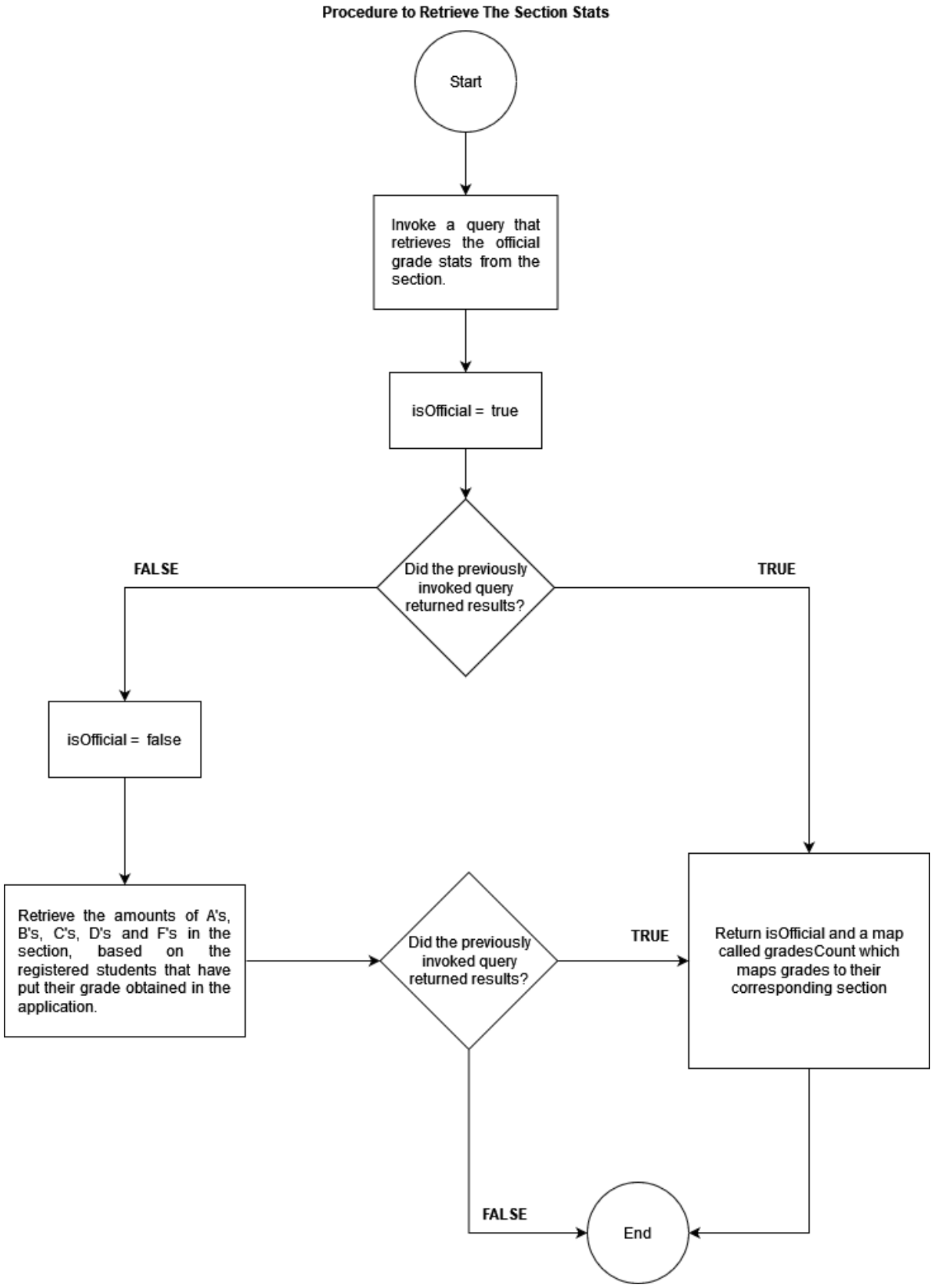
Due to the nature of the application, where the student api is actually more used than the sections api, it is safe to assume that our server should be able to handle 900 clients per minute with what is considered as an average response time by today's standards. However, this is an improvement when compared to the local results. Given that the server response time is less on the deployed backend, that implies that the deployed server uses less CPU time than the local server, so this factor might be the one that is helping the deployed backend to hold more concurrent users, even on the free tier plan of Heroku.

