

---

# **Design Document**

**for**

# **Majorizer**

**Version 1.0**

**Prepared by Jacob Desilets, Jonathan Nordby, Ryan Stewart, and  
Kaitlyn Witt**

**Condor**

**11/10/21**

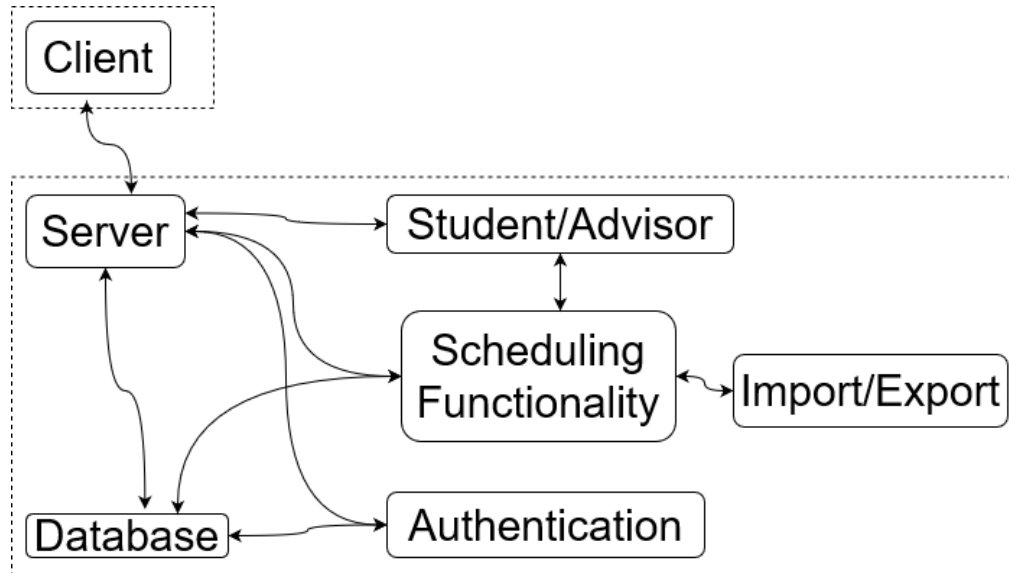
# Table of Contents

Revision History	2
<b>High-Level Architectural View</b>	3
<b>Class Diagrams</b>	4
<b>Sequence Diagrams</b>	9
<b>Database Layout</b>	12
ER Diagram	14
Relational Schema	14
<b>GUIs</b>	15

**Revision History**

<b>Name</b>	<b>Date</b>	<b>Reason For Changes</b>	<b>Version</b>
All of us	11/10/2021	Initial Writeup	1.0

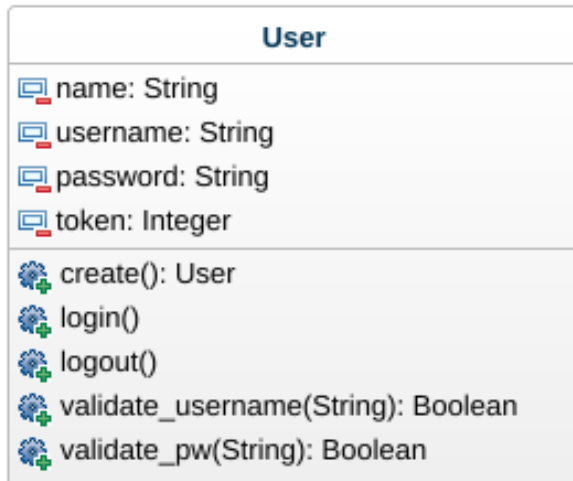
## High-level Architectural View



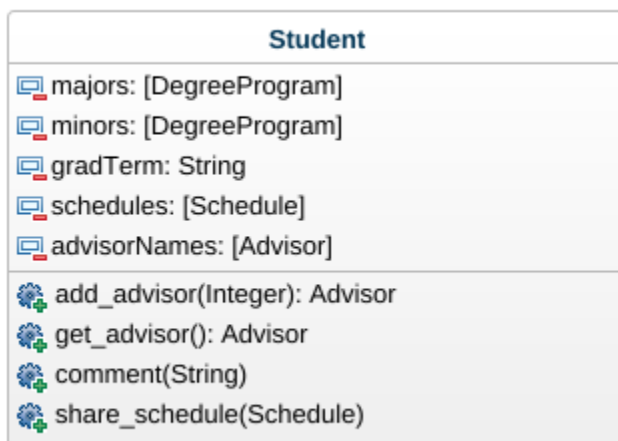
This application is based on a classic client-server model. All of the key functionalities of the application interact with each other within the server, followed by the interactions between the client and server separately.

