CMD Degree Guide

Ethan Cha, Haley Figone, Yaya Yao, Peyton Elebash

Software Design Specification

Table of Contents

1. Revision History **2**
2. System Overview **3**
3. Software Architecture **3**
3.1. Components 3
3.2. Interactions 4
3.3 Architectural Diagram 5
3.4 Rationale 6
4. Software Modules **7**
4.1. Authentication 7
4.2. Major/Course/Term Selection 9
4.3. Generative Algorithm 12
4.4. Degree Guide Display 14
4.5. Database 16
5. Dynamic Models of Operational Scenarios (Use Cases) **18**
5.1 New User - Create New Guide 18
5.2 Returning User - View Existing Guide 19
5.3 Returning User - Edit Existing Guide 20
6. References **21**
7. Acknowledgments **21**

1. Revision History

Date	Author	Description
10/07/2023	Peyton Elebash	Updated the title and names. First draft of 2.System Overview. First draft of 3.1. Components.
10/08/2023	Yaya Yao	Added more explanations in System Overview and details to Components.
10/08/2023	Haley Figone	Updated section 3.1, added tentative module headings to section 4.
10/20/2023	Haley Figone	Updated System Overview and System Architecture
11/20/2023	Haley Figone	Updated System Architecture again to reflect changes to course information in the database. Updated Modules, Interactions.
11/30/2023	Peyton Elebash	Updated section 2 and section 3.4. Wrote the design rationale and alternative design portions of all modules in section 4.
11/30/2023	Yaya Yao	Updated all static models and dynamic models.
11/30/2023	Haley Figone	Created charts for static and dynamic models from Yaya's written frameworks.
12/1/2023	Haley Figone	Finished use case models and finalized document formatting.

2. System Overview

The current DuckWeb degree guide system used by the University of Oregon is notorious among students for being difficult to read and interpret. Continued reliance on subpar technologies has made planning a graduation journey unnecessarily confusing for many; student soften additional support from advisors to ensure timely completion of their degree requirements. The CMD Degree Guide aims to streamline the degree planning process and empower students by providing them with a simple and clear path to graduation.

The CMD Degree Guide will be composed of a user log-in or create an account page, a page where the user selects their major from the three available options (Computer Science, Mathematics, and Data Science), the number of terms they have left, and the courses they have taken, an algorithm to generate a degree guide, a page displaying the generated degree guide, and a database containing course information and user login information as well as save state. The landing page will prompt the user to log in or create an account. If they are a new user, it will prompt them for the input information (terms, major, and courses). If the user has previously used the software, it will pick up from the last state.

3. Software Architecture

3.1. Components

Authentication (Login/Signup):

- ➔ Prompts user for their username and password.
- ➔ Displays an error message for invalid login or signup information.
- ➔ Upon successful login, redirects to the Degree Guide and creates a request to load the saved degree guide state from the appropriate user profile.
- ➔ Upon successful signup, redirects to the Major/Course Selection page for first-time setup.

Major/Course Selection:

- ➔ Prompts the user for their major (Computer Science, Data Science, or Mathematics)
- ➔ Prompts the user for their year and potential graduation date.

Generative Algorithm:

- ➔ Generates a possible degree guide for the user based on user inputs from the Major/Course Selection module.
- ➔ Degree guide is generated in the form of a two-dimensional array.

Degree Guide Display:

- ➔ Displays the degree guide as a table. The table is populated by mapping the output of the Generative Algorithm onto the cels.
- ➔ Displays a section of useful links to additional resources, such as advising, more specific academic requirements, and the course catalog.

Database:

- ➔ A database to store both static course information used by the generative algorithm, and user information used by the main web application.
- ➔ Divided into two tables: one for the course roster, one for the student information.

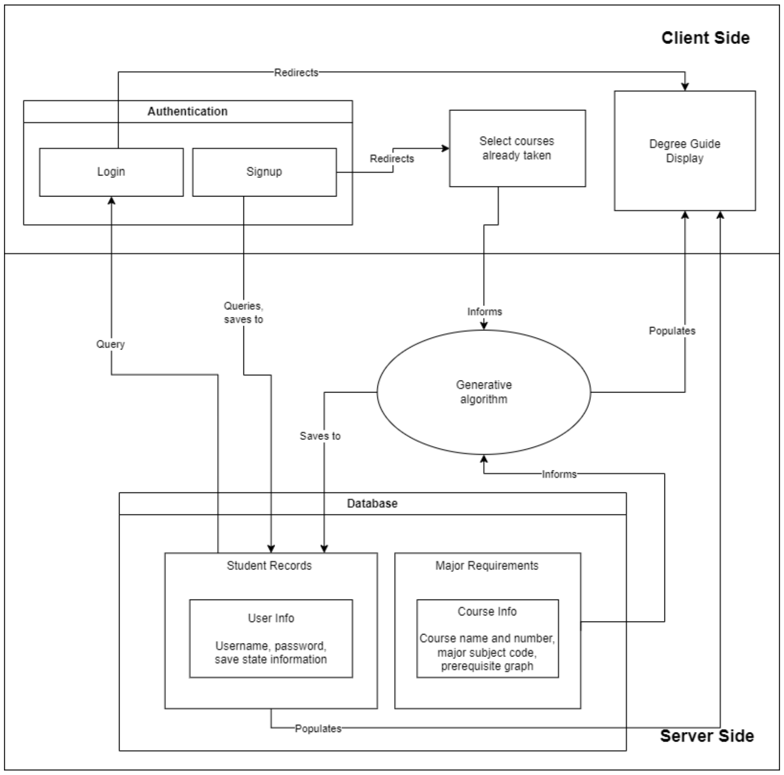
  Course Roster:

  - ➔ Stored as a table in the database.
  - ➔ Each entry is a course.
  - ➔ Each course contains the following information: Table index;
  - ➔ Course data includes course names, numbers, subject codes, and credit hours
  - ➔ General education requirements that do not fall under the supported majors will be represented by subject-group placeholders.