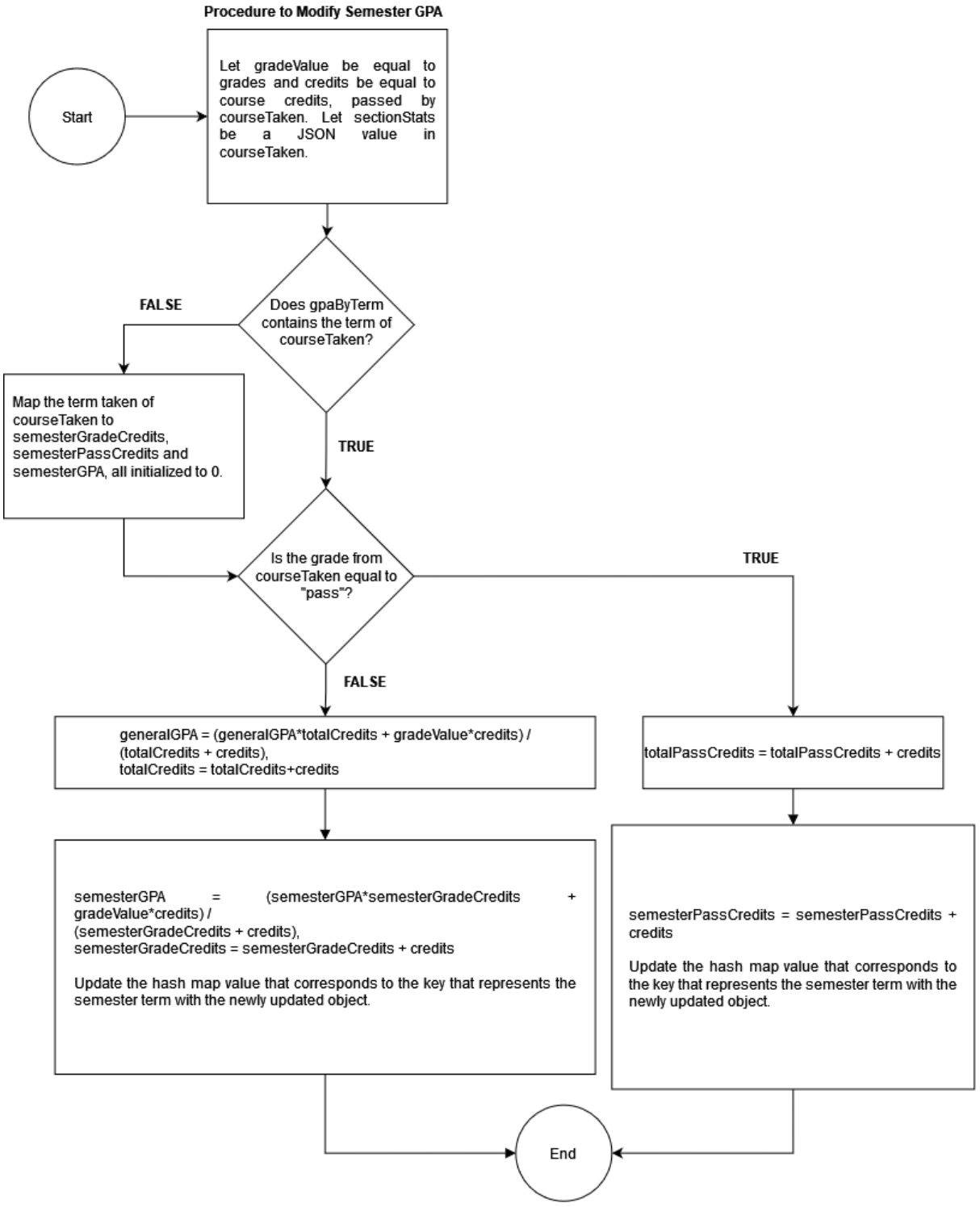
10.4 Future Considerations

Due to the nature of language, there exists cases where a person's legitimate name might have a profane meaning in another language. Because of this,, the use of a profanity filter might incorrectly deliver penalties to users who have not committed any fault. The existence of false positive bans calls for the implementation of an appeal system. Appeal systems are commonly used in platforms where users might receive a penalty, and including one in our project would definitely help improve the user experience. Since the feedback and recommendation for implementing such a system was received during the final presentation, it will not be feasible to implement said system due to time constraints, but the team agreed that, if the app were to be further maintained and developed, it would be of utmost importance to implement. By allowing users to appeal a penalty they believe is unfair, an environment of harmony can be further conserved.