# Class Diagrams

## User Class:



User is the basic class for all users. It contains their name as a string, which is the user's first and last name in one string. The username, password, and token are the parameters that have to do with authentication and verification.



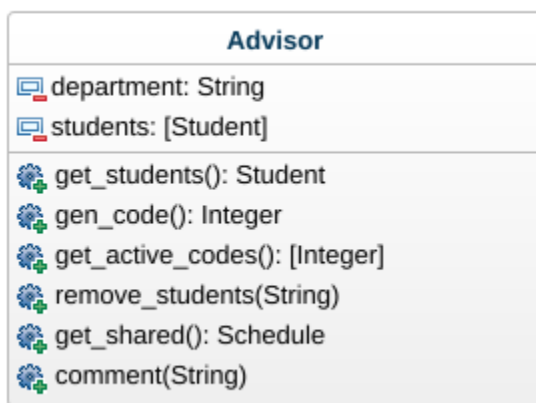
## Student Class:

The Student class is of type User, which means it contains all of the attributes within the User class combined with all of the attributes listed above. The majors and minors attributes are arrays of type DegreeProgram, which will be discussed more, because students can have more

than one major/minor combination, meaning multiple programs. We have a schedules attribute that is an array of Schedules since students are able to create multiple schedules and choose which one works best for them. Finally, the advisors attribute is an array of advisors because students can have multiple advisors if they have multiple majors.

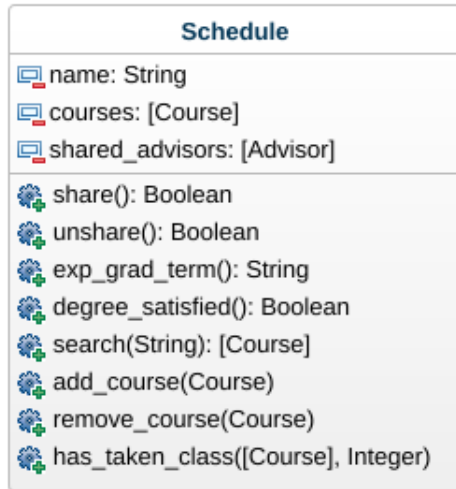
For the methods listed, `add_advisor` takes an integer as a parameter, since that is the code sent to the student from the advisor. `get_advisor()` is a basic get method. In addition, `share_schedule()` and `comment(String)` both have to do with the schedule functionality of the overall application and will be discussed in more detail.

### Advisor Class:



The Advisor class is of type User, which means it contains all of the attributes within the User class combined with all of the attributes listed above. Advisors only have two private attributes since advisors are based on the department in which they work and their list of students so that they can assist each student individually and know the primary information. However, advisors have more methods than the other user classes. Similarly to the Student class, Advisor has several simple get methods in order to see schedules that students have shared with them, they can get their students, and get codes to link their students to their accounts. Advisors can remove students whenever necessary. Finally, one of the main functionalities of the Advisor class is the ability for advisors to comment on the shared schedules and they can have a back and forth with the student, where they can make suggestions and approve/disapprove schedules, this is why the Student class also has a comment method.

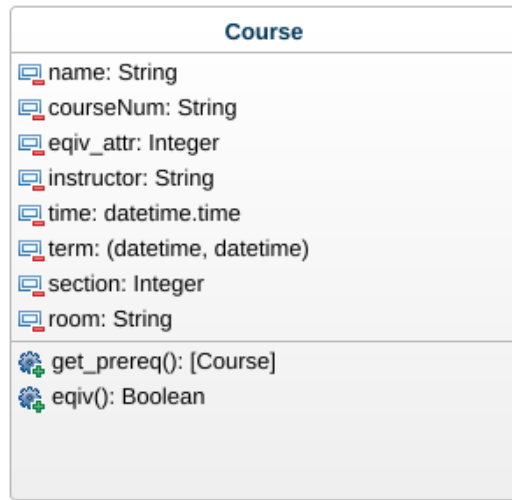
## Schedule Class:



This is the class that allows us to create the overall functionality of this application. The Schedule class is where all creation occurs in order for people to make their schedules. As stated previously, this application allows users to create multiple schedules on their account, so we allow users to choose different names for their schedules for organization purposes (private name attribute). The next two attributes have to do with who can see their schedules and which courses they have chosen to take for the given semester. In this class diagram it shows the courses has an array, but that's just how it's stored here, it will be displayed in a calendar like view on the homepage.

