CityDevsCCNY

GradSchoolZero
Design Report
For Web Application

Version 1.0

Revision History

Date	Version	Description	Author
25/Nov/2021	1.0	Phase 2 Report for GradSchoolZero Project	Anil Bhusal Anthony Liang David Jimenez Moshahid Kallol Nafis Khan

Table of Contents

1. Introduction:

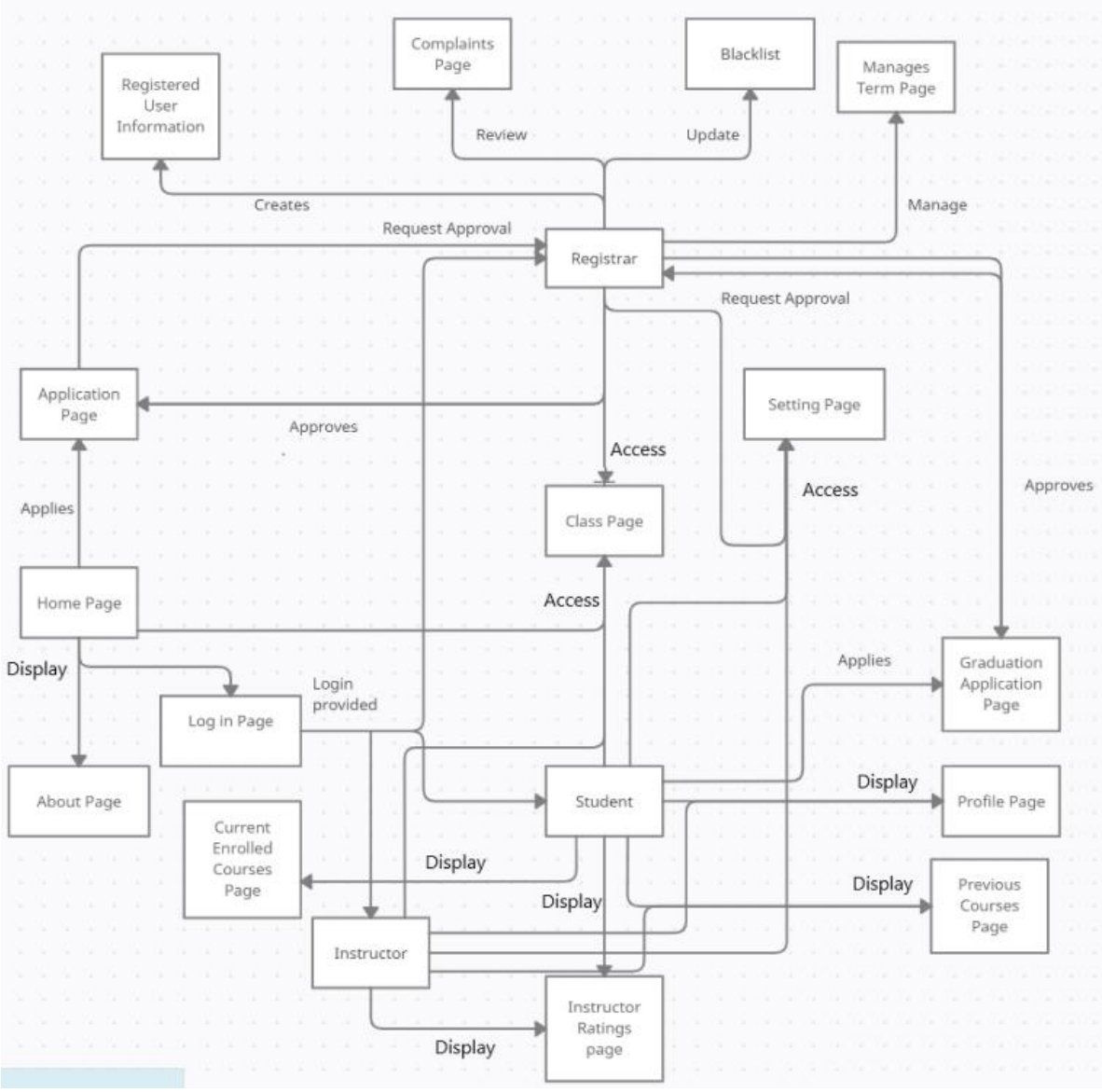
This is an overview of the design and functionality of GradSchoolZero developed by the organization CityDevsCCNY. Our main goal is to establish a simple user interface between CCNY Registrars, Professors and the students. The interface will make the process of setting up and enrolling in classes much simpler.

1.1 Purpose

This report details all the functionalities and uses of GradSchoolZero so that readers can have a good understanding of how everything works, what types of users the website is dedicated towards and how to navigate around the website in general.

1.2 Collaborative Class Diagram

A collaboration class diagram model describes how various groups of elements in our design collaborate with each other through various processes. The diagram shows the fundamental objects of our system and the various processes that define their interactions with each other. The following collaboration class diagram outlines the interactions of our various users and elements for our GradSchoolZero site.

