İSTANBUL TECHNICAL UNIVERSITY

Faculty of Computer Science and Informatics

Sistroid
PRESENTS THE PROJECT PLAN OF THE

SISTROID

The Student Information System

Prof: Assoc. Prof. Dr. Ahmet Cüneyd Tantuğ

Sarp Özdemir 150180207
Talha Ünsel 150170206
Ata Barış Akın 150160215

06 June- 20 June/ 2021

Table of Contents

1. Introduction ……..…… 2

1.1 Purpose .....……. 2

1.2 Scope ...……. 2

2. Data Model ……..……..……..…….. 2

2.1 Stored Data …….. 2

2.2 Functional Dependencies ……..……..……. 3

2.3 Normalization Steps ……. 4

2.4 ER-Diagram …… .. 7

3. Application Details ……..…….. 8

3.1 Project Architecture …….... 8

3.2 Project Documentation …… 9

1 Introduction

1.1 Purpose

The sistroid web app aims to provide a student information system which is robust, easy to use and nice to look at. The Sistroid web app uses react in the front end, flask in the back end and mysql as server. Sistroid provides novel new features such as gpa mentor which provides the students with data about the grades they should achieve in order to reach their desired gpa. Unlike other student information systems Sistroid has more focus put into making the user experience smoother by providing easier to navigate ui coupled with a more modern design choice.

1.2 Scope

Sistroid provides a student information system with the features of students being able to add and drop their classes, view their standing and information, calculate the grades they should achieve to reach a certain GPA and be able to check histograms of past classes, topics, and teachers. Sistroid has an included authorization system which can distinguish between students and teachers but having higher privileges is not in the scope of this project.

2. Data Model

2.1 Stored Data:

For students we keep data such as phone number, mail address as their personal information together with data such as major and password which is required for the system to function correctly together with a person_id which is used to identify the person. The password is kept in sha256 hashed format in case of data leaks. For the testing data this is the sha256 hashed value of the string “password”. We keep a created_at field in all of the tables for logging purposes but other than that it does not serve any practical purpose other than some semester inference tricks for classes.

For the semesters we keep the starting and ending dates to determine its span. For class topics we keep their descriptions, the letters they cover and the major requirements they have. Separate from the topic for the classes themselves we also have data for the teacher id, the time slots and classroom posts which themselves consist of sender id crn and content. The time slots only consist of a single start date and a single end date. We also keep names and descriptions of universities together with the faculties in those universities and the majors in those faculties.

2.2 Functional Dependencies

Values of Name, Surname, photo_url, address, phone and major are functionally dependent on the person. People have the primary key of “person_id” in the normalized form of the database so we can say that those values are functionally dependent on person_id.

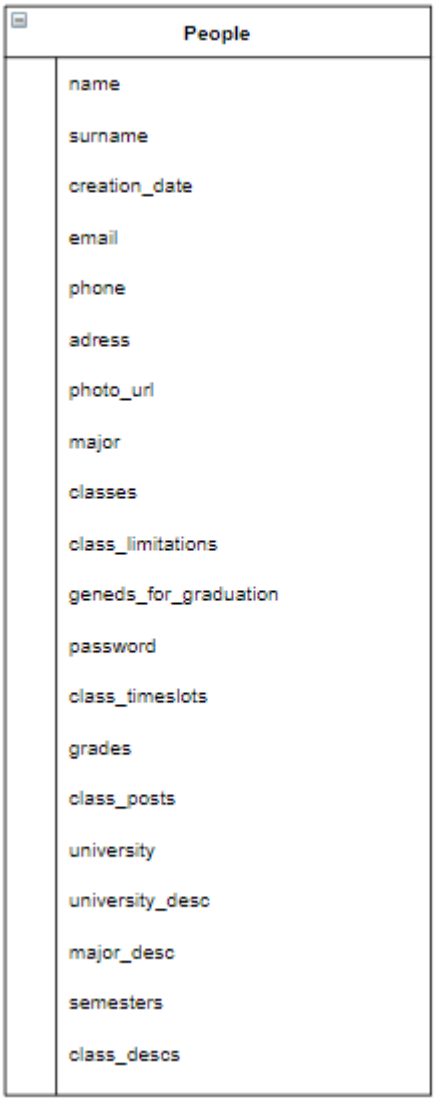
Class_limitations class descriptions and letters are dependent on the class_name. Ex: ISE316 class provides letter H for humanities and has major restriction ISE with description “computer ethics”.