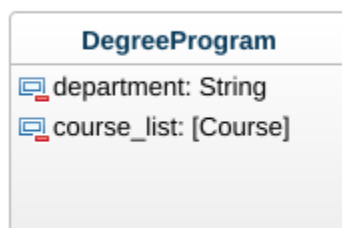
When building a schedule users need to be able to search, add, and remove classes with ease. In addition, there will be share buttons with each schedule and users can choose if they want one or both of their advisors to see their schedule if they have more than one advisor. The `exp_grad_term()` method tells the user when they should plan to graduate given their academic progress and `degree_satisfied()` tells them whether or not they have completed their given program(s) yet.

## Course Class:



This class contains all of the important information for all courses offered. It includes the course name with the course number/code followed by the instructor, time of the class and scheduled days, the term it's offered, the section number, and location of the course. The `equiv_attr` is our way of checking equivalence of classes across different departments. For classes that are equivalent to one another, each group will have its own integer to distinguish the difference between the groups. If a course has no equivalent, the field is left null. For example, some EE classes are equivalent to some CS classes so say the CS class you want to take it's offered at a time that works for your schedule then you can look at the EE offerings and see if that time/day works better. This is why there is also an `equiv()` method that checks course equivalence. Also we it comes to courses some require prerequisites, class completion before eligible for chosen course, so the `get_prereq()` method gets a list of the prerequisites of the course the user is trying to take and if all of them are already fulfilled then they can take the class, if not they will most likely take the prerequisites they are missing and take the class afterwards.

## DegreeProgram Class:

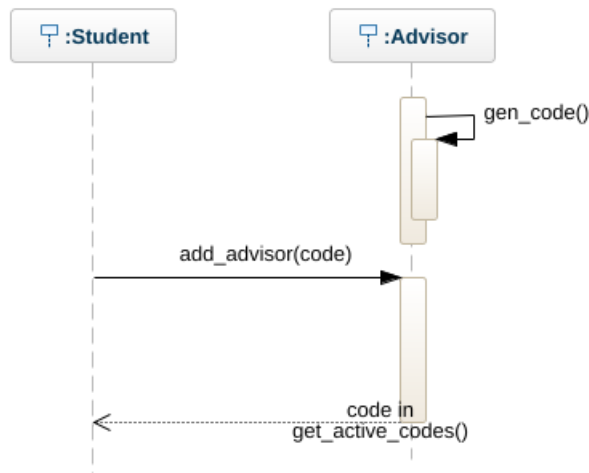




For this application we are assuming that majors and minors are the same thing, meaning they are both collections of classes that need to be taken in order to complete the program. With that being said the difference between the two is the amount of class/credits needed to complete them. Therefore, the Degree Program class only consists of two private attributes, the department of the chosen major and/or minor and the list of classes associated with it.

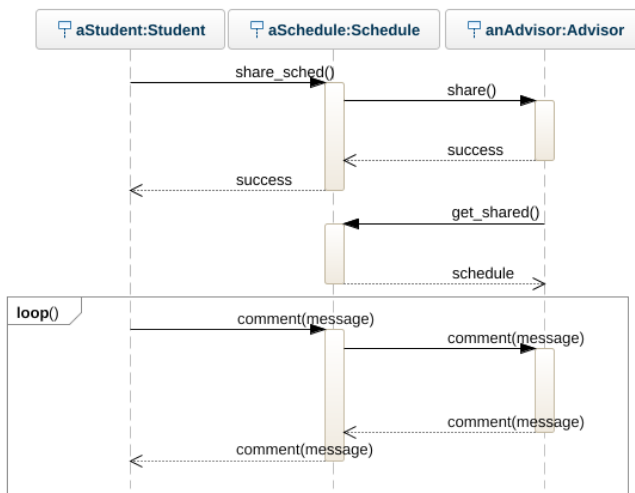
# Sequence Diagrams

## Linking students to their advisors:



This sequence diagram shows how advisors link themselves to their students. Advisors are able to generate a one-time code to send to a student, then the student is able to enter the code and see their advisor on their profile and now they can share and comment on things when needed.

