Building a Web-Based Dynamic Registration System for a College/University

(Design Document)

Team Members:
Andrew Stephens (Team Leader) - asteph11@oldwestbury.edu - 516-348-4044
Gwen Alessi - galessi@oldwestbury.edu - 703-477-9338
Brian Mejia- bmejia2@oldwestbury.edu - 631-885-4068

www.owsysdb.com

Table of Contents

1. Introduction

1.1 Purpose of the system

The purpose of the system is to function as a web based dynamic registration system for a University. The system will be a multi-user system split into Students, Faculty, Administration and Research Staff. Interaction with the system between user types is unique. Based on their user type, every user will be granted an appropriate level of access to specific information and functionality of the system. A user's level of access is defined by the business rules. The system is designed with intent to restrict access of information and functions to a user if they do not have the required permissions.

1.2 Design goals

The website is hosted by an AWS Ubuntu server instance, performing on the LAMP stack.The front-end UI is written in React with inclusion of JavaScript, TypeScript, CSS, and HTML.The back-end includes formation of the LAMP stack. With Node JS connecting the front-end and the back-end. The MySQL database houses an influx of data, including 2791 user records with their accompanying attributes, a collection of courses with their associated prerequisites, and a collection of programs with their associated course requirements. The database also includes a collection of stored procedures, utilized by the front end to get and set data for the UI. The data within the database was meticulously derived to be realistic and follow correct logic according to its grouping. Through these appropriate development practices, we expect to deliver a quality application that provides relevant functionalities for an intuitive user experience.

Our goal is to create a complex web-based registration system that is not only intuitive for the user, but also provides the proper restrictions between user types. Upon logging in, each user has their own home page specific to their user type. With each different home page comes different viewing and function permissions. The data manipulated upon the request of one user may affect another if they are associated with the same relationship. For example, if a student registers for a course section, then that student will be visible on the roster of the faculty who is teaching that course. Then, the faculty who is teaching that course section will be able to view that student on their roster, granted that they're viewing it within

appropriate times. In addition, an administrator will also be able to view this change to the course section roster. This provides a clear depiction of user purpose, and with designated functions, each user can focus on their personal intent for the system.

1.3 Data Preparation

1.3.1 Preliminary data:

The initial step in the process of data compilation was the gathering of preliminary data. This preliminary data came from many different places. Our courses, departments and schools are a modified version of the SUNY Old Westbury Course Catalog. The modifications came when we realized that inconsistencies existed within the course catalog and required alterations. For our Ph.D department, we used a select few departments and courses from Stony Brook University.

1.3.2 Automated Generated Data:

The automated generated data used in our project extends only to the personal information of our users. Attributes such as names, addresses, cities, etc. are the only pieces of data that were generated automatically.

1.3.3 Manually Generated Data:

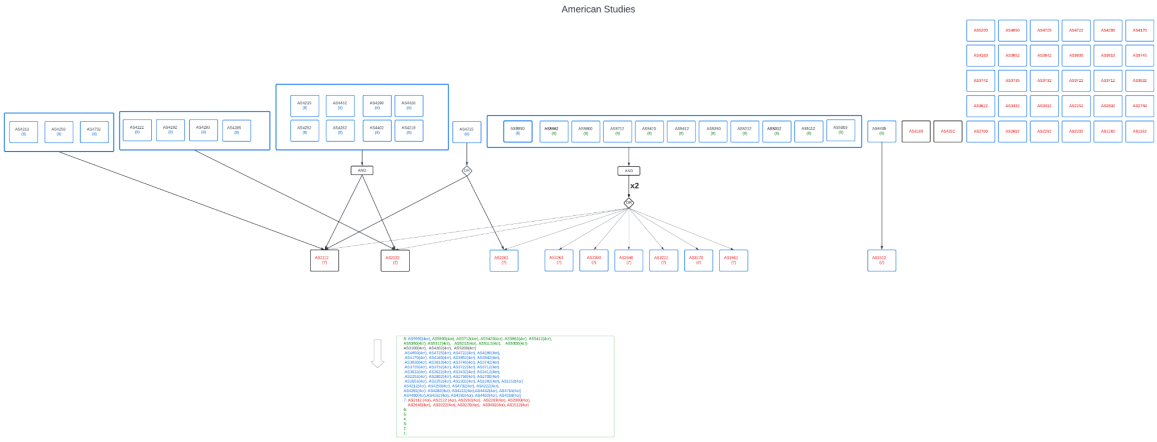
The data that was generated manually is what mainly comprises the atomic data of the database. This includes things such as student histories, semester dates, time windows, and anything having to do with the fulfillment of requirements for both courses and programs. Intense testing was required in order to assure the data was made correctly and fit into our vision for what the final database was going to look like.