Time slots are functionally dependent on the class crn’s. Also class descriptions, class name and teacher id’s depend on the class crn. The semester however does not functionally depend on the crn but it depends on the enrollment instead. This is the case because classes such as BLG500 may have enrollments created in different semesters for the same crn which makes this information functionally dependent on the enrollment. grade information is functionally dependent both on crn and student number. grade information might have also been dependent on the class name instead of crn but that assumes that a student can have only a single grade for a class which unlike ITU is not the case for all universities.

The letters a class covers are dependent on the name of the class. Just like that also the major restriction of a class is dependent on the name of that class. post_content functionally depends on crn and person_id. And finally the descriptions of universities classes majors and faculties are also all functionally dependent on their names.

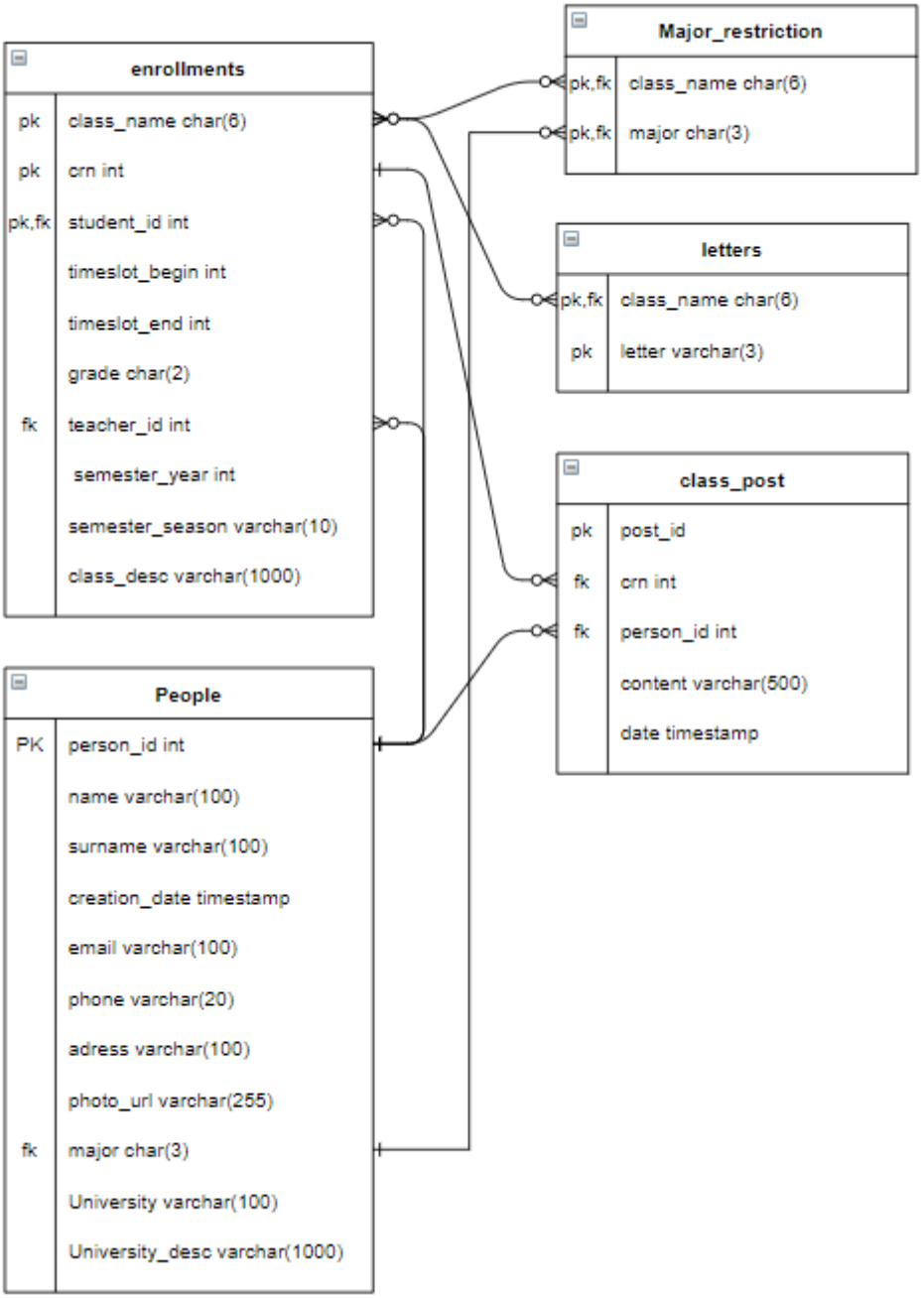
2.3 Normalization Steps

0’th normal form:



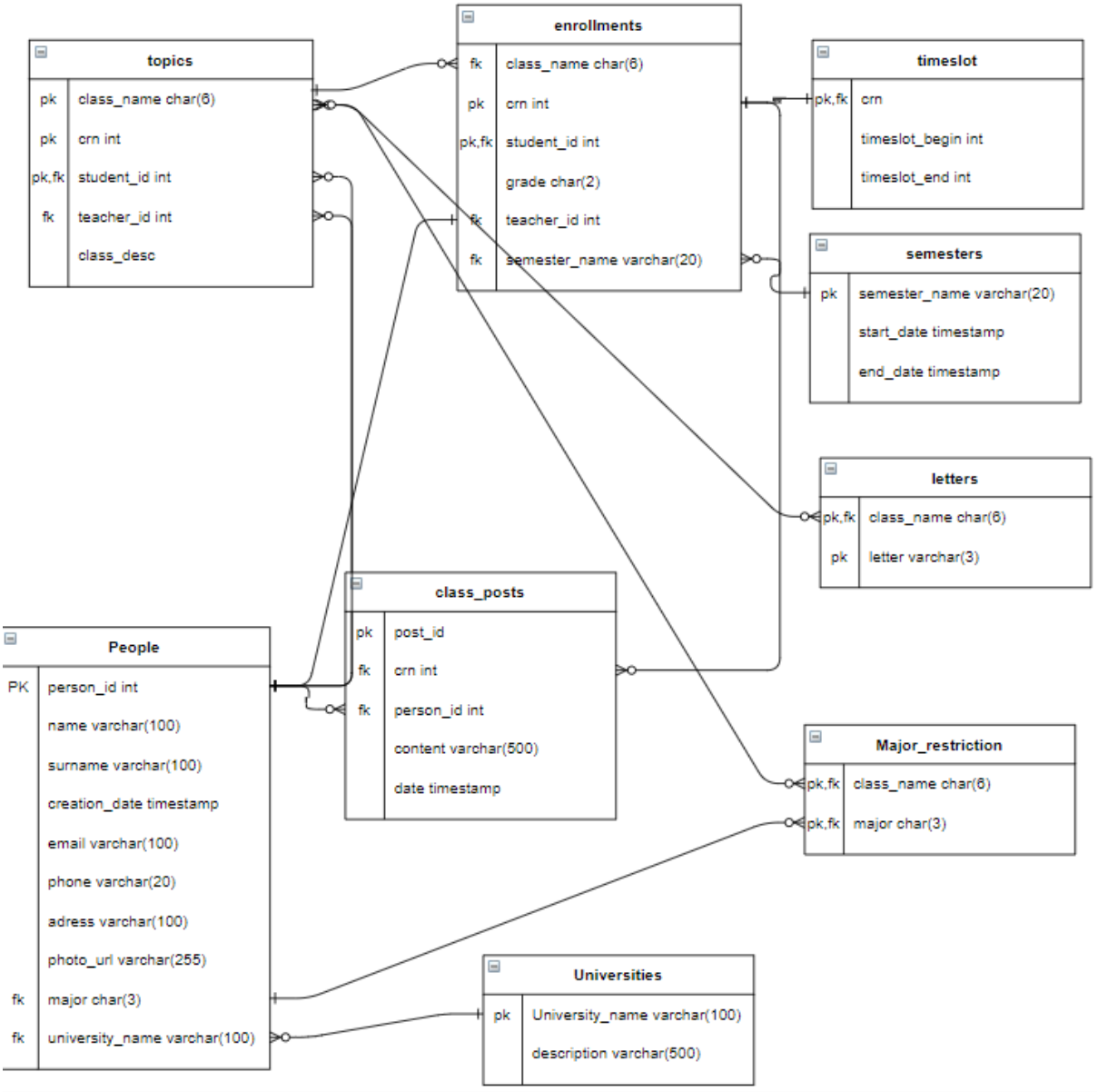
In the 0'th form we have All the fields that we will keep the data for. We have the personal data of students' enrollment information grades and class posts. We also intend to keep passwords for people in hashed form for authentication purposes. We also have data for universities.