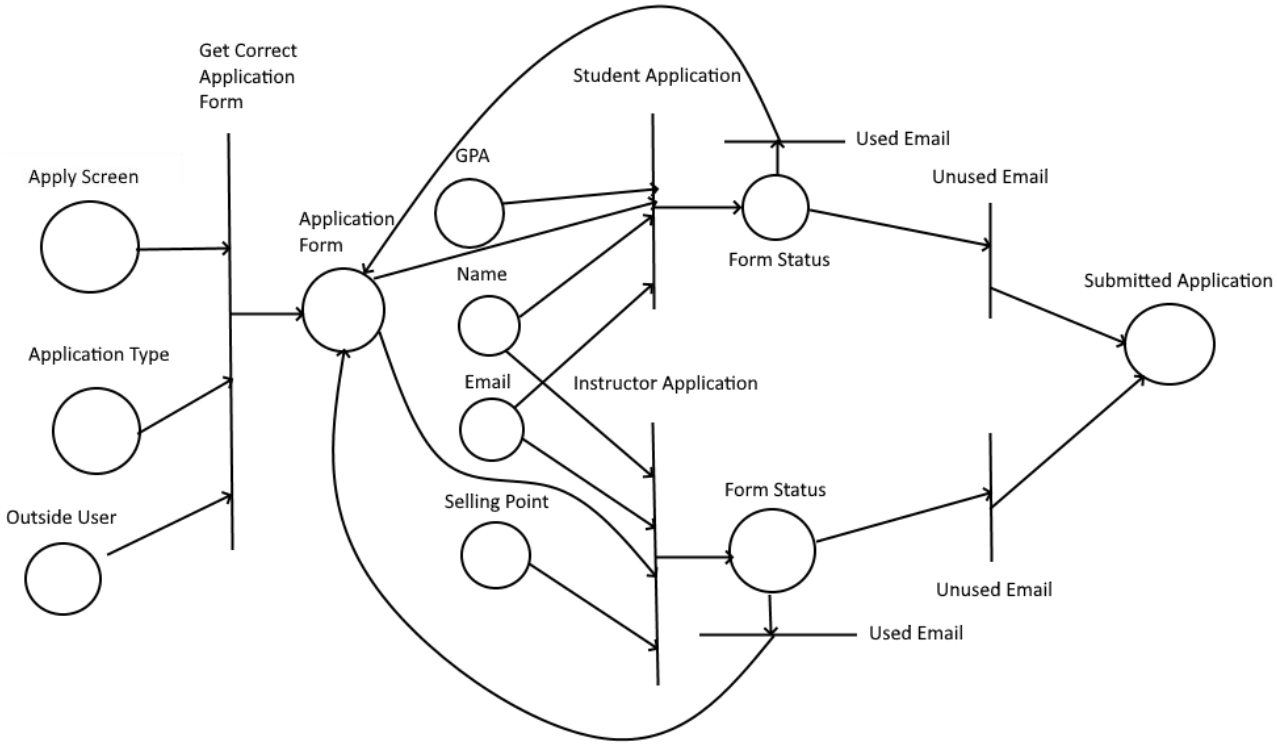


2. Use Cases Analysis:

2.1 Apply

a. Normal: A user inputs their application information (name, email). Once the information required by the application form type (either student or instructor) have been filled out and the form submitted, the form will be sent to the database.

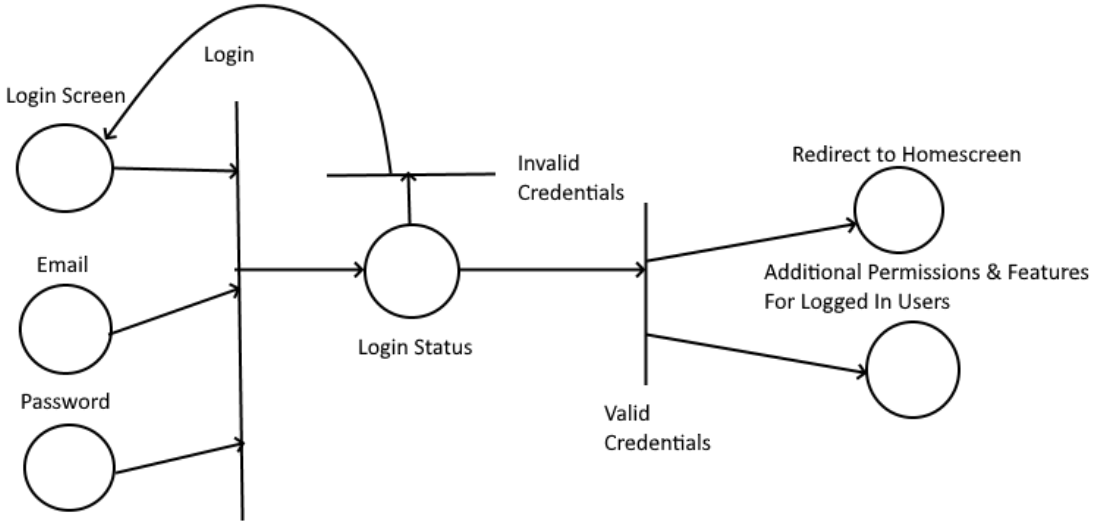
b. Scenario: If the user is applying to be a student, they will also input their GPA into the form. If the user is applying to be an instructor, they input some information about why they're qualified. If the email they used for their application has been already used in a different (unchecked) application, they'll get a notification telling them to use a different email.



2.2 Login

a. Normal: A user inputs their login information using their GradSchoolZero email, and the password given to them. The user Logs in as the type of user they applied as.

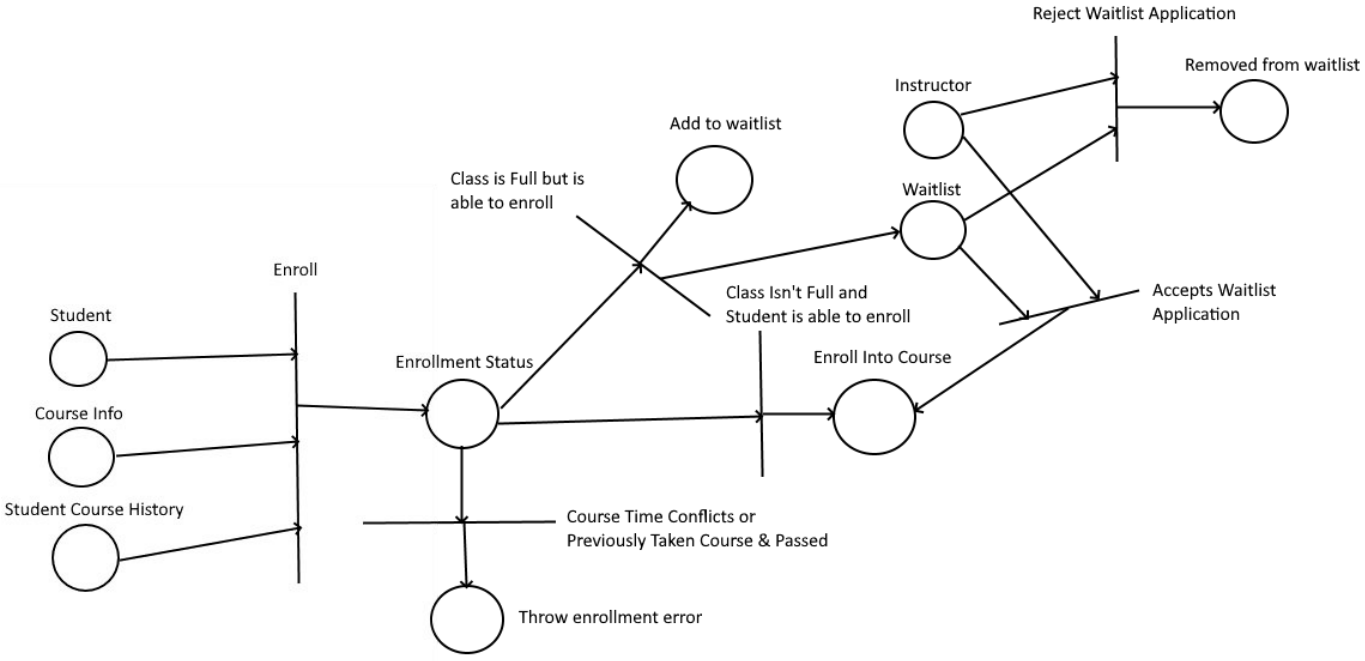
b. Scenario: If a user tries to log in with the wrong credentials, they get an alert on the login screen that notifies them of the invalid login credentials and so has to try again with the correct credentials to be able to access their data.