Student Records:
- ➔ Stored as a table in the database.
- ➔ Also contains save-state information from previous logins so that previously generated degree guides may be restored.

3.2. Interactions

- The authentication module accepts user input and sends a query to the database. The database informs the authentication module whether the provided input is already present and the authenticator handles the response accordingly, either by displaying an error or directing the user into the degree guide.
- On login, the app checks if a schedule exists in the student's records. Existing user data gets loaded into the degree guide display from the student records in the database if their schedule in the record is not null, otherwise redirect them to the input page to generate a degree guide
- The generative algorithm takes input from both the course selection module and the course roster as parameters for generating the degree guide.
- The degree guide display is populated by either the generative algorithm or from an existing degree guide in the database, if it exists.
- The generative algorithm saves its output to the student's account information in the database.

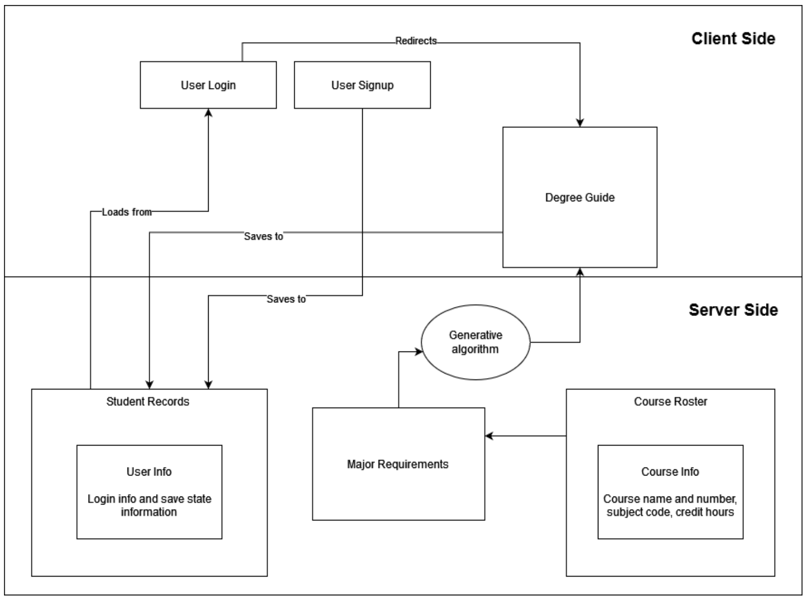


3.3 Architectural Diagram

The architectural diagram illustrates the components as described in Section 3.1, and their interactions as described in Section 3.2. Notably, login and signup are represented as distinct pages within the authentication module, due to the difference in their interactions with other modules. Similarly, the course roster and student records are represented as two separate table elements inside the database module.

3.3.1 Design Changes

The following diagram describes the first iteration of our software's architecture. The architectural design stayed mostly the same throughout, though we did make a few significant changes a short while into development:



1. The Student Records and Course Roster were originally conceptualized as two separate modules/"spaces"; most of us were unfamiliar with database setup and management, and as such we weren't quite sure how we were going to organize data at first. We ended up storing these modules as two tables in the same database. We made some representational changes to the architectural diagram to reflect this, redrawing the Student Records and Course Roster as two "submodules" of a larger "database" module.

2. We created a separate module for Major/Course Selection. Originally we considered this functionality as a part of the Degree Guide module. However, due to the risk of confusing the purpose of the Degree Guide module, and the dependencies between the modules (Major/Course Selection feeds data to the Generative Algorithm, which then in turn feeds data to the Degree Guide Display), we decided to separate it into its own separate module.

3. The requirements for each major were originally planned as their own separate module. The Major Requirements module would draw courses from the course roster and relate them using an adjacency list, representing a directed acyclic graph (DAG). The Generative Algorithm would then take that DAG and use a topological sort to create a degree plan.

4. Login/Signup are now correctly referred to as two pages in a single "Authentication" module.

5. Because of how small the functionality provided by the Major Requirements module would have been, it ended up being subsumed into the other modules. The courses stored in the database now have a piece of information indicating what major they are requirement for and what their prerequisite relationships are with other courses in the major (courses which are a requirement for multiple majors have multiple copies in the database, one for each major). The graph of the course prerequisites is thereby stored in the database itself. The Generative Algorithm then draws the courses from the database based on the selected major and builds an adjacency list from the data it provides, which it then topologically sorts on.

3.4 Rationale

The CMD Degree Guide is a web application for students at the University of Oregon. The main goal is to allow students to easily seek confirmation they can graduate on-time as well as generate a tentative schedule based on their major's requirements. The schedule is generated after the student selects the courses they have taken, their desired graduation date, and their major (as of now either CS, Math (Pure Track), or DS majors only). The guide includes possible courses to take fall, winter, and spring of each year they have left until graduation. The guide is stored in a database along with the student's username and password, so that they can access it whenever they log-in to CMD. By using the Degree Guide, students will finally have a platform that provides them with a readable and easy way to confirm they are on the right track to graduate by their desired time.

4. Software Modules

4.1. Login/Create Account

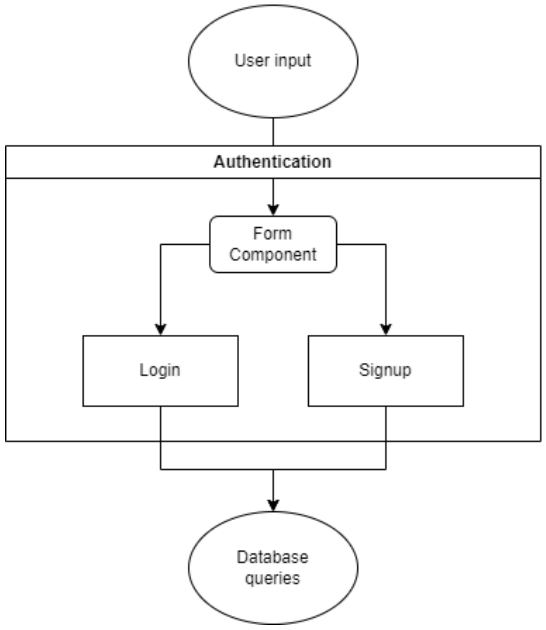
4.1.1. Authentication

This module authenticates the user, allowing them to log in to an existing account or create a new one. It redirects the user into the app on successful authentication and gives them an error on failure. The main purpose of this module is for the user to be able to load an existing degree guide if they have already created one, so that they can come back to look at or change it later.