1.3.4 Data Constraints:

The data that populates our database must adhere to the rules and constraints that were given to us by the client. This includes, but is not limited to: restrictions as to what courses can exist within a student's academic history, the dates to which attendance had already been filled and so on. These constraints allowed for more exact data to be made, however it also limited the way interactions between the website and the actual database were used.

1.3.5 Student Histories:

As stated above, the academic history of students was manually generated in order to assure it met the constraints of the student user. A process was created of drawing out the potential student histories for each degree program offered at our university, followed by creating rough outlines of all necessary histories for each type of user. Ranging from Undergraduate to Ph.D. level, the histories account for the largest chunk of data in our project at over twenty-one thousand individual records that were logged into the database. The following images will display what the process behind this data generation was composed of. Grades were developed with a Standard Deviation curve, with the mew at 0.86 and the theta at 0.3989.



The program requirement tree made for the American Studies degree program

H20

	A	B	C	D	E	F	G
1	Student Num	Student ID	Course ID	Grade	Term	Year	
2	1	308355	BU3071	B	Spring	2021	84
5	1	308355	BU3605	A	Spring	2020	97
6	1	308355	MA2300	B	Spring	2021	84
7	1	308355	PE2430	A-	Fall	2021	90
8	1	321427	PSY604	B	Spring	2021	84
9	1	381896	BU4500	B+	Fall	2021	86
10	1	308355	BU3600	B	Fall	2019	85
11	1	308355	BU3800	B+	Fall	2019	87
13	1	308355	BU5190	B	Fall	2021	85
14	1	308355	BU3511	B-	Spring	2021	79
18	1	308355	BU3905	B	Spring	2020	85
20	1	308355	PE2420	B	Spring	2020	85
21	1	308355	BU3502	B	Fall	2020	83
22	1	308355	BU4110	B	Fall	2019	85
24	1	308355	MA2080	B	Fall	2020	85
25	1	326934	MA1020	A-	Spring	2020	89
26	1	308355	BU4500	A	Fall	2021	93
27	1	308355	BU4510	B	Spring	2022	84
28	1	308355	BU5115	B	Spring	2022	84
30	1	308355	BU5555	B-	Spring	2022	81
31	1	308355	MA2000	A-	Fall	2020	90
32	1	326934	BU3502	B	Fall	2020	85
33	1	326934	BU3600	B	Fall	2019	84
34	1	326934	MA1010	B	Fall	2019	84
35	1	326934	MA2080	C	Fall	2020	74
36	1	396685	BS6560	B-	Spring	2022	79
37	1	308355	BU4570	B	Fall	2021	84
38	1	308355	BU4762	B	Spring	2021	82

The organization of students and the classes they have taken, including the semester completed and grade (numerical grade in the right column converted to letter grade used in the Grade column)