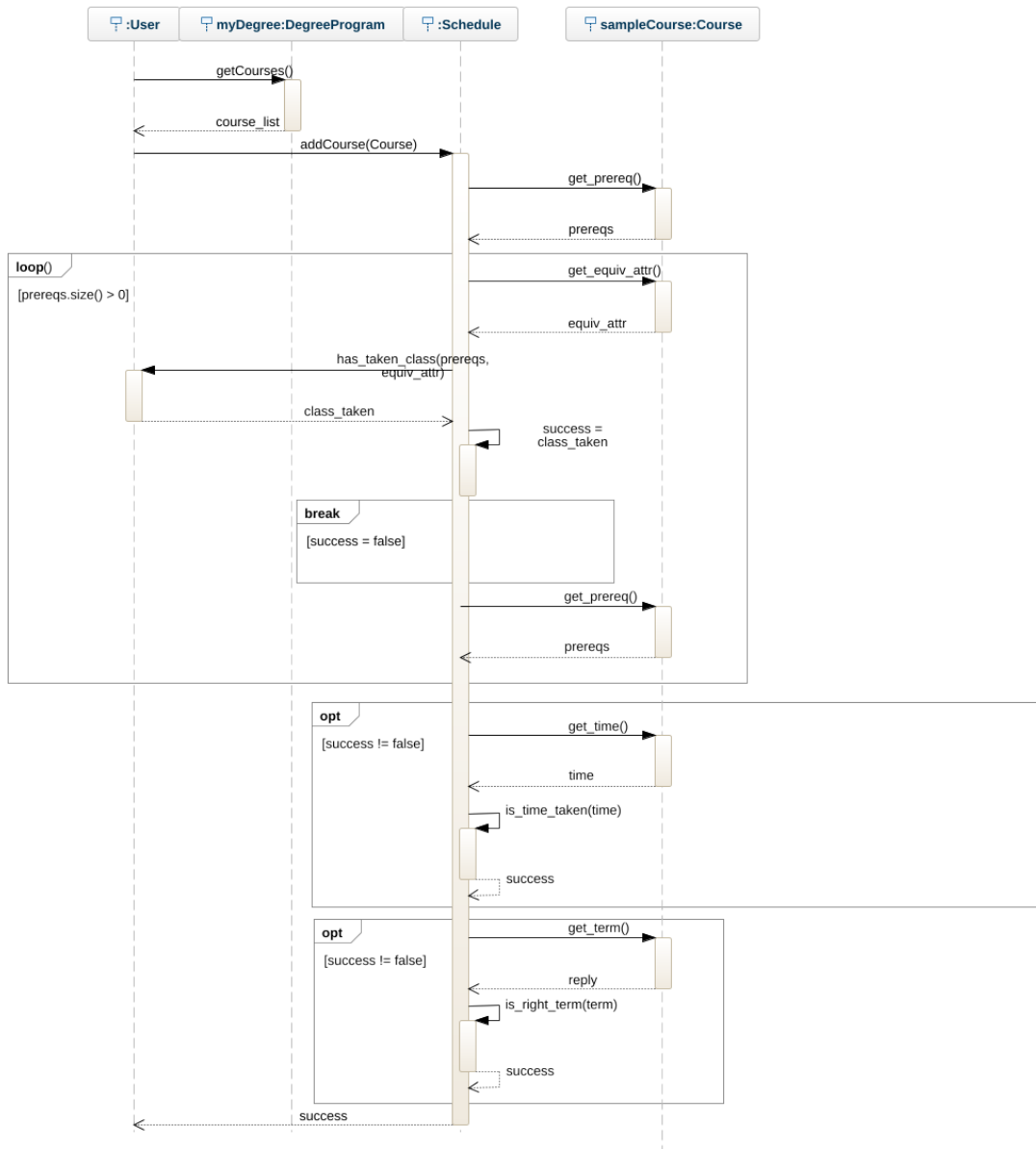
## Interactions between students and advisors with shared schedules:



Students have to give their advisors permission to see their schedules, if they have two advisors they can choose if both of them see it or only one of them. After the schedule is shared,

advisors and students can communicate back and forth via comments regarding the schedule. There will be a commenting section next to the shared schedule, as shown later on in this document.