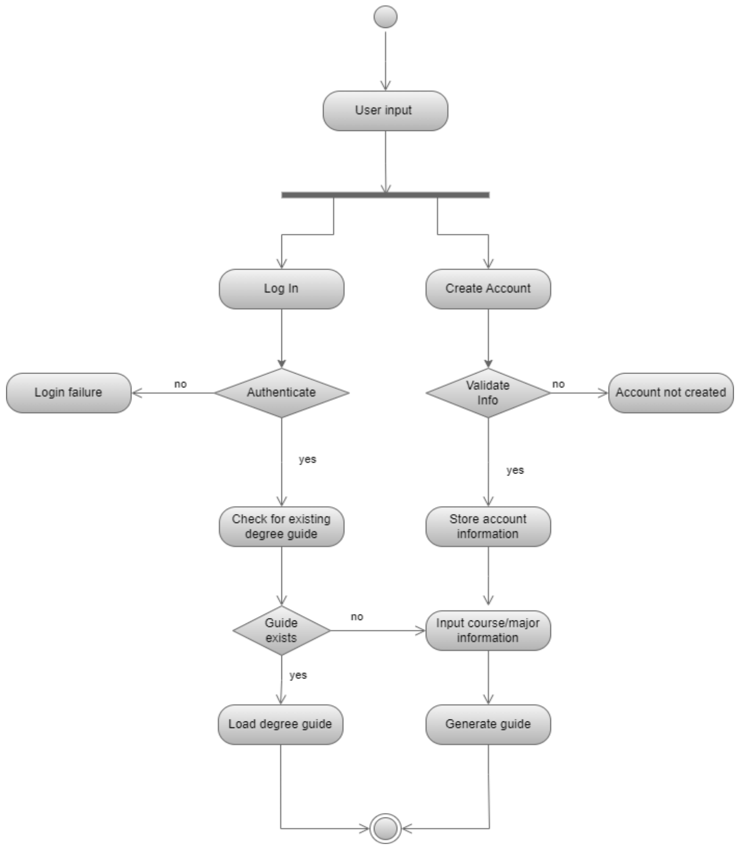
4.1.2. Interface

The interface for login/create and account is a simple HTML form. The user chooses one or the other via a link on the home page. The login/signup interfaces with the database via queries.

4.1.3. Static Model



4.1.4. Dynamic Model



4.1.5. Design Rationale

The login/create account module is a crucial element of the CMD web app. The user's information being stored in the database relies on the user providing information in the form of a username and password. The login/create account module also is crucial in determining if a user has used the degree guide before. The web app immediately prompts the user to either login or create an account. This makes a clear distinction on if the user has used this degree guide before. If the user is new, they will create an account and be prompted for a username and password. This username and password will be saved to the database after verifying it doesn't already exist there. If the user has used CMD before, they will be asked to provide their pre-existing username and password. This information is then sent to the database to retrieve their last saved state. If the username and password is not in the database, they will be prompted to try again. Overall, this design of the login/create an account module leads the user through the necessary steps to begin using CMD without overcomplicating the process.

The design for this module was discussed as a group. The group came up with an initial design (discussed further in alternative designs) together and then decided to split the work between three group members. Two worked on the frontend- the styling and layout of the login/create account page and one worked on the backend- connecting the input to the database as well as providing error checks.

4.1.6. Alternative Designs

Originally, the login/create account module was going to only prompt the user to login, with a smaller "If you are a new user click here to sign up". We changed this because we thought we should present the users with both options "Login" and "Create an Account" (instead of sign up) so as not to cause confusion for new users. New users if only prompted with "Login" may assume they have to go somewhere else to create a new account, and after not finding said place, they could give up and exit the CMD Degree Guide.

4.2. Major/Course/Term Selection

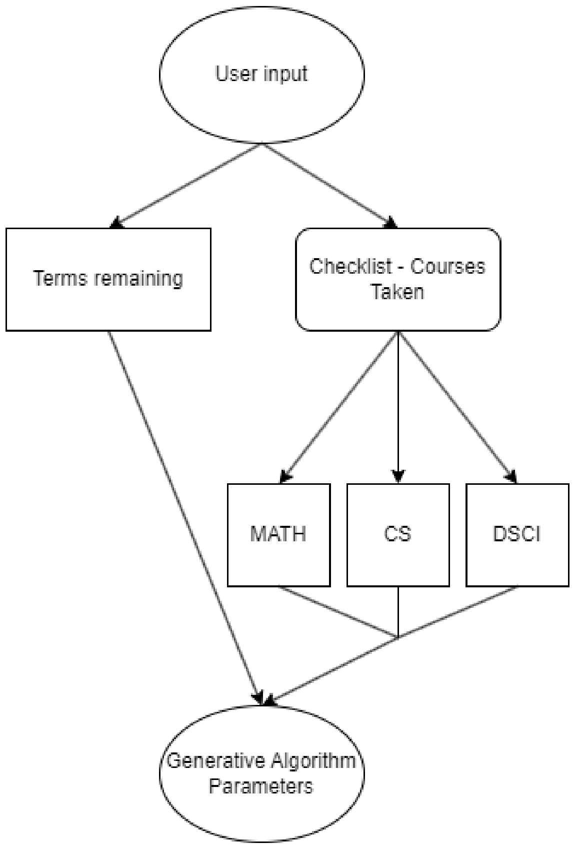
4.2.1.Primary Function

The major/course/term selection module allows the user to specify how many terms they have left until their graduation, what their major is and what progress they have made in it so far. This informs the output of the degree guide generator — the courses that are scheduled are, obviously, informed by the user's choice of major. Classes that have already been taken are excluded from the sort. It also informs the user if their current progress will allow them to graduate in their desired number of terms.