2.3 Class Enrollment

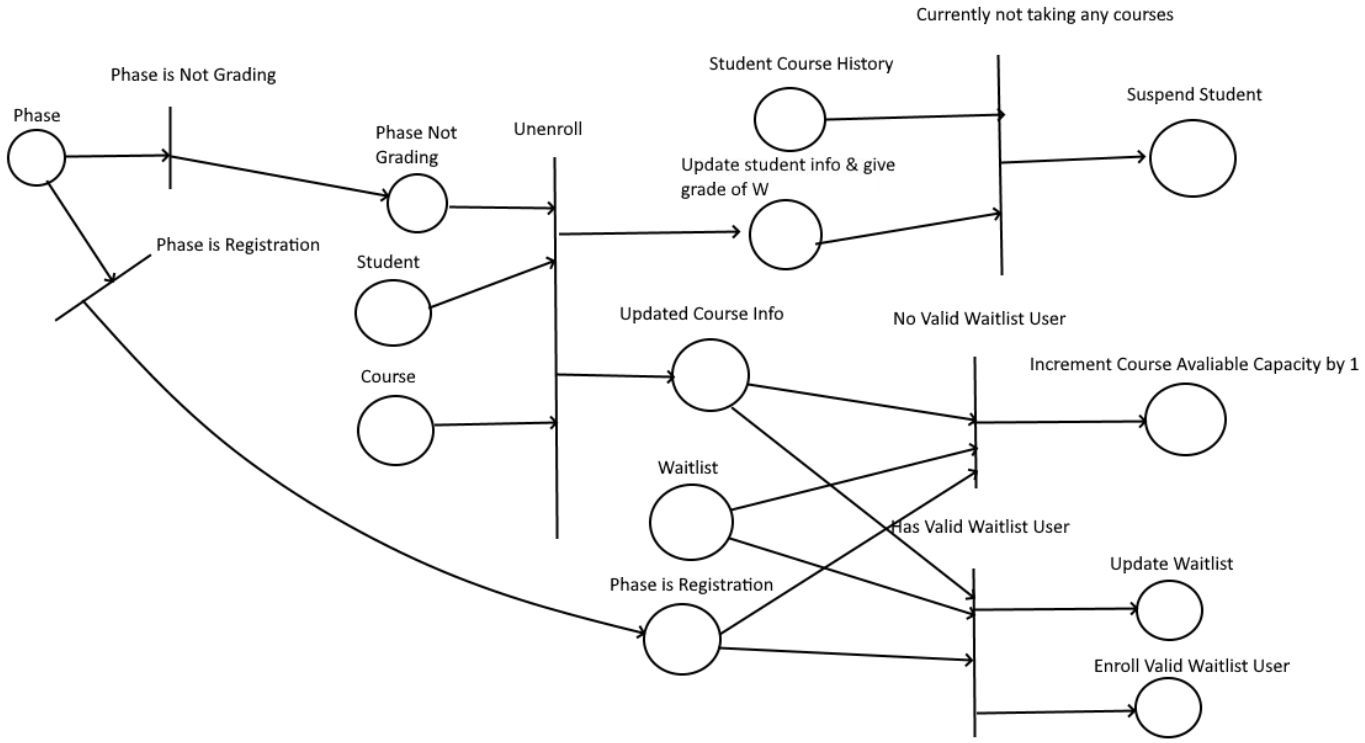
a. Normal: A student is able to enroll in a course during the course registration phase given that they have no time conflicts with other enrolled courses and if the course isn't full.

b. Scenario: If the course is full, the student is added to the waitlist where an instructor can add them in. If the student withdraws from a course and tries to enroll in the course again, they'll be rejected. A student can enroll in the course if they received an F previously. A student can also enroll if they receive special registration permission.



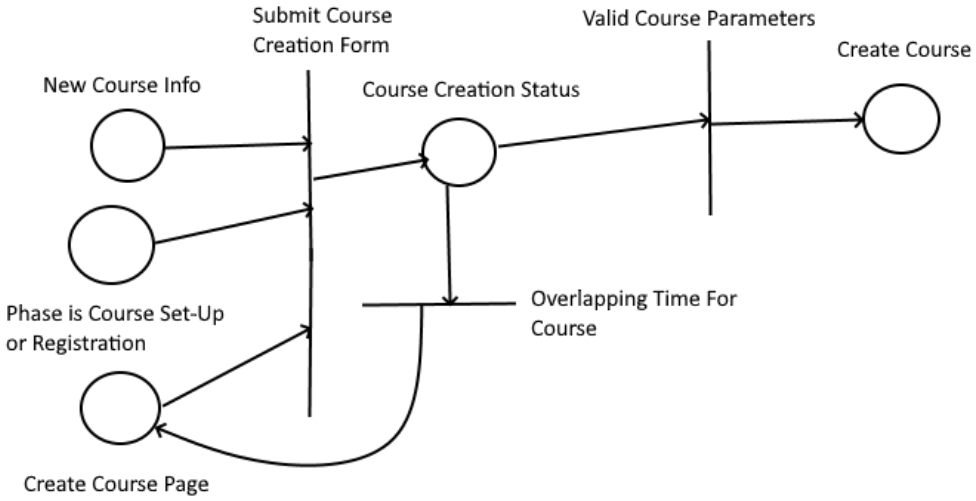
2.4 Class Unenrollment

a. Normal: If the student is enrolled, they can withdraw from the course for a grade of “W”. The first student on the waitlist that can enroll in the course will be enrolled given the phase is still in registration.
b. Scenario: Students are not allowed to unenroll from a course during the grading phase due to the course coming to an end and restrictions made by the system/school. If the user dropped all their courses, they’ll be suspended.



2.5 Create Course

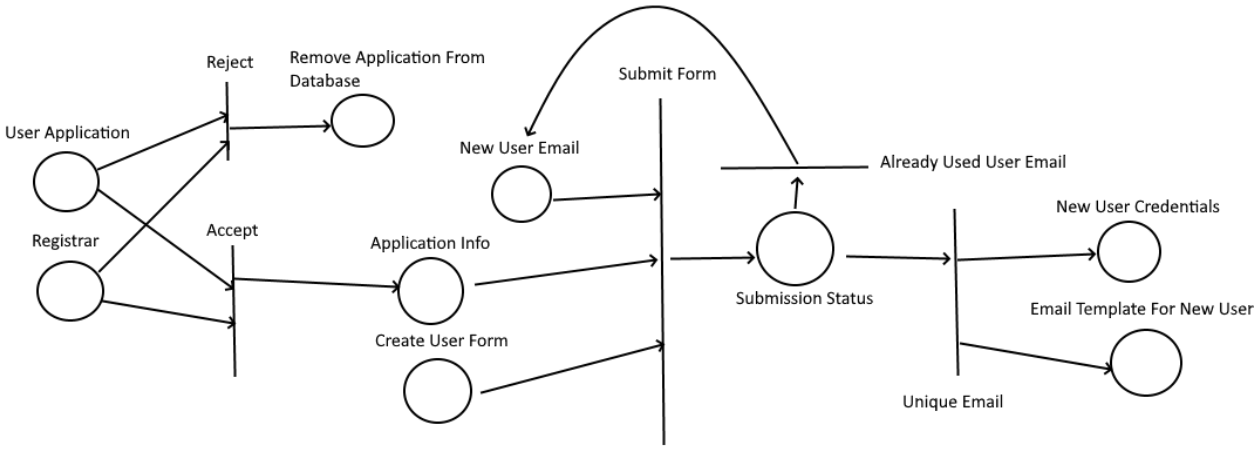
a. Normal: The registrar creates a course for the new term from the course list during the course set-up and registration phase and assigns the course to an instructor. The registrar also sets the maximum capacity for the course and sets the time for it.
b. Scenario: If we set overlapping times for when this course will happen, an error will show up that notifies the registrar of the overlapping time slot.