A1 | fx | Is

	A	B	C	D	E	F	G	H	I	J	K
1	Is used?	Course ID	Spring 2023	Fall 2022	Spring 2022	Fall 2021	Spring 2021	Fall 2020	Spring 2020	Fall 2019	LEFT Total
2		AS1152	6	0	5	0	17	8	6	0	42
3		AS1282	9	0	0	14	15	6	5	0	49
4		AS1512	0	20	5	14	8	5	0	0	52
5		AS2020	12	0	13	20	7	6	5	0	63
6		AS2112	9	33	22	22	20	11	10	0	127
7		AS2122	7	28	8	26	21	8	7	0	105
8		AS2202	0	0	0	5	11	6	0	0	22
9		AS2252	5	5	11	16	6	9	0	0	52
10		AS2262	15	23	15	12	7	0	0	0	72
11		AS2263	17	5	11	17	8	10	8	9	85
12		AS2300	11	5	8	8	6	5	5	7	55
13		AS2640	14	8	13	10	6	11	8	6	76
14		AS2652	6	7	14	8	0	0	0	0	35
15		AS2700	7	11	6	5	5	0	0	0	34
16		AS2750	0	5	0	0	8	0	0	0	13
17		AS2802	10	9	14	11	30	9	11	5	99
18		AS3100	14	10	0	0	0	0	0	0	24
19		AS3222	14	6	12	5	0	5	7	5	54
20		AS3247	5	5	6	15	0	6	0	0	37
21		AS3270	0	0	6	0	0	0	0	0	6
22		AS3310	15	9	5	0	0	0	0	0	29
25		AS3462	0	5	0	0	0	0	0	0	5
27		AS3632	0	5	5	0	0	0	0	0	10
28		AS3712	0	0	6	0	5	0	0	0	11
29		AS3722	0	0	5	0	5	0	0	0	10
33		AS3745	6	0	0	0	0	0	0	0	6
34		AS3800	7	12	5	26	6	9	0	0	65
36		AS3820	0	5	5	16	0	0	0	0	26

A count of how many students are taking a specific course in any given semester, alongside a total of how many times the course is being taken overall

1.4 Criteria

1.4.1 Usability:

Our project takes into account four different types of users: Administrator, Faculty, Student and Researcher. The website was designed to fulfill the multiple needs of these users, from registering for a course, managing the courses being offered, and viewing data based on the statistics of the students. The ease of use for all of the users listed above is assured through testing and the implementation of many quality of life features to the website.

1.4.2 Performance:

Performance optimization is something that was taken into consideration during the implementation of this project. Actions such as pagination when loading a page that contains a larger chunk of data, an elastic IP in order to maintain a static IP address, and limitations in the return of data from the database when making a call from the front end.

1.4.3 Extendibility:

The website and database were created in a forethoughtful manner, aiming to use little-to-no hard coding in both the front end and back end of the registration system.

1.4.4 Modifiability:

While the basics of the website can be applied to any of the different user types we outlined above, the website also features a select few components that can only be viewed or interacted with by a specific user. Good examples of this is the unidentifiable data that is presented to the Researcher in place of the more easily identifiable data shown to the Administrator, Student and Faculty users.