10.5 Updated Flowcharts




Procedure to Retrieve The Section Stats
Start
Invoke a query that retrieves the official grade stats from the section.
isOfficial = true
Did the previously invoked query returned results?
FALSE
TRUE
isOfficial = false
Retrieve the amounts of A's, B's, C's, D's and F's in the section, based on the registered students that have put their grade obtained in the application.
Did the previously invoked query returned results?
TRUE
Return isOfficial and a map called gradesCount which maps grades to their corresponding section
FALSE
End


Procedure to Modify Semester GPA
Start
Let gradeValue be equal to grades and credits be equal to course credits, passed by courseTaken. Let sectionStats be a JSON value in courseTaken.
Does gpaByTerm contains the term of courseTaken?
FALSE
TRUE
Map the term taken of courseTaken to semesterGradeCredits, semesterPassCredits and semesterGPA, all initialized to 0.
Is the grade from courseTaken equal to "pass"?
TRUE
FALSE
generalGPA = (generalGPA*totalCredits + gradeValue*credits) / (totalCredits + credits),
totalCredits = totalCredits+credits
totalPassCredits = totalPassCredits + credits
semesterGPA = (semesterGPA*semesterGradeCredits + gradeValue*credits) / (semesterGradeCredits + credits),
semesterGradeCredits = semesterGradeCredits + credits
Update the hash map value that corresponds to the key that represents the semester term with the newly updated object.
semesterPassCredits = semesterPassCredits + credits
Update the hash map value that corresponds to the key that represents the semester term with the newly updated object.
End

11 References

1. Managers.org.uk. 2022. *Setting smart objectives - chartered management institute. Chartered Management Institute*. [online] Available at: <https://www.managers.org.uk/wp-content/uploads/2020/03/CHK-231-Setting_Smart_Objectives.pdf> [Accessed 10 February 2022].Managers.org.uk. 2022. *Setting smart objectives - chartered management institute. Chartered Management Institute*. [online] Available at: <https://www.managers.org.uk/wp-content/uploads/2020/03/CHK-231-Setting_Smart_Objectives.pdf> [Accessed 10 February 2022].

2. En.wikipedia.org. n.d. *IEEE Standards Association - Wikipedia*. [online] Available at: <https://en.wikipedia.org/wiki/IEEE_Standards_Association#Standards> [Accessed 15 February 2022].

3. Ieeexplore.ieee.org. 2017. *1012-2016 - IEEE Standard for System, Software, and Hardware Verification and Validation - Redline*. [online] Available at: <https://ieeexplore.ieee.org/document/8356723> [Accessed 15 February 2022].

4. Cdn.standards.iteh.ai. 2021. *iTeh Standards*. [online] Available at: <https://cdn.standards.iteh.ai/samples/78981/5c068220670742f98a48d5cd7ad4d050/ISO-IEC-IEEE-24774-2021.pdf> [Accessed 15 February 2022].

5. YellowAnt. 2022. *ORM: Rethinking Data as Objects*. [online] Available at: <https://blog.yellowant.com/orm-rethinking-data-as-objects-8ddaa43b1410> [Accessed 16 February 2022].

6. Amazon Web Services, Inc. 2022. *What is Caching and How it Works | AWS*. [online] Available at: <https://aws.amazon.com/caching/> [Accessed 17 February 2022].

7. GeeksforGeeks. 2022. *Prefetch_related and select_related functions in django - GeeksforGeeks*. [online] Available at: <https://www.geeksforgeeks.org/prefetch_related-and-select_related-functions-in-django/#:~:text=In%20Django%2C%20select_related%20and%20prefetch_related,and%20make%20program%20much%20faster.> [Accessed 16 March 2022].

8. Kachot, D., 2022. *Top 8 Best Practices to Develop Secure Mobile Apps*. [online] Clariontech.com. Available at: <https://www.clariontech.com/blog/top-8-best-practices-to-develop-secure-mobile-apps> [Accessed 17 March 2022].

9. Loader.io. 2022. *Application Load Testing Tools for API Endpoints with loader.io*. [online] Available at: <https://loader.io/> [Accessed 14 May 2022].