1’st normal form:



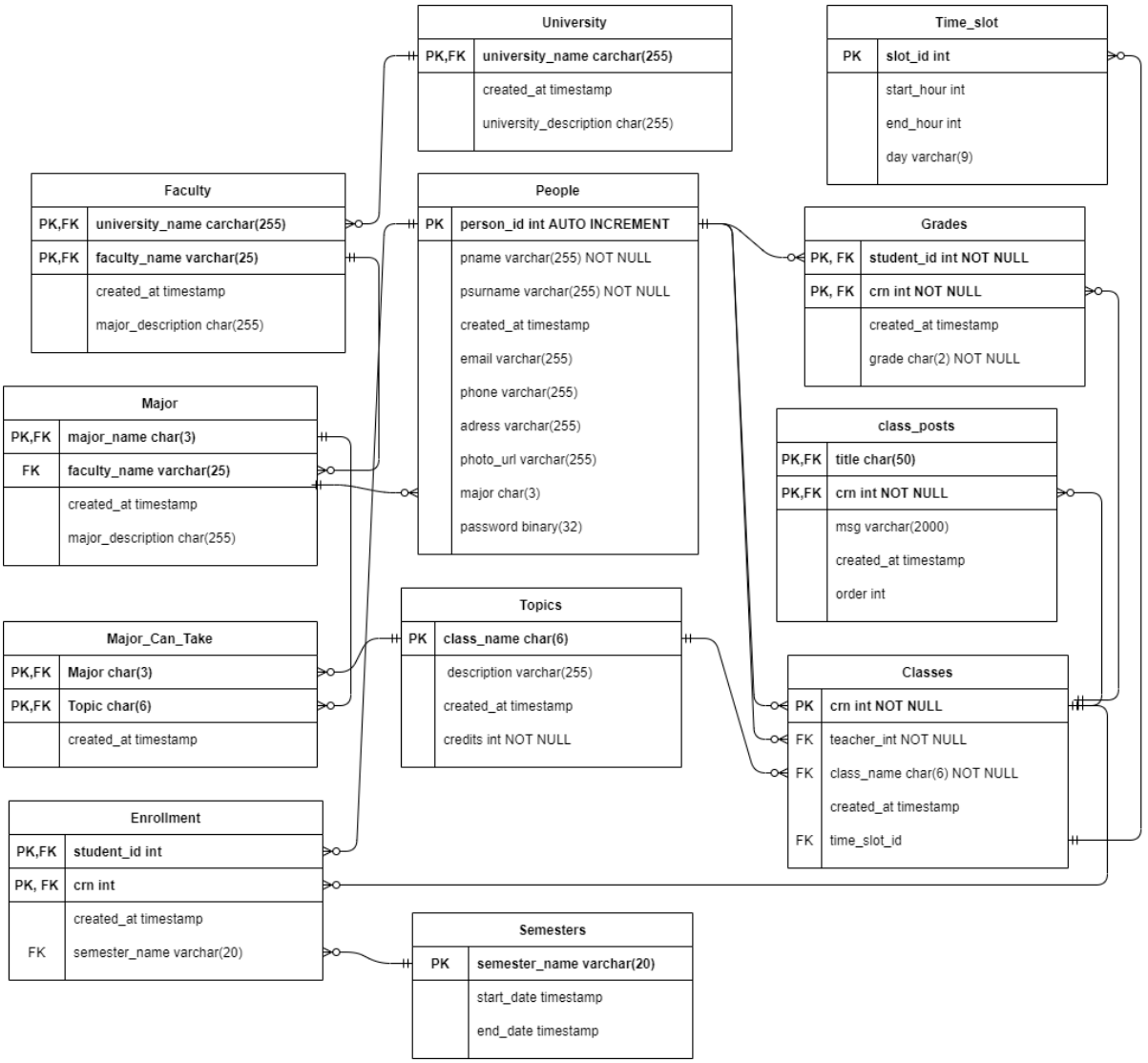
In the first normal form we move major restrictions letters and class posts out of people. We also separate classes from students. We do this because in the 0’th form we have had these fields as groups which it shouldn’t in the first normal form. In this diagram the major restriction decides who can pick what class. if a class does not have any major restrictions anyone can take that class.