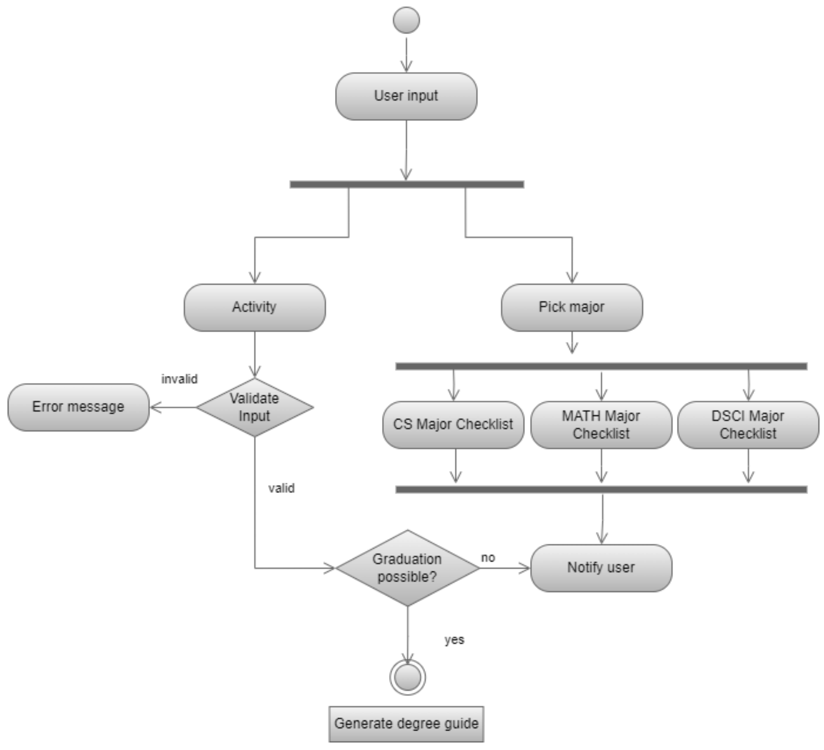
4.2.2. Interface

The user is initially presented with a text box allowing them to input the number of terms they have left to graduate — the text box rejects invalid inputs (negative inputs, inputs greater than twelve, non-integer inputs). The user must also select their major from a drop-down menu. Upon doing this, a checklist appears on which the user can click to select what courses they have taken already. Choosing a different major from the drop-down causes the checklist to change to the list appropriate for the major.

4.2.3. Static Model



4.2.4. Dynamic Model



4.2.5. Design Rationale

This module is only presented to the user if it is their first time using the CMD degree guide or if they choose to make changes after logging back in to view an 'old' degree guide. This design choice does restrict the user to generating a new degree guide each time they want to make changes. However, this restriction was intentional since it prevents the user from having too much free will and accidentally changing too many factors, which would leave them with an inaccurate and confusing degree plan.

The user is initially prompted to input how many terms they have left and their major. Then, a checklist of courses specific to their major appear. The courses only appearing after the major was selected was important in stopping the user from selecting incorrect courses- we worried if they had access to other major's courses it would increase the risk of them accidently clicking something not relevant to them.

The entire team constantly gave input on the design of this module, with frontend and backend both giving rough diagrams based on layout and what information was necessary. Two members developed the frontend aspect, while one backend developer weighed in on input based on what their generative algorithm needed and the other connected the input with the algorithm.

4.2.6. Alternative Designs

The design for this module was constantly changing/evolving as our project progressed. Originally, we wanted the user (after logging in or creating an account) to be immediately taken to the degree guide page, with input options above the table with the schedule. We decided this wouldn't work since in order to generate the schedule we needed their input first, and not forcing them to do so, would cause issues with even producing this page.

Our next solution was to not display the table and then have it generate once their information is selected above. This ended up being scrapped because we thought it would be easier to just create two pages if the table wasn't going to be initially displayed anyways. We also realized that restricting the user from constantly manipulating the table could mitigate risks of them spamming incorrect information or misclicking- they need to confirm they want to make changes before doing so.

4.3. Generative Algorithm

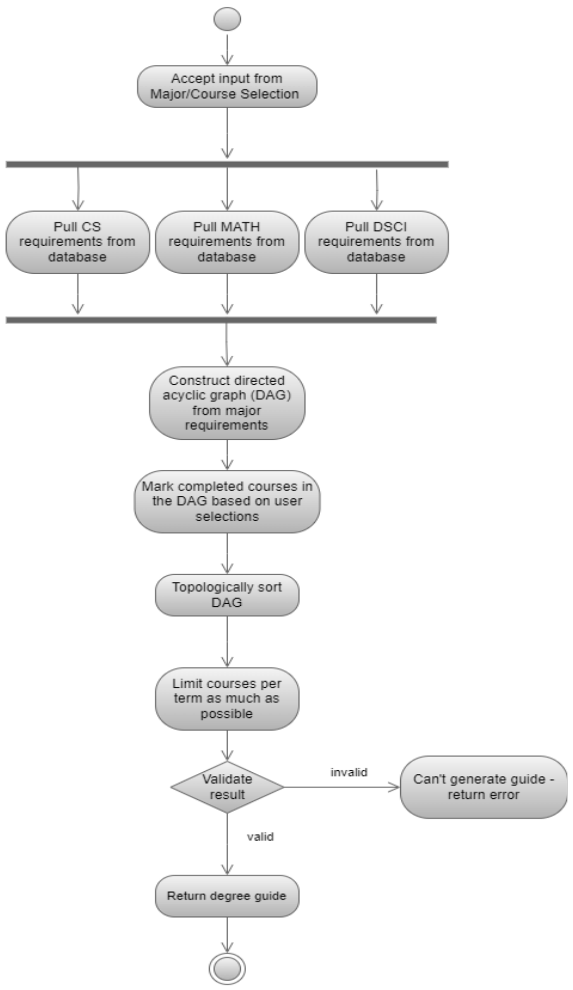
4.3.1.Primary Function

The generative algorithm performs the main sorting logic for generating the degree guide. It uses the information provided on the course selection page to find the classes from the user's major which they have not already taken. It then generates a term-by-term schedule for completing the major requirements in the specified number of terms. If such a schedule is not possible, it returns an error.