1.4.5 Reusability:

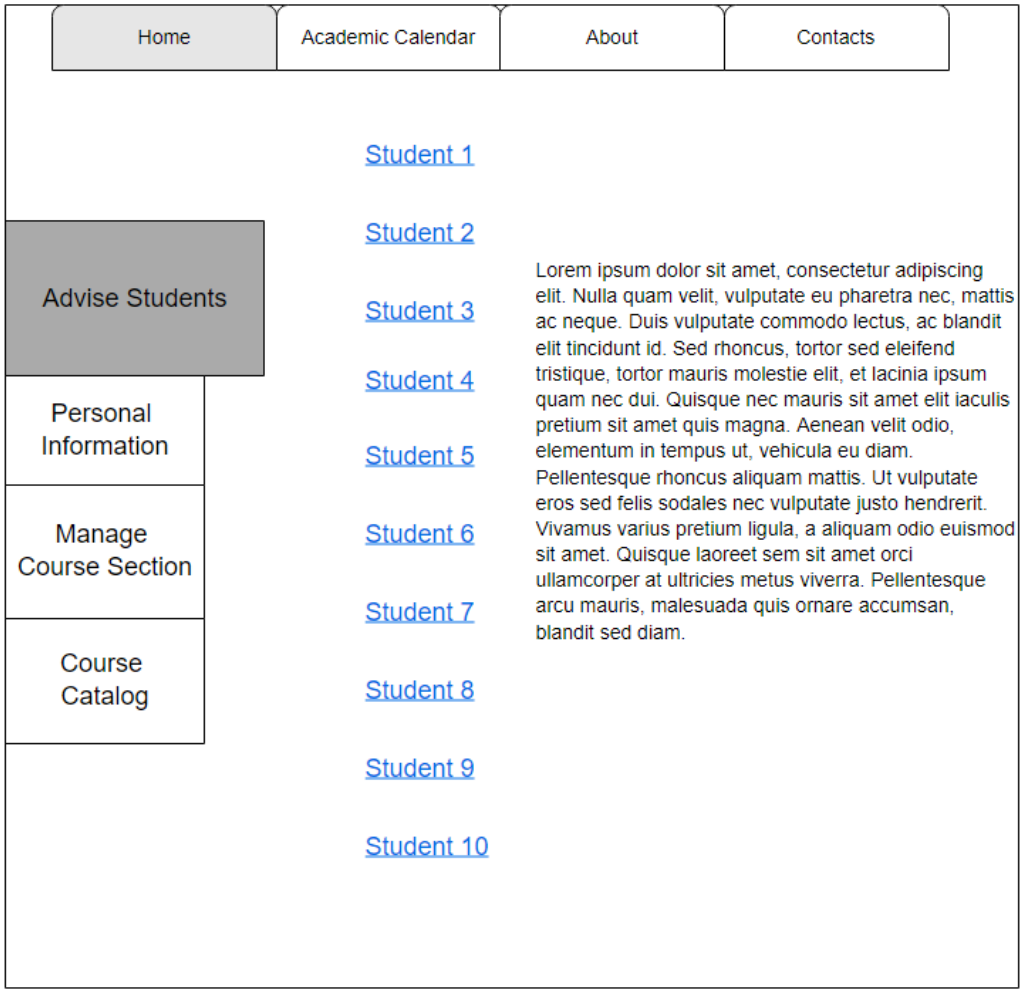
The nature of the project allows it to be reused for multiple semesters, however certain steps were taken to ensure that the registration system was functional beyond the test semester that we used. Stored procedures in the database are not hard coded with the current and next semester in mind. They are dynamic, returning data that is only relevant to the current timeframe that the user finds themselves in. Semester schedules, master schedules, attendance records and course sections are examples of that type of information. Of course, reusability is possible only if the database contains the appropriate data for future implementations.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `viewMasterScheduleBySemesterID`(IN Sem_ID char(3))
BEGIN
    SELECT s.Term, s.Year, c.CRN, c.CourseID, course.DepartmentID, uf.UID AS 'FacultyID', pi.FirstName, pi.LastName, b.BuildingID, r.RoomID, b.`Building Name`, r.RoomNum,
    t1.TimeslotID AS 'TimeSlotID1', t2.TimeslotID AS 'TimeSlotID2',
    d1.Name AS 'DayName1', d2.Name AS 'DayName2',
    p1.StartTime AS 'StartTime1', p2.StartTime AS 'StartTime2',
    p1.EndTime AS 'EndTime1', p2.EndTime AS 'EndTime2'
FROM Section_Timeslot st1
    JOIN Section_Timeslot st2
    JOIN Course_Section c
    JOIN Course course USING (CourseID)
    JOIN Semester s USING (SemesterID)
    JOIN Timeslot t1 JOIN Timeslot t2
    JOIN User_Faculty uf ON c.FacultyID = uf.UID
    JOIN PersonalInformation pi USING (UID)
    JOIN Day d1 JOIN Day d2
    JOIN Period p1 JOIN Period p2
    JOIN Room r ON c.RoomID = r.RoomID
    JOIN Building b USING (BuildingID)
    WHERE (
        c.SemesterID = Sem_ID AND
        d1.DayID = t1.DayID AND
        d2.DayID = t2.DayID AND
        p1.PeriodID = t1.PeriodID AND
        p2.PeriodID = t2.PeriodID AND
        t1.TimeslotID = st1.TimeslotID AND
        t2.TimeslotID = st2.TimeslotID AND
        st1.CRN = st2.CRN AND
        c.CRN = st1.CRN AND
        st1.TimeslotID < st2.TimeslotID
    );
END
```

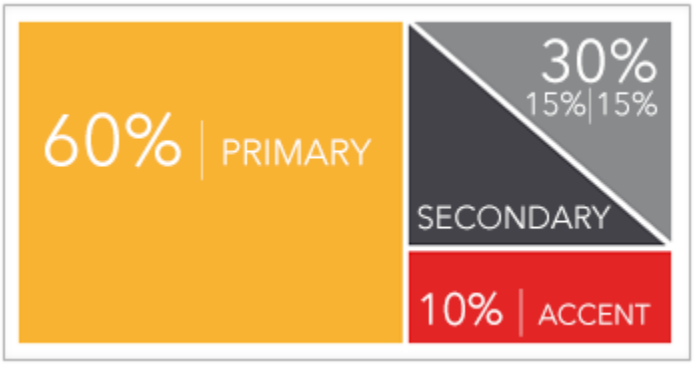