2’nd normal form:



In the second normal form we have divided the classes table further into separate parts. classes made up of three kinds of information which are the data functionally dependent on the crn the student id and the class name so we divide these 3 parts into new tables. We also separate the timeslot data from classes and university data from people to satisfy second normal form conditions.

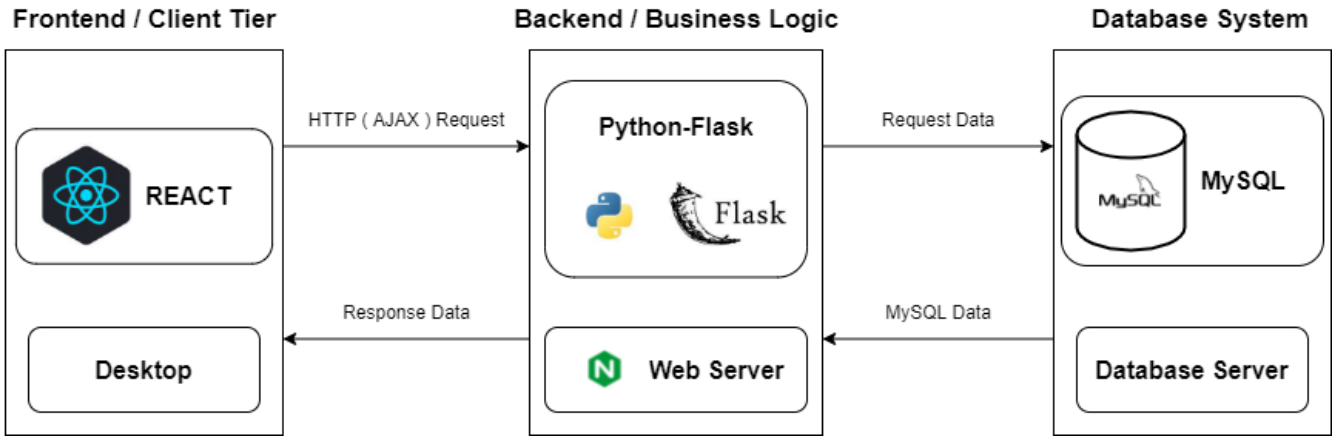
3’th normal form (ERD)



This Is the third and final form of our database. Here I have moved grades out of the data that was tied to the crn because the student number in that field depended on crn and the grade depended on the student number which created a transitive relation. Here we have also added information about faculty and majors but this was added after we have achieved third normal form so it has just been added to this table. In this database grades have delete and update cascade on their crn’s and person id’s which means that if a class or student gets removed from the database their grades also get deleted. The same thing applies for classes and student posts too. This also allows us to change class data without the need to modify past enrollments or grades. For example if a class was called mth112 and at some point and we decided that we wanted to delete it when we did that all the crn’s connected to that class would be deleted and take the enrollment and grading that with itself.

3. Application Details

3.1 Project Architecture



Frontend: As sistroid promises a student information system with a modern UI, we decided to use React framework for our front-end. With React , the request to our base URL is being responded with the whole content of the Sistroid application and user's will not wait for pages to load every time they skip between pages. Instead of requesting heavy html templates, users only request necessary data via AJAX requests. Less overhead of responses result in higher speeds of data retrieval.