4.3.2. Interface

The user will select their terms left and the classes taken, which is passed to the generative algorithm. It is accessed through calling a runner function in the input page, which makes API calls to the server side given an SQL query to get information about the inputted major's classes and its prerequisites.

4.3.3. Dynamic Model


Accept input from Major/Course Selection
Pull CS requirements from database
Pull MATH requirements from database
Pull DSCI requirements from database
Construct directed acyclic graph (DAG) from major requirements
Mark completed courses in the DAG based on user selections
Topologically sort DAG
Limit courses per term as much as possible
Validate result
invalid
Can't generate guide - return error
valid
Return degree guide

4.3.4. Design Rationale

The generative algorithm is crucial in actually providing the user with a customized schedule based on their input. The goal was to ask for as little input as possible whilst providing the user with a schedule that:

1. Split courses into terms without violating prerequisites.
2. Did not have courses the user has taken.
3. Accounted for the terms the user has left until graduation.
4. Gave open slots for optional courses the user could decide to take.
5. Did not overload courses unless necessary.
6. Was as efficient as possible in determining a viable 'path' or schedule for the user.

The algorithm design is summarized as follows. The courses required for each major are stored in a DAG (directed acyclic graph) format in the database based on each course as well as what it is a prerequisite for. The algorithm grabs this DAG from the database depending on what major the user selected. The DAG is then modified based on what courses the user has noted as being taken. Then, a topological sort is enacted on the DAG, providing a path that doesn't violate prerequisites. After this, the path is sorted in terms, on the first run only putting two courses per term. If this doesn't work, then three courses are put per term, and if this doesn't work four courses are allowed per term. At the end, if there aren't enough terms for the courses left, an error is displayed to the user. If there is a valid schedule generated, it is sent to the table that the user sees. The entire schedule is generated as a 2D array, with the schedule being the main array, and each term being a smaller array within. This algorithm was created to achieve the above goals as efficiently as possible, and we found performing topological sort on the DAGs achieved this.

The group assigned two members to focus on this algorithm. One member created the algorithm and modified it throughout the project. The other member connected the database to the algorithm, fixed certain bugs in the transition from python to javascript, and connected the algorithm's output to the frontend (table).

4.3.5. Alternative Designs

Originally the algorithm was only able to return a single array of a valid path to take courses. This obviously needed to be adjusted, since the user can take multiple courses per term. We adjusted this by creating a method for sorting the original returned path into terms. After this, we realized the design had an overloading flaw- the algorithm would place as many courses as possible into the 'earlier' terms. We decided to update the algorithm's design again by creating a 'rule' that each term could have a maximum of four courses in order to solve this. This worked by limiting the term's course load to a plausible amount, but still would organize as many courses as possible without violating the 4 course maximum into the early terms. One final alternative design we changed was only having major requirements in each course. We decided it would be important to show the users that they could have more than just their major's requirements in the terms. We implemented an "optional" course that would populate any extra slots in each term.

4.4. Degree Guide Display

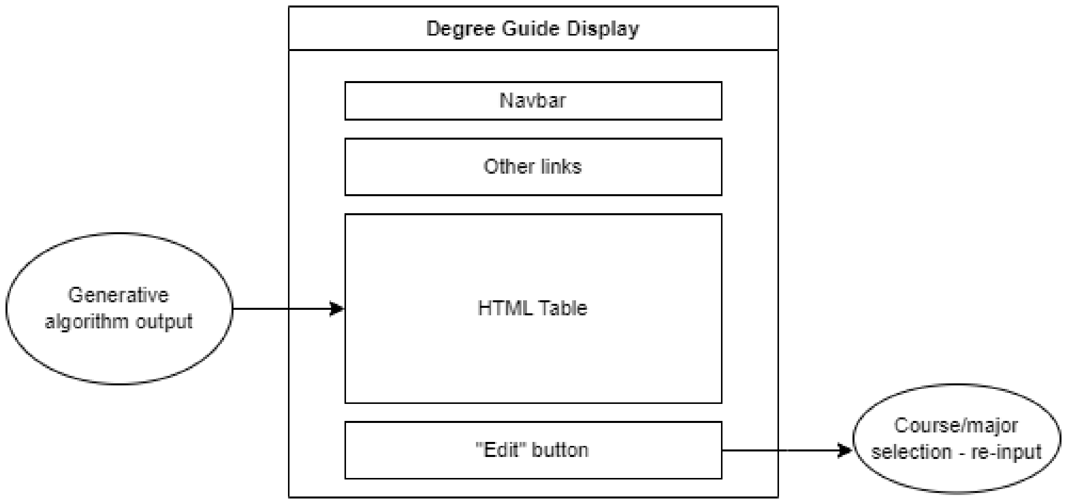
4.4.1.Primary Function

The degree guide displays the output of the generative algorithm in a tabular format upon successful creation of a degree guide. It also provides links to additional resources such as UO's advising website and the course catalog.

4.4.2. Interface

The user is faced with a table displaying a tentative schedule for them to follow based on their input from the major/course/term selection module. The user cannot interact with the degre guide itself, but they have the ability to go back and generate a new degree guide or to follow links detailing more information about university requirements.