An example of a stored procedure used in the database to return the master schedule, using only the current semester passed in

1.4.6 Simplicity:

An aspect of the registration system that was high on the list of priorities was to make the website complex in its abilities, yet simple in its look and use. Small yet integral aspects, such as color scheme and button layouts, were among the first things drawn out and decided on before the implementation process began. UX Design rules, such as the “60-30-10” color scheme rule, were discussed early on and implemented in the proper ways later in development.



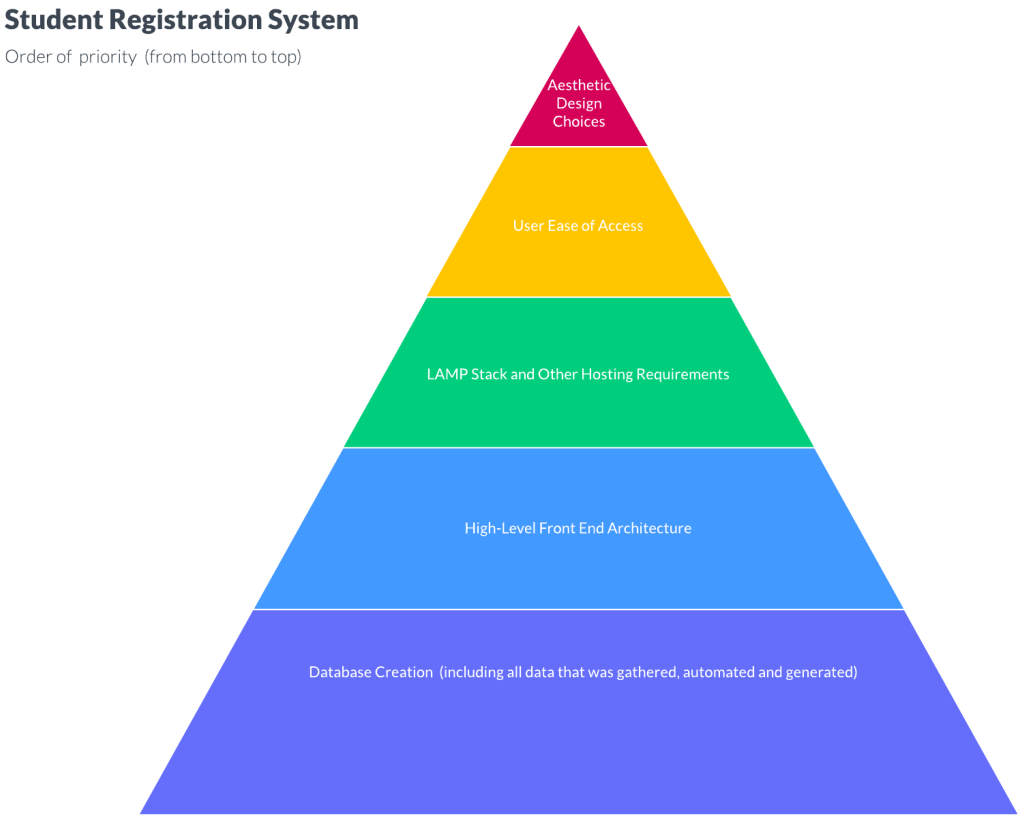
Early concept of website layout, taking into consideration the user experience and ease of access for multiple different components.



An example of the 60-30-10 rule used when designing the website color scheme

1.4.7 High Priority vs Low Priority:

When deciding on how the project would be completed within the allotted time, certain aspects of the project were deemed necessary for the overall structure of the project as something of a foundation to build the more delicate aspects on top of. The main concern was the integrity and reliability of the database, as most of the other functions of the website rely on consistent and accurate communication between the front end and the back end.



Visual representation of design priority

1.4.8 Dynamic:

Our system was designed to work with not just an upcoming semester, but any semester passed the current one that is being taken. Options and texts seen by the user are dynamically read from the database and filtered using user input. Little-to-no hard code was used in both the back-end and front-end of the application. Theoretically, this system will be viable for the foreseeable future, from Fall of 2023, onwards.

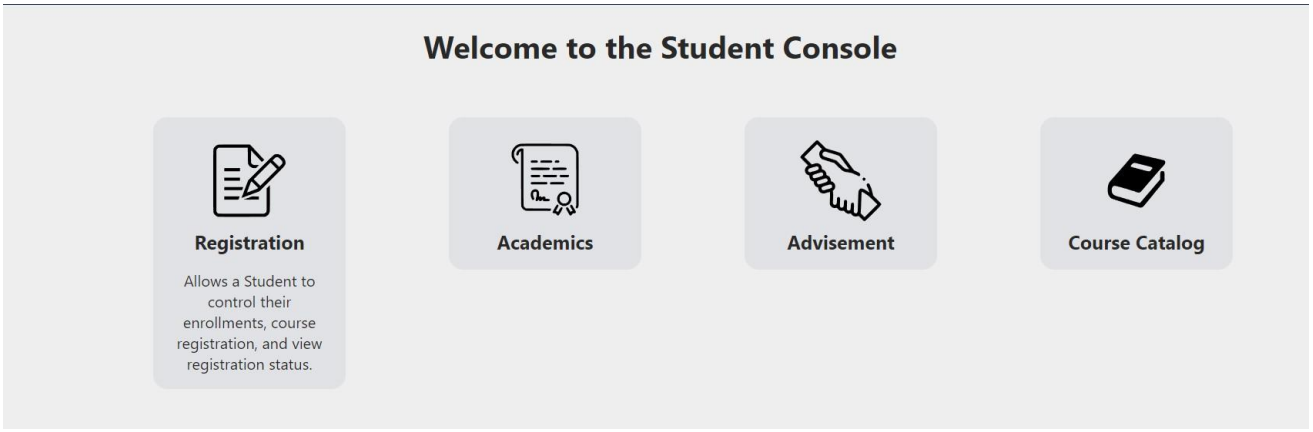
Views on the website, course schedules for students and everything else that a user interacts with can be updated on the database and displayed on the website when the time comes without the need to update code in the front-end or stored procedures in the back-end database.