2.6 Create User/Application Acceptance

a. Normal: The registrar can accept the user given they meet the requirements. They can reject the application if the user doesn't satisfy the program requirements (i.e.: not high enough gpa). An instructor can be accepted or rejected without reasoning. After accepting the application, they'll be presented with a form to create a user with the data already filled out. After submitting that form, they'll receive a template email to send to that user that we created an account for.

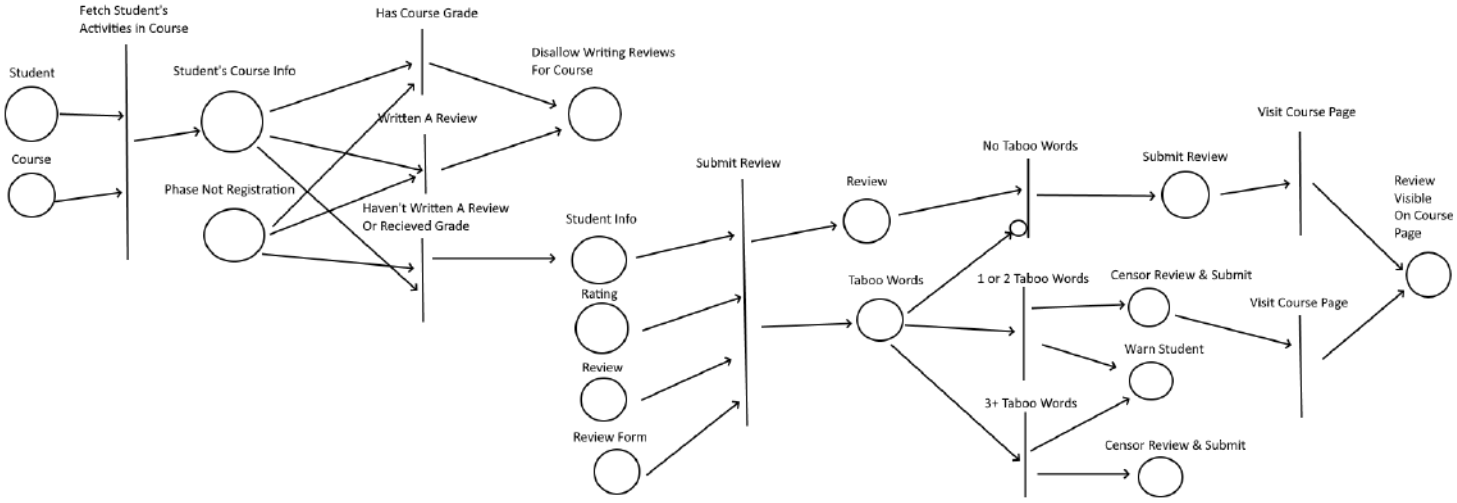
b. Scenario: If the registrar accepts/rejects the requirements, they're required to give a brief justification. When creating the user, if the auto-filled-out information (specifically the email) has already been used, they'll be prompted with an error to change the email.



2.7 Write Review

a. Normal: A student enrolled in a course can write a review for the course given they haven't received a grade and that the phase isn't the registering period. They can assign the instructor a rating from 1 to 5 and give a brief description for the reasoning.

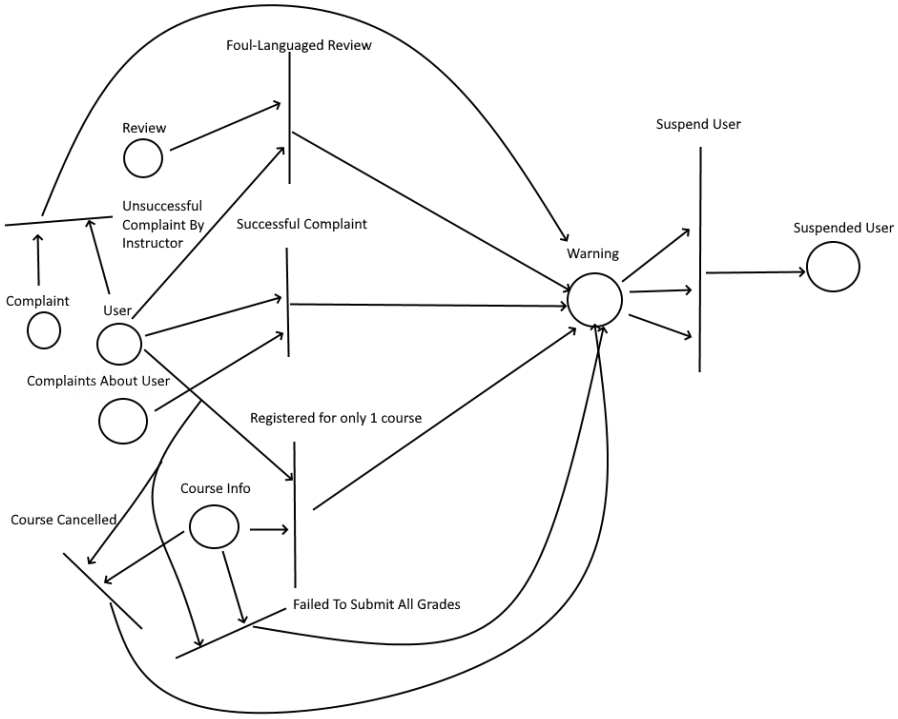
b. Scenario: If the review contains some taboo words, those words will be censored, and the student will receive a warning. If the review contains 3+ taboo words, the user is given 2 warnings and the review won't be displayed on the course page.



2.8 Warnings

a. Normal: An instructor or student may receive a warning from a complaint. Students may receive additional warnings from failing to register for enough courses, writing a foul-language review, or getting a GPA between 2 and 2.25. An instructor may receive additional warnings if their class has been canceled, if they submitted a false complaint, or didn't submit all their grades.