4.4.3. Static Model


Degree Guide Display
Navbar
Other links
HTML Table
"Edit" button
Generative algorithm output
Course/major selection - re-input

4.4.4. Design Rationale

The design for the degree guide display module is a simple table that covers most of the page. This is to avoid the confusing nature seen in the existing UO degree guide. The goal of CMD is to provide as much necessary information as possible without overwhelming the user. The generative algorithm achieves this by providing a concise tentative schedule based on the user's information. This is then displayed to the table, which exists on a page separate from the input page in order to provide the user with a clear and directed, viewing experience. The links to additional helpful information are above the schedule, but are formatted in a way that still keeps the main focus on the generated schedule. These links allow the user to have easy access to helpful and relevant resources, but don't make the page too busy since we decided not to actually add in the information- just to provide a place to access it.

All members of the group aided in the design process of this module as well as in creating it. Two members focused on the frontend and organized the layout and design of the table to make it as user friendly as possible. Two members focused on backend, with one working on the generative algorithm (this populated the table) and the other worked on connecting the frontend and the algorithm.

4.4.5. Alternative Designs

Originally, as discussed in the major/course/term selection module, the degree guide was supposed to be displayed with options to alter it above. We changed this to mitigate risks on the user's end. Another factor we had considered was adding a drag and drop option to allow users to move courses to the terms they wish- however, this ended up not being plausible for our group. It gave the user the ability to violate prerequisite restrictions (which would have caused too many bugs) and given the time constraints, was too complicated of a feature to add.

4.5. Database

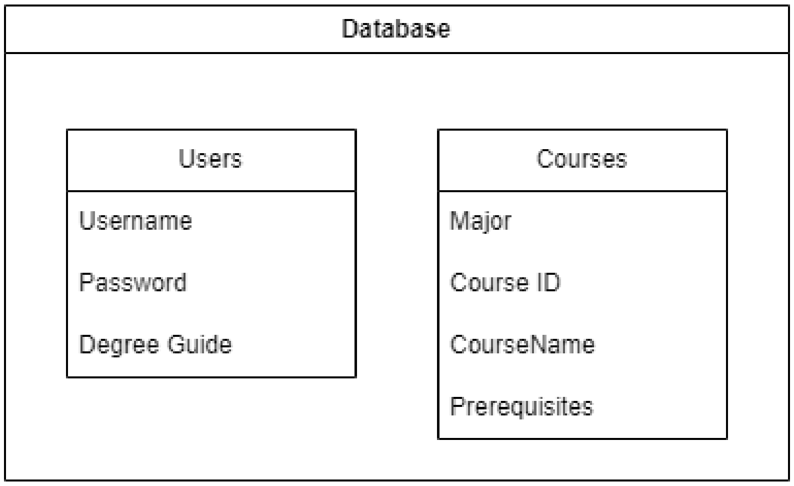
4.5.1.Primary Function

The database stores the information for user accounts (username, password, and their saved degree guide) and the information for courses and the sets of major requirements to which they belong.

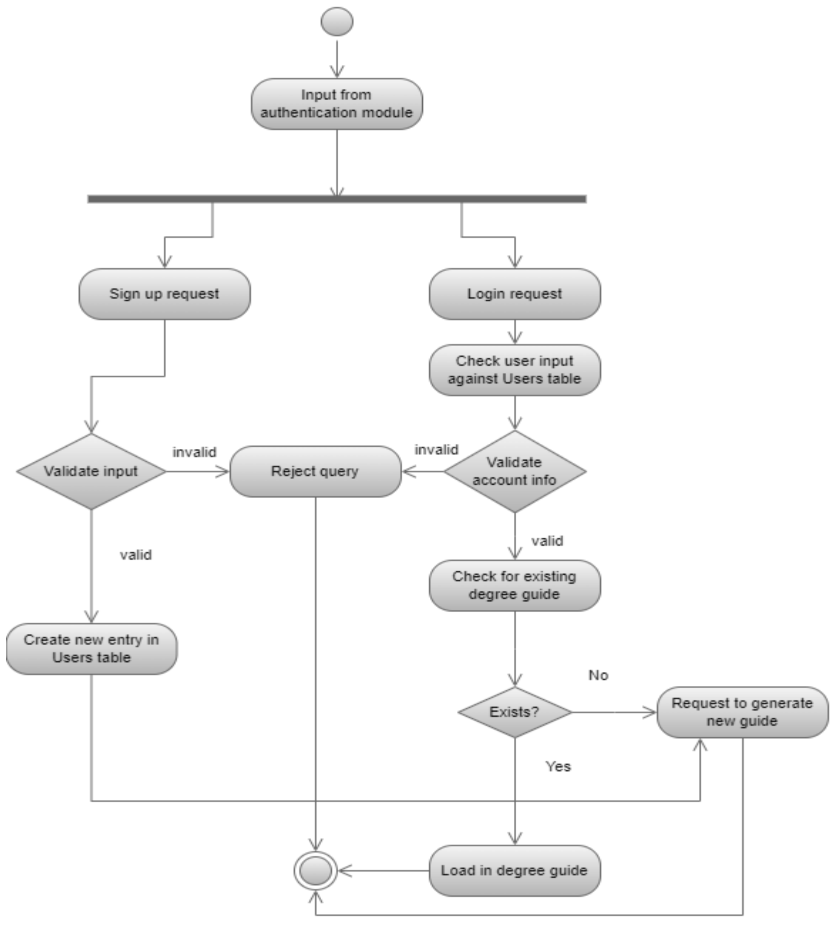
4.5.2. Interface

The user does not have access to the database, but the developers do. It was accessed via mysql workbench and stored the information in tables with clear labels.