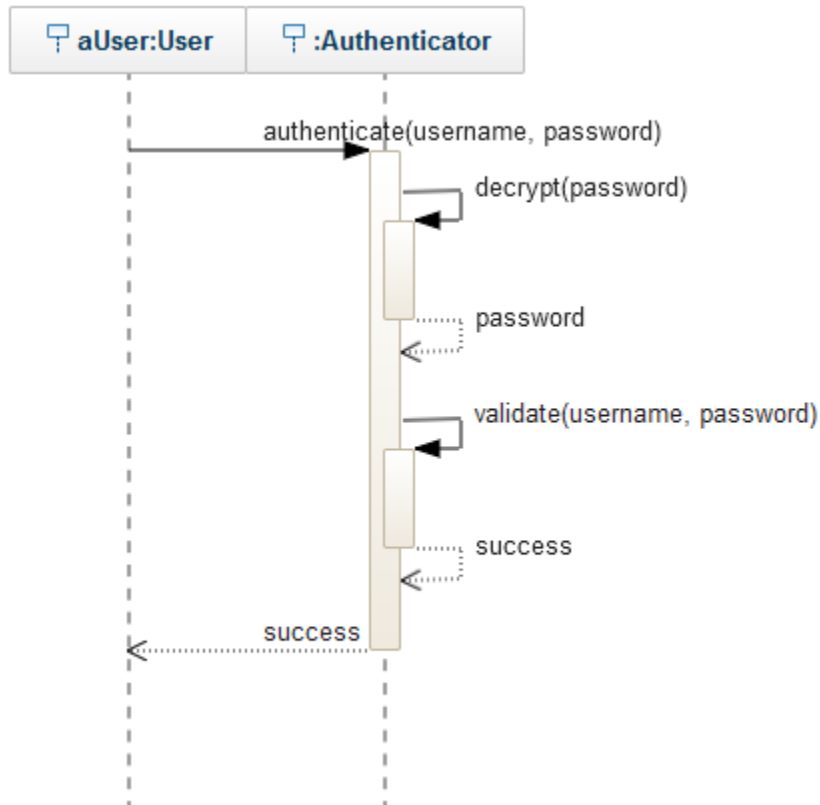
## Schedule functionality:



This diagram shows the most important feature of this application, schedule functionality. This demonstrates how students are able to build their schedule based on their chosen major(s) and/or minor(s) and the process they go through to add classes while seeing all of the relevant

information needed to make those decisions. If they have all of the necessary prerequisites and the times fit in their schedule they will be able to add their chosen course.

### Authentication:



This diagram depicts the authentication process. This depicts the username and encrypted password travelling across the network and being decrypted and validated by the Authenticator. The Authenticator then informs the user if the authentication attempt succeeded.

# Database Layout

The database layout overall closely resembles the classes from the class diagrams pictured above. Most tables will share the same parameters as their classes. However, there are some notable differences.

The User table is not the base of any student or advisor. This will be linked via a foreign key constraint at some point, but there is no clean way to describe it in the following diagrams. The User table still contains all of the relevant authentication information for students and advisors. Guests will not be able to save anything to the database, and therefore are omitted from the diagram. The User table will contain a username of type varchar(32), password of binary(60) and token is a varchar(64). The username is also the primary key of the table. The username and token are stored directly in the table, but the stored password is a bcrypt hash of the inputted password. The token is also nullable in case the user account has been created, but the user has not signed in before.

The Student table will contain a primary key id, the student's graduation term and the student name. The graduation term is stored as a varchar(6) in the example formats of fa2021 or sp2022. The student's name is stored as a string. All related majors, minors, advisors and schedules are related via different tables.

The Advisor table will contain a primary key id, the advisor's name as a string and the department id as a foreign key. The department id was added as a column because advisors are assumed to only be part of exactly one department. All links between advisors with students and schedules are located in different tables.

The Department table will contain a primary key id and the name of the department. The reason this is a table is because it normalizes the departments that show up in the Advisor table without needing to copy the string many times. This also provides a central location to rename the department in the future if needed without needing to propagate wide-spread changes.

The Schedule table will contain a primary key id, the name of a schedule as a varchar(64) and student id of the student who created it as a foreign key. The student id was added as a column because only one student can own a schedule that's been created, but it can be shared with multiple advisors. Since multiple advisors can potentially see a schedule, another table is used to store this relation.