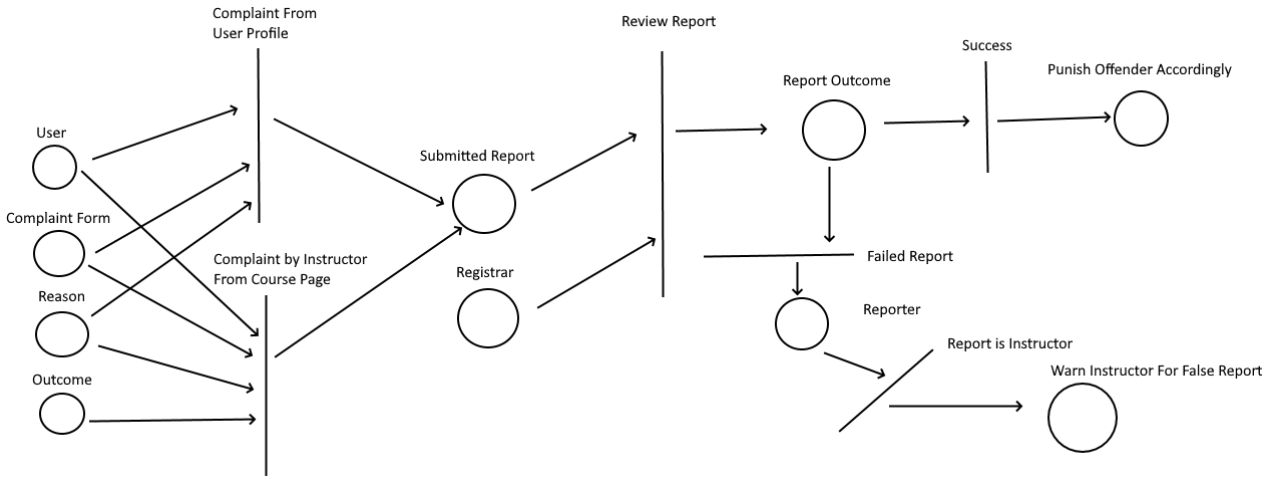
b. Scenario: If a user receives 3 warnings, they're suspended in the next semester.



2.9 Complaint

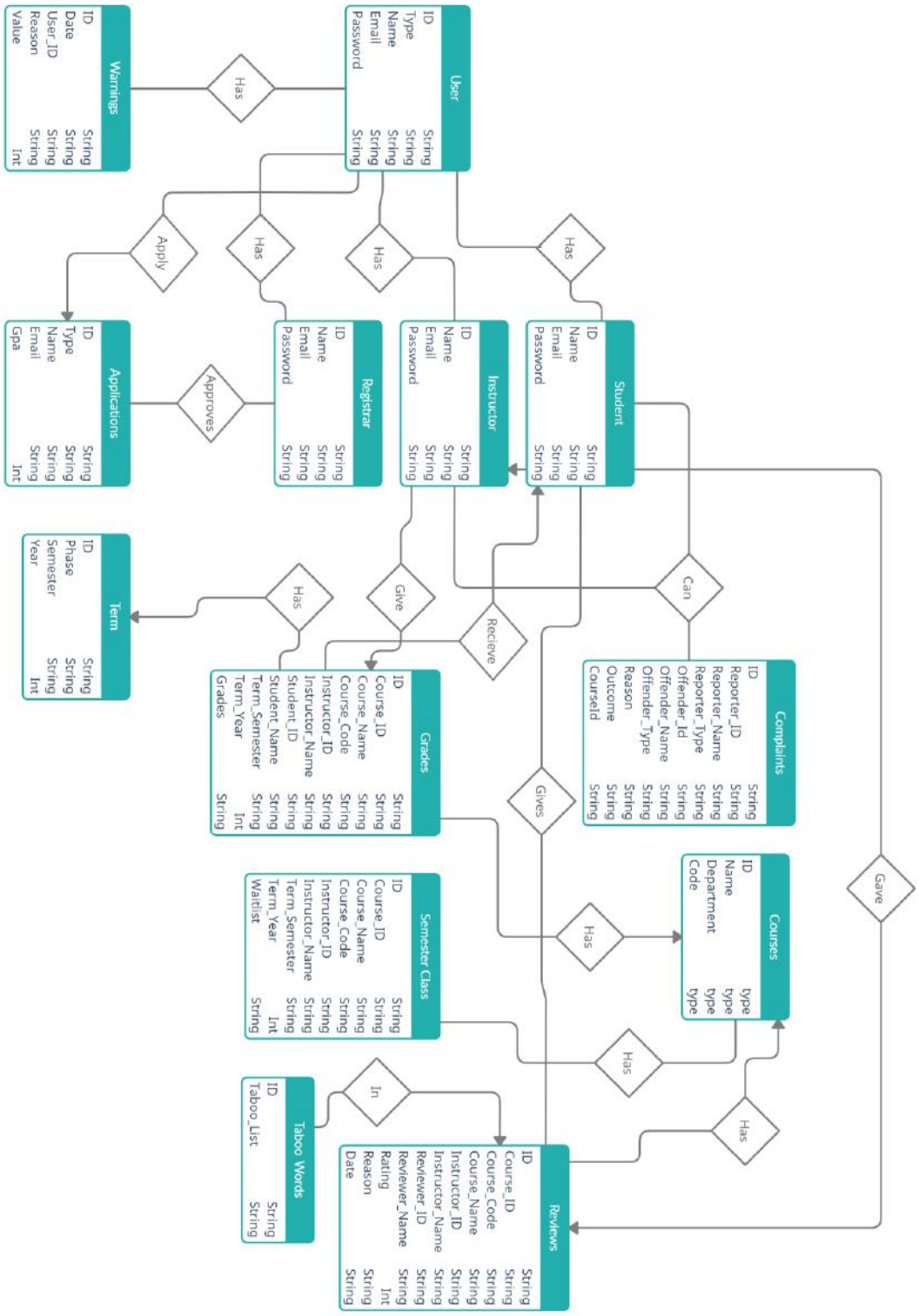
a. Normal: A user can submit a complaint to a student or instructor from a report button on the user's profile page. They must provide a reason for the complaint. If the warning is successful, the offender will receive a warning.

b. Scenario: An instructor can submit a complaint from the course page (that they teach) and selectively report students. An additional option will appear that'll allow the instructor to request for deregistration of the user in the complaint. If the instructor's complaint is successful, the outcome they desire will occur (student is deregistered or warned). If the complaint is unsuccessful, the instructor will receive a warning.



3. Entity-Relation Diagram:

The Entity-Relation Diagram is a visual representation of the information system depicting the relationships between different models such as people, objects, places, concepts, etc. The following ER-Diagram is a representation of the information or database system of the GradSchoolZero website that helps connect registrars, professors, and students and then helps them manage all courses using the website.