Backend: For our backend, we are using python's Flask framework. We've built our REST API to listen for requests such as GET,POST, and DELETE. After receiving the request, Flask delivers the related SQL query to MySQL database, processes the business logic and makes calculations if required, then serves the response of the AJAX request to the client. If the requests include wrong data types or structures, the backend sends an error object.

Database: We are using MySQL as our database as it offers high speed to execute many queries at a time and provides management systems with modern user interfaces such as MySQL workbench. MySQL is responsible for storing, updating and deleting data when the related query is received from the Flask backend. The data in MySQL is stored in 3rd normal form and the tables are structured according to that making the retrieval and update of data faster and less error prone.

Documentation

- **Description**

Sistroid is a student information systems web application written in Javascript-React for the frontend and Python-Flask for the backend with MySQL as database. It is built to provide a smooth user experience to the students with a modern user interface. The project source code is maintained in GitHub as an open source project. Related github link is https://github.com/WebPunisher/sistroid

- **Installation Instructions**

Clone down the repository using the link given above with the .git extension. Make sure to have node and npm installed on your machine to run the project properly. After cloning, change to the frontend directory and follow the steps given below to start the frontend server on localhost at port 3000.

- yarn install => to install dependency packages
- yarn run => to run the frontend server

After that, change to the flask directory to start the backend server. Note that the backend server runs on port 1999 as default.

- python app.py => to run the backend server

For the application to run properly, you need to have MySQL installed with necessary user configuration which is specified in the app.py file. To experience the demo with example data, you have to make a call to /reset api endpoint. You can do that via entering http://127.0.0.1:1999/reset to the URL bar on your browser.

- **Backend API (Application Programming Interface)**

/create_tables = Creates all the database tables . Used for development purpose and initialization.

/clear_tables = Removes all the tables from database. Used for development purposes

/seed = Randomly seeds the database with the data taken from seeddata.json file . Used to generate example data for the database.

/gpa_calculator = Calculates the gpa taking the classes and the grades as input. Returns the calculated gpa.

/ranking/<major> = Queries the database to get the ranking of the students from the specified major in terms of GPA. Returns the ranking that includes the name and the gpa of the students.

/student_info/<student_id> = Queries the database to get personal information, classes , gpa, and grades of the student specified with the student id.

/available_classes = Queries the database to fetch the classes that are available for the current semester. Returns the classes list that include information such as credits, time, crn, teacher_name, letters, and major restrictions for each class.

/crn_info/<crn> = Fetches and returns the general information, students enrolled, grades, and class average information for the class with the specified crn.

/mentor/<student_num>/<available_credits>/<desired_gpa> = Takes the current situation of the student as input and makes calculations to check if the student can reach desired gpa in the current status. Returns the solution if there is one, returns a message if not.

Histogram Endpoints

/crn_histogram/<crn>
/class_histogram/<class_name>
/teacher_histogram/<person_id>
/teacher_class_histogram/<person_id>/<class_id>

The four histogram endpoint given above fetches the related information from the database with queries. Then uses multimedia functions importing the cv2 library and generates a jpeg file with the given data. Generated jpeg file is returned to the frontend to be displayed there.

React Components

- **add_drop**
  - Shows ongoing classes and include available_classes
  - Takes the classes CRNs to drop or add as input and make the necessary API call to the backend server as a POST request

- **available_classes**
  - Fetches classes that are available for the current semester in a list format with filters and properties such as teacher name, crn, semester, etc.
- **class**
  - A component to display properties of a class, takes class_name, class_description, credits, teacher_name, crn, and semester as props.

- **gpamentor**
  - Fetches the gpa and the old classes of the student
  - Takes the desired gpa and available credits as input and provides a possible scenario to achieve desired gpa.

- **histogram**
  - Takes teacher_id, class_name, crn, or their combinations and displays the histogram related to the input. The histogram is created in the backend, the component just request it and display the response histogram.

- **myclasses**
  - Fetches old classes and ongoing classes of the student
  - Display the classes in a list format with filters in every property to make lookup faster and easier.

- **profile**
  - Fetches the student's personal information, contact information, and image.
  - Displays the information in an editable format for contact information such as phone number, and non-editable format for general information such as major and age.

- **ranking**
  - Fetches the ranking of the students in terms of their GPA from all majors
  - Provides a filter for the major