The Course table is different from the Course class in the sense that the Course table in the database just contains the course id, course name, course number and equivalence attribute. This is because the same course can be offered multiple times in different rooms, at different times or by different professors. The course id is the primary key, except unlike most id keys, this id will be the one given by the university as a course id, rather than automatically assigned by the database. The course name is a varchar(64), the course number is a varchar(8) and the equivalence attribute is a nullable integer. The equivalence attribute works as described in the class diagrams.

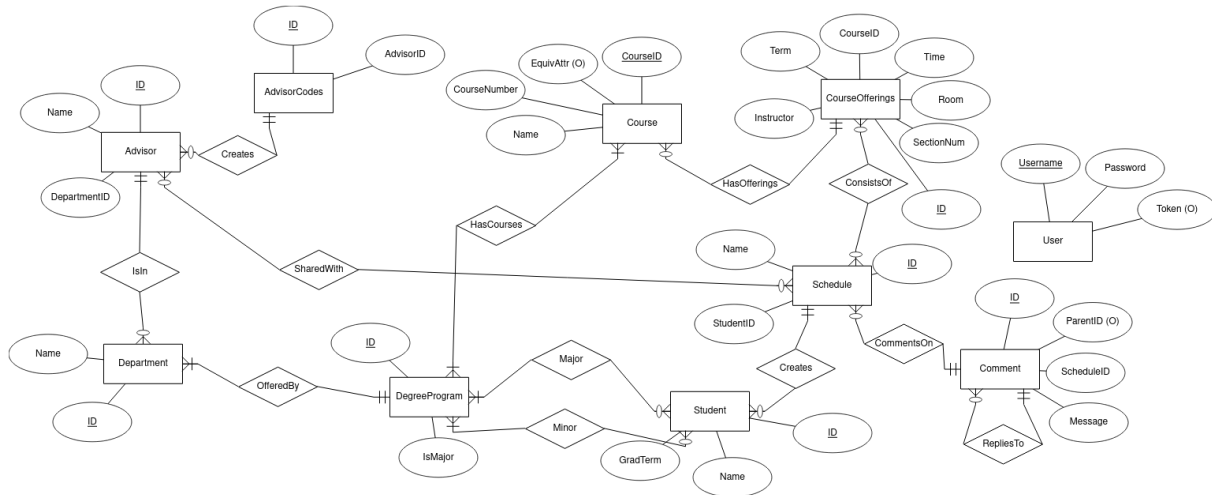
The CourseOfferings table works as if they were instances of a Course object in code. This class contains a primary key id, the term, instructor, time, room, section number and a course id foreign key. The term is stored as a varchar(6) in the example formats of fa2021 or sp2022. The instructor is a varchar(64) and does not link to advisors even if an advisor is an instructor. The time is a string which contains the days of week the class is held on, and the time. The room is a varchar(16) of the standard Clarkson room numbering format. The section number is an integer.

The DegreeProgram table contains a primary key id, boolean to determine whether it's a major or minor and a foreign key department id to distinguish which major this department is for. It is assumed there is only one department a major or minor can be a part of, therefore it was added as a foreign key column.

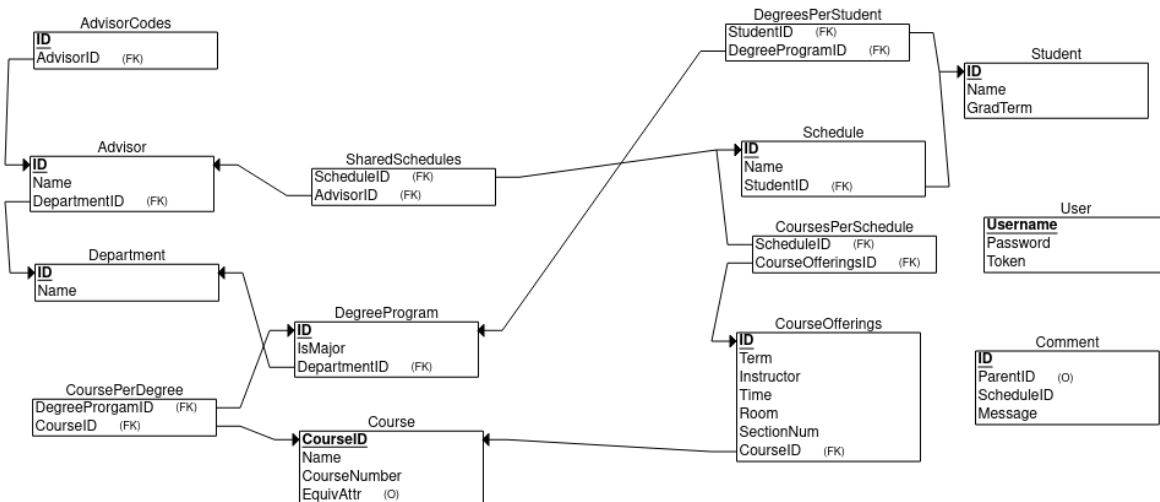
Last of the entity based tables, the Comment table contains a primary key id, self referential nullable parent id, a foreign key schedule id and a message string. The parent id is used so that comments can be replied to and it's possible to follow the chain. Currently, there is no way of determining whether a student or advisor sent the message, but that is an issue that will be addressed soon.