4. Detailed Design:

4.1 Create Application

HTTPS-POST
Received: application (name, email, GPA or description)
Returned: True or False

---

data = {
    name: name,
    email: email,
}
Also added to data on submission:
    id: uuid
    type: submitted form type
Depending on the type of the application we insert into data:
    gpa: gpa [Form type is Student]
    description: description [Form type is Instructor]
Returns true or false whether this POST request was successful or not

4.2 Check if Application Email is Used

Received: email
Returned: True or False

Check Applications database to see if there exists an entry in which the “email” = inputted email:
    If the length of the returned result is greater than 0:
        Return true
    Else send back false

4.3 Resolve Complaint

HTTPS-PATCH & HTTPS-DELETE
Received: complaintInfo (id, reporter: {id, name, type}, offender: {id, name, type}, reason, extra: {outcome, courseId}), outcome
Returned: True or False

Check if the outcome is “approve”:
    If the reporter is an instructor or registrar AND complaintInfo.extra.outcome is “de-registration:
        Fetch the grade object for the student and for course
        Set the grade of that object to “DW”
        Send a PATCH request with the updated grade object
    Else, run the addWarning function to add a warning to the user
Else, if the reporter was an instructor, give them a warning with the addWarning function
Remove the complaint from the database with a DELETE request
Returns true or false if the DELETE request was successful

4.4 Add Course

HTTPS-POST
Received: courseBase (id, code, name), instructorId, instructorName, time, capacity (max, available), termInfo (semester, year)
Returned: True or False

---

data = {

```
id: uuid,
course: {
        id: courseBase.id,
        code: courseBase.code,
        name: courseBase.name,
    },
instructor: {
        id: instructorId,
        name: instructorName,
    },
term: {
    semester: termInfo.semester,
    year: termInfo.year,
},
capacity: {
    max: capacity,
    available: capacity,
}
time,
waitlist: []
}
```

Returns true or false whether this POST request was successful or not

4.5 Cancel Course

HTTPS-PATCH & HTTPS-DELETE

Received: courseId

Returned: True or False

Fetch from the database the information of the course and all the students taking the course:

- For each student:
  - Give property of “specReg: true”
  - Send PATCH request to update student information
  - Send DELETE request to delete student’s grade object for the course
- For instructor information from course object:
  - Give them a warning using the addWarning function

Send a DELETE

Returns true or false if the DELETE request was successful

4.6 Enroll Course

HTTPS-POST

Received: user (id, name, specReg, suspended, expelled), courseInfo (id, course: {code, name}, instructor), termInfo (semester, year, phase)

Returned: Status Object (status, title, details)

Get user information:

- If phase !== “registration” AND !specReg, suspended = true, or expelled = true:
  - Return status object with status value of “error”

Get all of student’s grades

- If currently taking < 4 courses AND hasn’t previously withdrawn from course AND taken the class 0 times or taken once but got an F:

Combine the times of all the courses currently taking and this course:
If no time conflicts:
Check if course if full:
If not full:
Enroll the student into the course, create a new grade object for that course and POST it.
Else, add the student to the waitlist
Return status object with status value of “success” if student was successfully enrolled or added to the waitlist or “error” if some error occurred
Else, return a status object with a status value of “error”
Else, return a status object with a status value of “error”

4.7 Unenroll Course

HTTPS-PATCH
Received: userId, courseId, termInfo (semester, year, phase)
Returned: Status Object (status, message)

If phase = “grading”:
Return status object with status value of “error”
Else, remove user from course by sending a PATCH request to update their grade object for the course with a “W”, then:
If phase = “running”:
Check if student has dropped all their courses:
If they have:
Give them 3 warnings through the addWarning function (to suspend them)
If phase = “registration”:
Fetch course information with GET request:
If the waitlist isn’t empty:
For each student in the waitlist:
If student can enroll into the course (given they meet the requirements):
Enroll student into the course
If no students were enrolled this way:
Send a PATCH request to update course to increment “available” property in “capacity” by 1
Else, send Patch request to update course to increment “available” property in “capacity” by 1
Return status object with status of “success”
Else, send Patch request to update course to increment “available” property in “capacity” by 1 and return a status object with status of “success”

4.8 Add Warning

HTTPS-POST & HTTPS-PATCH
Received: userInfo (id, name), reason, warningCntValue
Returned: True or False

Send a POST request to the database, posting a warning object structured as:

```
{
    id: uuid,
    date: new Date(),
```

```
        user: {
            id: userInfo.id,
            name: userInfo.name,
        },
        reason,
        value: warningCntValue
    }
```

Fetch the user from the database and increment their “warningCnt” property by 1 and update the database with this using a PATCH request

Return true or false whether the PATCH request was successful

4.9 Submit Complaint

HTTPS-POST

Received: reporter (id, name, type), offender (id, name, type), reason, extra (outcome, courseId)

Returned: Status Object (status, message)

```
data = {
    id: uuid,
    reporter: {
            id: reporter.id,
            name: reporter.name,
            type: reporter.type
        },
    offender: {
            id: offender.id,
            name: offender.name,
            type: offender.type
        },
    reason,
    extra
}
```

Returns a status object with status value of “success” if the POST request was successful, otherwise, returns a status value of “error”

4.10 Suspend User

HTTPS-PATCH

Received: id

Returned: N/A

Fetch user with the input id from the database:

    If their “warningCnt” >= 3:

        Send PATCH request to set “suspended” to true and “warningCnt” to “warningCnt” - 3

4.11 Add Review

HTTPS-POST

Received: courseInfo (id, code, name), instructorInfo (id, name), reviewerInfo (id, name), rating, reason

Returned: Status Object (status, message)

Count the number of taboo words in “reason” (save value into “numTaboo”) and filter “reason” (save value in “filteredReason”):

    If “numTaboo” > 0:

If “numTaboo” < 3:
Give reviewer 1 warning using the addWarning function
Else, give reviewer 2 warnings using the addWarning function

Send a POST request with the following body:

```
data = {
    id: uuid,
    course: courseInfo,
    instructor: instructorInfo,
    reviewer: reviewerInfo,
    rating,
    reason: filteredReason,
    show: numTaboo < 3,
    date: new Date()
}
```

Returns status object with status value of “success” if the POST request was successful, otherwise, return a status value of “error”

4.12 Add Taboo Word

HTTPS-PATCH
Received: word
Returned: True or False

Send PATCH Request with body of [...tabooList, word]
Return true or false if the PATCH request was successful or not

4.13 Delete Taboo Word

HTTPS-PATCH
Received: word
Returned: True or False

Send PATCH Request with body of the “tabooList.filter(tabooword => tabooword !== word)”
Return true or false if the PATCH request was successful or not

4.14 Next Phase

HTTPS-PATCH
Received: None
Returned: Status object (status, msg)

Get basic structure of the next phase object:
If next phase is “set-up”:
Remove all suspended users
For each instructor:
Update their ratings
Warn instructor if their average rating is < 2 and if they failed to assign grades to all the students in their courses
Notify instructor with a class gpa < 2.5 or > 3.5
For each student:
Update their GPA
If student GPA < 2
Expel student
If student GPA > 2 & < 2.25:

Give warning to student demanding a meeting with the registrar
If student GPA > 3.75 for the term or > 3.5 overall:
Give student honor roll for this outgoing semester
Remove 1 warning from student's "warningCnt"
If student fails the same course twice:
Expel student
For each user:
If user's "warningCnt" >= 3:
Suspend user
For each course:
Update its overall rating
If the next phase is "registration":
If semester current has no courses:
Return status object with status value of "danger"
If the next phase is "running":
Indicate special registration phase has started in database
For each student:
If student is taking < 2 courses:
Warn student for taking < 2 courses using addCourse function
Get a list of all cancellable courses and for each of these courses:
If class has < 5 enrolled students:
Run the cancelCourse function
For each instructor teaching no courses:
Give them 3 warnings (which will suspend them in the next semester once we return to the "set-up" phase") using the addWarning function
If next phase is "grading":
End the special registration phase (if the registrar has or hasn't end it)
Send a PATCH request updating the phase in the database and return a status object with status value of "success"