4.5.3. Static Model



4.5.4. Dynamic Model


Input from authentication module
Sign up request
Login request
Check user input against Users table
Validate input
invalid
Reject query
invalid
Validate account info
valid
valid
Check for existing degree guide
Create new entry in Users table
No
Exists?
Request to generate new guide
Yes
Load in degree guide

4.5.5. Design Rationale

The database consists of two tables. One table contains the required courses for each major, stored in a DAG format based on their prerequisites. This structure was chosen based on the generative algorithm. Since the algorithm used topological sort, a database structured as a DAG was the most useful. It also makes it much easier to add a new major to the degree guide. All that is necessary is gathering the required courses to graduate and storing them in DAG format in the database under the major's 'code' (i.e. CS for Computer Science). The second table is for storing the user's information. When they create a new account, they pick a unique username and password combination which is saved to the database. Once their degree guide is generated, it is saved to the database under their account. This allows the user to come back to the same guide the next time they log in.

One member of the group mainly worked with the database as they were most familiar with it. All members played a role in the design of it, however the two backend members focused more on its structure since it engaged frequently with the generative algorithm.

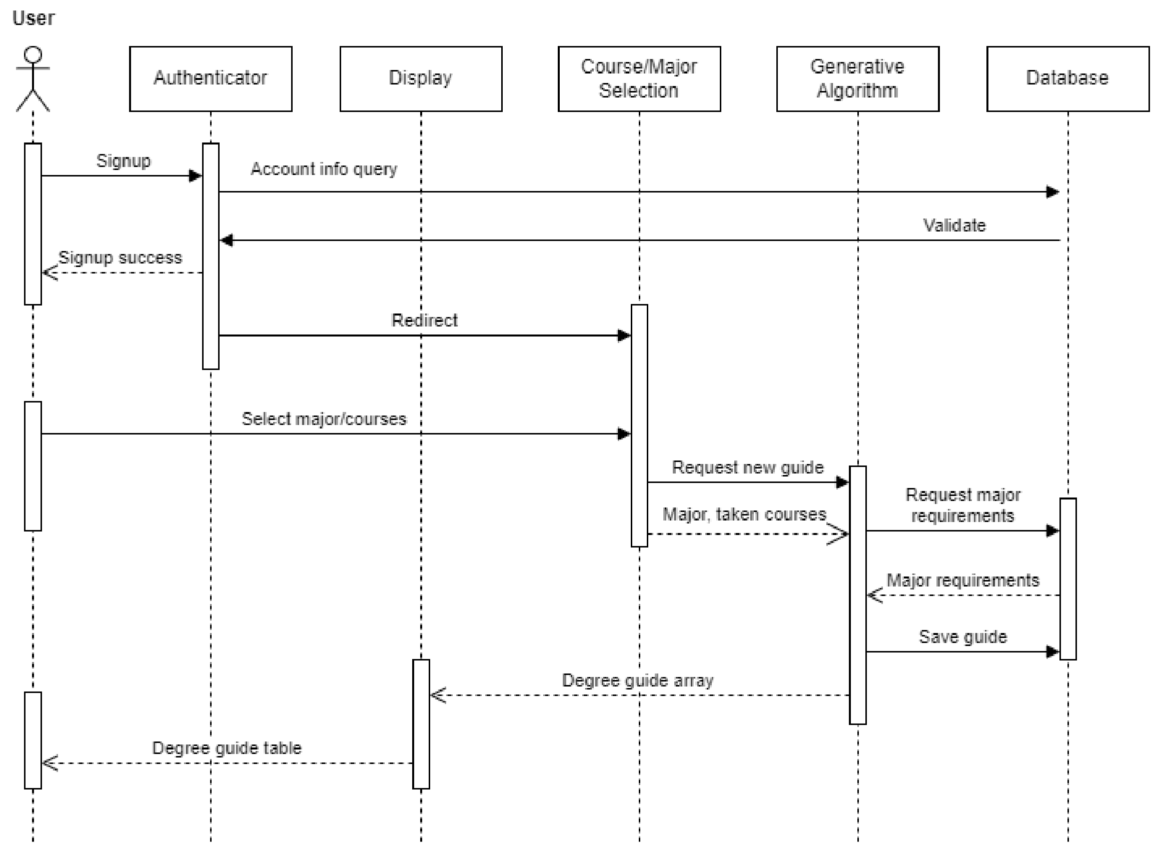
4.5.6. Alternative Designs

Originally, the database was not going to be stored in a DAG format. We instead planned to have the course listed along with its prerequisites. This ended up making the generative algorithm more expensive, since it would then organize this information into a DAG with the courses listed with what they themselves are prerequisites for.