The following tables consist entirely of foreign key columns and map the many to many relationships between different entities in the database. The SharedSchedules table maps the schedule id to the advisor id in order to allow advisors to see their student's ids. The CoursesPerSchedule table maps the schedule ids to their course offering ids so that schedules may be built. The CoursePerDegree table maps the DegreeProgram id to the course id so that majors and minors can be laid out. Lastly, the DegreesPerStudent table maps the student id with the DegreeProgram they are a part of. This allows students to have more than one major or minor and of course more than one student can be enrolled in a single major or minor.

## ER Diagram:



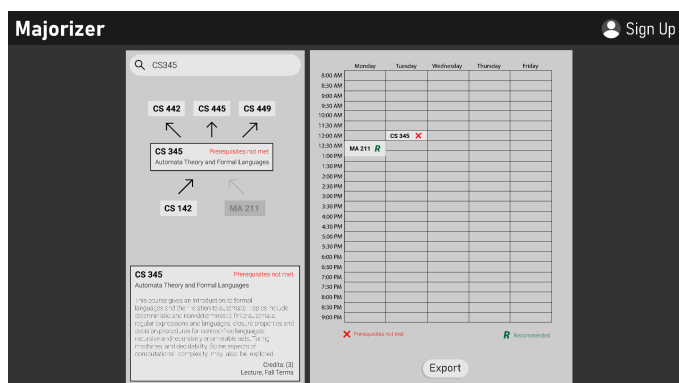
## Relational Schema:



# GUIs

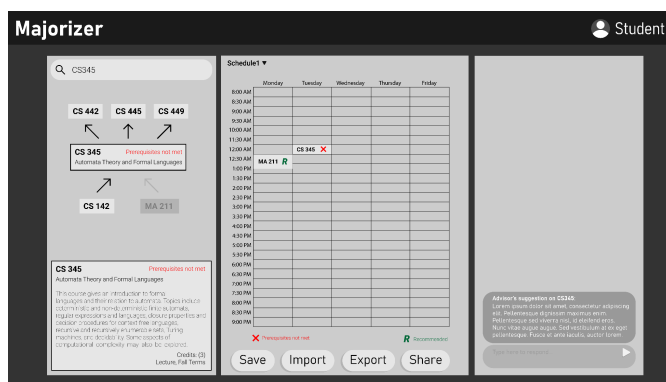
## Home page for guests:

Guests can make and export schedules by dragging and dropping classes from the class search to the schedule.



## Home page for students:

Students can make and save, import, export, and share schedules with their advisors. Students can also chat with advisors.



## Profile page for students:

Student's profiles store personal information, show linked advisors, add linked advisors, and show saved schedules.



## Majorizer

Personal Information

Name:John Smith

Graduation Year:2024

Majors:Computer Science

Minors:Mathematics, Digital Arts



Advisors

Advisor code:

Link

Edit

Saved Schedules

Schedule1	Classes:4	Comments:0	Shared:No	Valid:No	
Schedule1	Classes:5	Comments:0	Shared:No	Valid:Yes	


## Home page for advisors:





Advisors can view shared schedules from each of their students and comment on them. Advisors can also use the class search to view prerequisites.

[illegible]

### Students page for advisors:

This page shows all of the advisor's linked students. Advisors can add or remove linked students [here](#).

Majorizer  Profile

Student1	Majors: Computer Science	Minors: None	Class: 2023	
Student2	Majors: Computer Science	Minors: Digital Arts	Class: 2023	
Student3	Majors: Computer Science	Minors: None	Class: 2023	
Student4	Majors: Computer Science	Minors: Digital Arts	Class: 2023	

+ Add a Student