4.15 **Create User**

HTTPS-POST
Received: userInfo (id, name, type, email, password)
Returned: True or False

```
data = {
    ...userInfo,
    warningCnt: 0,
    suspended: false,
    removed: false
}
```

Depending on the type of the user we're creating:
If student:
specReg: false
graduated: false
honorRoll: []
GPA: null
applyGrad: false
If instructor:
rating: null

Returns true or false whether this POST request was successful or not

4.16 Check if User Email is Used

Received: email
Returned: True or False

Check Users database to see if there exists an entry in which the “email” = inputted email:
    If the length of the returned result is greater than 0:
        Return true
    Else send back false

4.17 Change Password

HTTPS-PATCH
Received: oldPass, newPass
Returned: Status Object (type, message)

Fetch user information from database:
    If “oldPass” = user’s password:
        If newPass is empty:
            Return Status Object with type value of “danger”
        Else, update the database with the new password with a PATCH request.
            If the PATCH request was unsuccessful:
                Return Status Object with type value of “danger”
            Else, return Status Object with type value of “success”
    Else, return Status Object with type value of “danger”

4.18 Remove User

HTTPS-PATCH
Received: userId
Returned: True or False

Fetch user information from database:
    Set user’s “remove” property to true and update using a PATCH request
Return true or false based on if the PATCH request was successful or not

4.19 Apply for Graduation

HTTPS-PATCH
Received: id
Returned: True or False

Fetch user information from database:
    Set user’s “applyGrad” property to true and update using a PATCH request
Return true or false based on if the PATCH request was successful or not

4.20 Calculate GPA

Received: gradeArray
Returned: gpa

Create a validGrades array in which we populate it with grades that aren’t defined
Create a total value in which we take the sum of the mapping of each grade in validGrades to its respective numeric value from a grade map (that maps letters to numbers)
    If the validGrades is empty:
        Return null

Else, return the total value divided by the number of valid grades and fix it to 2 decimal places

4.21 Get Student GPA

Received: id
Returned: gpa

Fetch all of the grades in the database for the student and create a grade array:
    For each grade returned:
        Add to the grade array the grade
    If the grade array is empty:
        Return null
    Else, return the result from the calculateGPA function with an input of the gradeArr

4.22 Calculate Average Rating

Received: reviewArray
Returned: rating

If the inputted array is empty:
    Return null
Else, get the sum of all ratings in the reviews and return that divided by the number of reviews

4.23 Get Instructor's Rating

Received: id
Returned: rating

Fetch all of the reviews in the database for that instructor id
    If the number of reviews is 0:
        Return null
    Else, return the result from the calcAvgRating function with an input of the reviews array

4.24 Check Time Conflicts

Received: timeArray
Returned: True or False

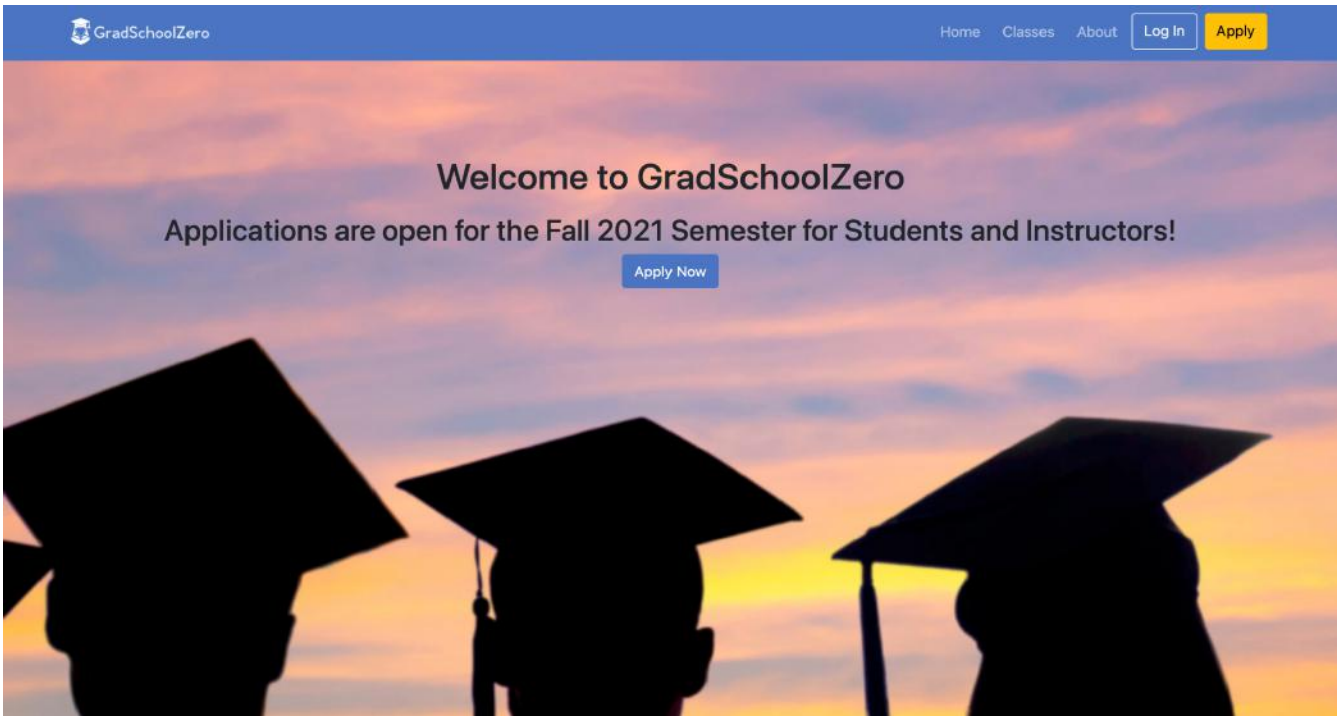
If the inputted array is less than 1:
    Return false
Create a conflicts variable initialized to false:
    For each time in the time array:
        For each time in the time array:
            If the indices are the same or the days are different:
                Skip this iteration
            Else:
                If the 1st start time is before the 2nd start time and the 1st end time is before the 2nd start time:
                    Set conflicts = true
                If the 2nd start time is before the 1st start time and the 2nd end time is before the 1st start time:
                    Set conflicts = true
Return the value of conflicts

5. System Screens:

Better GUI is required in the program to make the program more user-friendly. In our Project, we are working hard to make that possible. Some of the GUI that we built for our Project is presented below:

5.1 Home Page

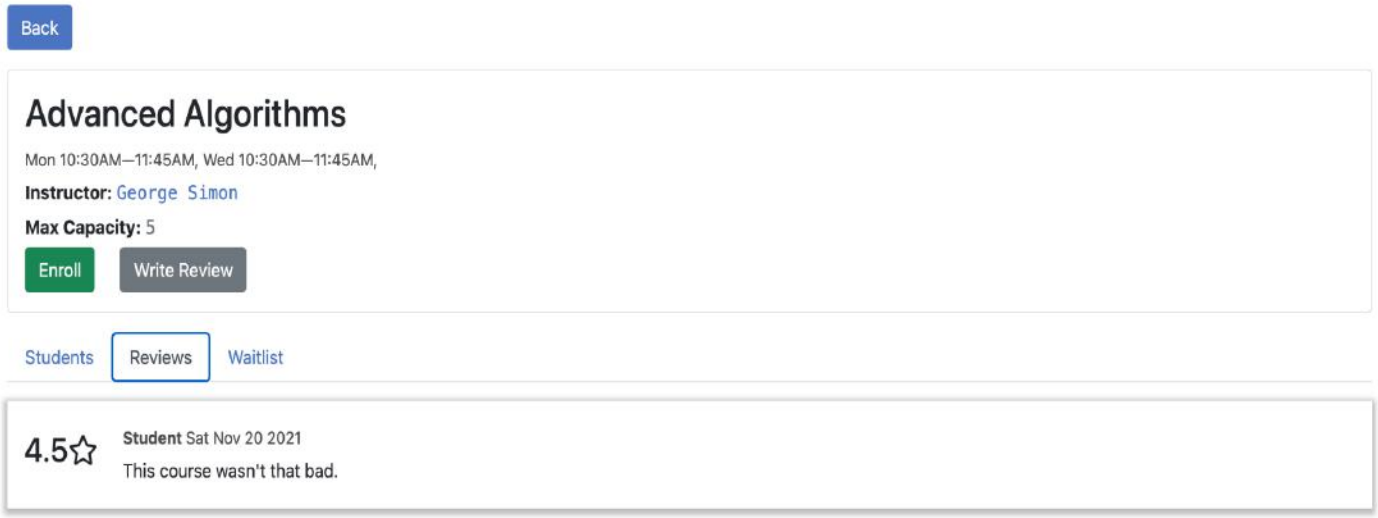
Home Page is the main attraction of the users. When they first open the program, they will see the home page. In-home page, different buttons direct us to different phases of the program. For example: the Apply button at the middle of the home page directs us to the apply page, which will show visitors to apply as a professor or as a student. Other buttons like classes which will show the offered classes for the semester (discussed below in detail), About button will direct us to the About page, which explains GradSchoolZero, the login button will direct us to the login page, from where all the users will log in. Apply button at the top right and middle of the page will perform the same operation.



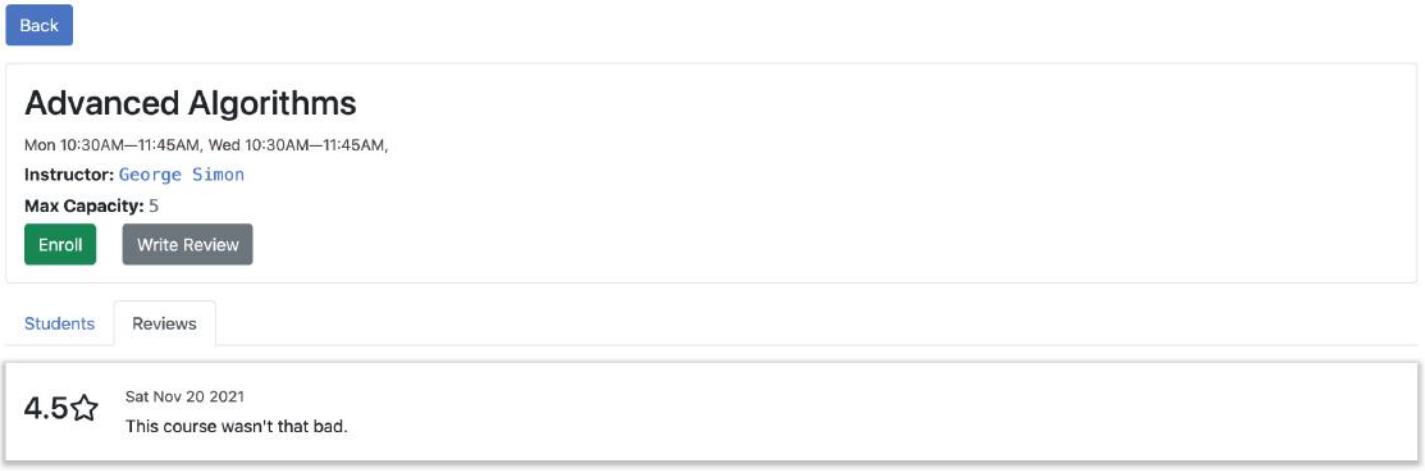
5.2 Course Page

We have created a page that displays the courses that are offered for the semester. The GUI of that page is shown below:

Course page is available to all users i.e. Register, Professor, Student, and visitor. But different users have different permission to do the things.
For Register: Register will have access to control everything for the courses. For example, the register can see who is in that class, what is the rating of the class, who are waiting for that class and so on. Registers can not enroll and write reviews for the class; those menus will not work for them but they will have all the control to them.



For instructor and student: The instructor and student have access to see who is in that class, see a review of that class, Max capacity of that class, who is assigned for the class, scheduled for the class. If the class is not full, students can enroll for that class and be on the waitlist. The GUI implemented for now for the professor and student is the same for now, but they have different functionality. Also, It is shown below:



The visitors can see the offered classes for the semester like professor and student, but they are only visible to them just to provide general information about the classes. But they cannot perform any things unless they are approved by the register.

6. Group Meeting Memos:

Meeting #	Date	Agenda
1	9/3/2021	Research/talk about ideas
2	10/1/2021	Brainstorm and discuss possible technologies that we are going to use for the project.
3	10/3/2021	worked on making a website architecture with possible pages and features on each page. Sort out who is going to start on which part.
4	10/19/2021	worked on phase 1 report: most of the time we stayed on call and worked on different parts
5	11/12/2021	detailed discussion on how we are going to implement each feature and how they are going to look.
6	11/13/2021	evaluate the progress we have made so far and discuss the phase 2 report
7	11/14/2021	Worked on phase 2 report.

The general outlines of the meetings regarding the phase reports can be found at:
https://github.com/CityDevsCCNY/GradSchoolZero/tree/main/Design%20Details

7. Supporting Information:

Our code can be found at the following address:
https://github.com/CityDevsCCNY/GradSchoolZero

Our work was/is distributed via the following GitHub project management page:
https://github.com/CityDevsCCNY/GradSchoolZero/projects/1

The phase reports can be found at:
https://github.com/CityDevsCCNY/GradSchoolZero/tree/main/Phase%20Reports

Hand written team collaboration memo's can be found at:
https://github.com/CityDevsCCNY/GradSchoolZero/tree/main/Design%20Details