5. Dynamic Models of Operational Scenarios (Use Cases)

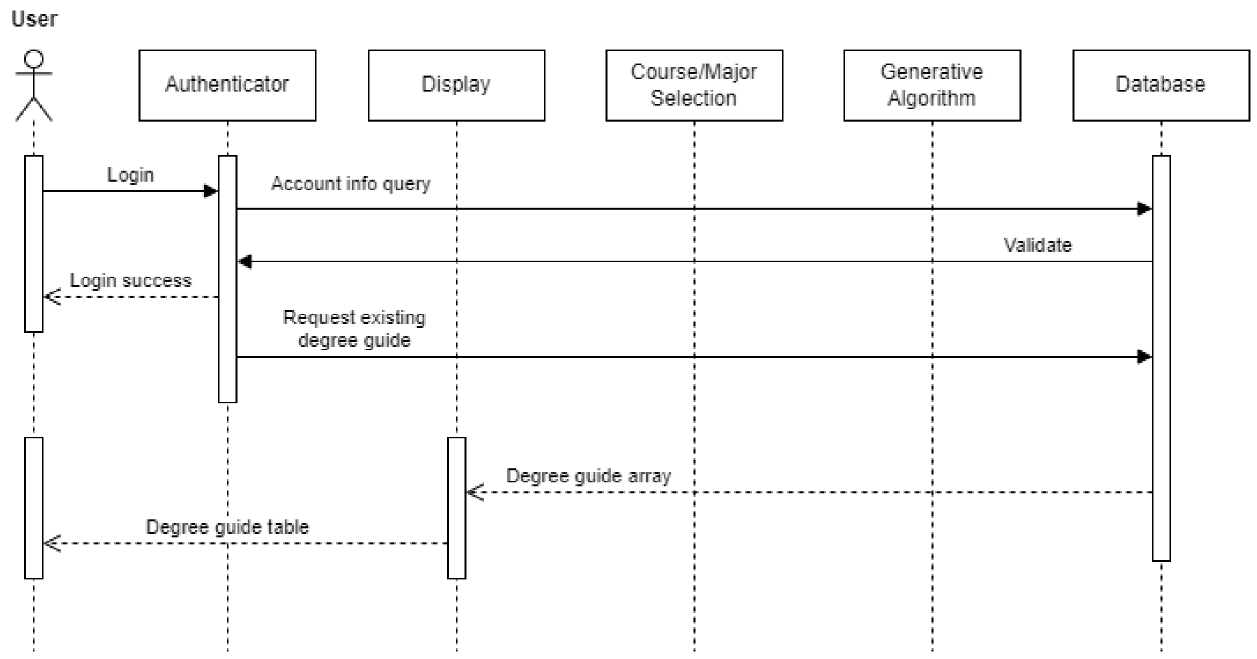
5.1 New User - Create New Guide

The first significant use case is that of a new user creating an account for the first time and generating a new degree guide.



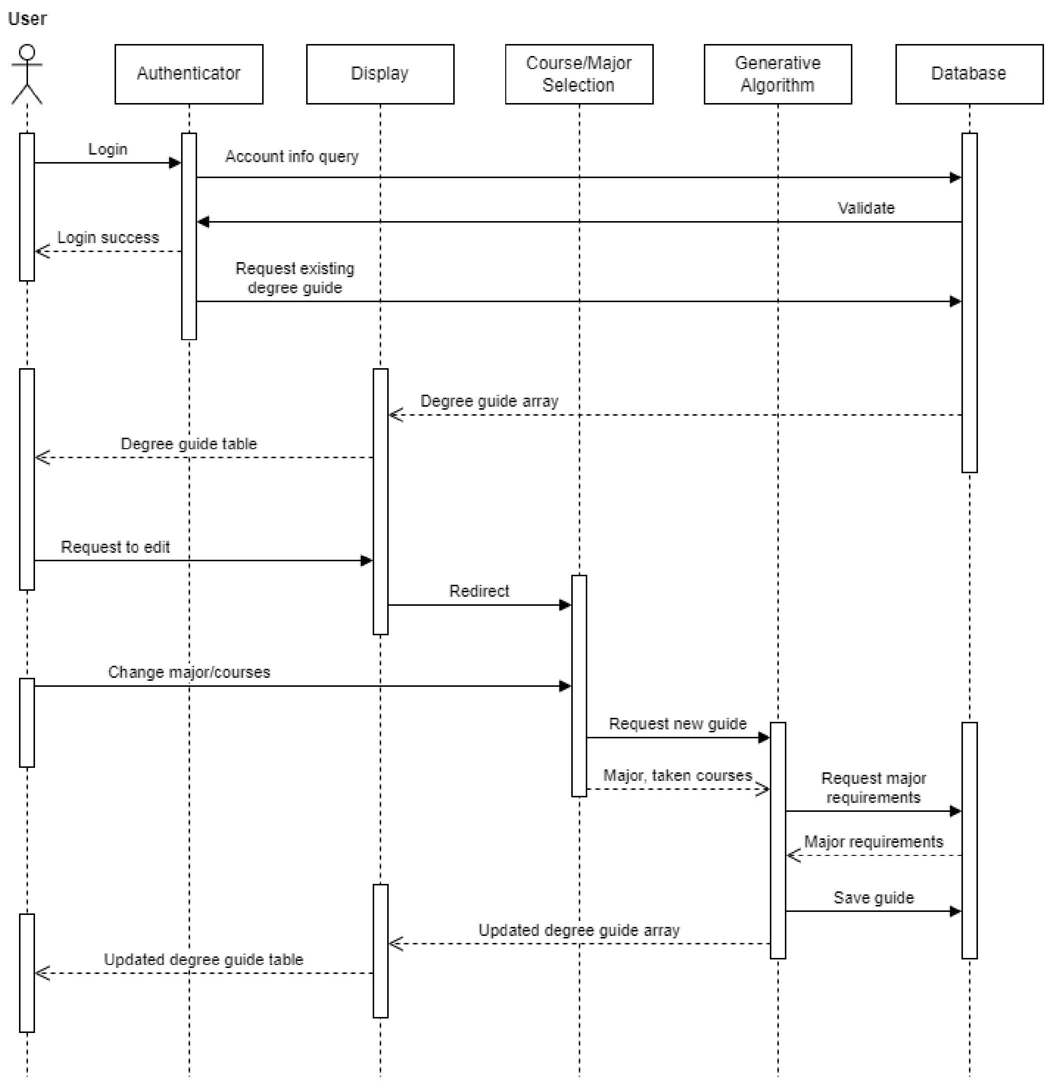
5.2 Returning User - View Existing Guide

The second significant use case is a returning user logging in to view their existing guide.



5.3 Returning User - Edit Existing Guide

The third significant use case is a returning user logging in to edit an existing guide.



6. References

IEEE Std 1016-2009. (2009). IEEE Standard for Information Technology—Systems Design—Software Design Descriptions. https://ieeexplore.ieee.org/document/5167255

Parnas, D. L. (1972). On the criteria to be used in decomposing systems into modules. *Commun. ACM, 15*(12), 1053-1058.

Class Diagram. In *Wikipedia*, n.d. https://en.wikipedia.org/wiki/Class_diagram.

Sequence Diagram. In *Wikipedia*, n.d. https://en.wikipedia.org/wiki/Sequence_diagram.

7. Acknowledgments

Special thanks to Prof. Juan Flores and graduate instructor Hammad Ather, whose feedback and advice was instrumental in our development process.