2. High-Level Front-End Architecture

The Front-End Architecture is made up of three concepts: the Software of the Client side application, the User Interface which is created within the application, and the Client Device/Network that points to the AWS server. In order to design the User Interface, there must be a platform/server in which the data and functions are hosted. In terms of the User Interface, structure and design are taken into account when creating consistent interactions for the user. The front end Design, composed in React, consists of HTML and CSS script as well as a combination of TypeScript and JavaScript.

2.1 HTML/CSS

HTML is utilized for structure of the webpages, while CSS is used for styling of the webpages. With each web page design, a familiar layout/stylizing is taken into account. For consistency, there is some code that is repurposed in order to present each individual web page as part of the site.



In terms of reusable components,the drop down bubble animation is utilized in each user profile. The visual is created within the ConsoleHome.css file.

```css
.page-bubble-label {
    font-weight: bold;
    font-size: 20px;
    padding-bottom: 16px;
}

.page-bubble-description {
    font-size: 16px;
}

.page-bubble > .page-bubble-description {
    visibility: collapse;
    max-height: 0;
    transition: max-height .4s ease-out, opacity 0.4s ease-out;
    padding: 0;
}

.page-bubble:hover > .page-bubble-description {
    visibility: visible;
    overflow: hidden;
    height: auto;
    max-height: 700px;
    padding-top: 16px;
    transition: visibility 0s linear 0.2s, opacity 0.5s ease-out, max-height 2s linear;
    transition-delay: 0s;
    opacity: 1;
```

This is the creation of the user interface for these specific user console pages. To sustain a cohesive design, CSS styling is referenced each time a user console is presented.

2.2 TypeScript/JavaScript

The language of TypeScript, a derivative of Javascript, is used for functional programming. While CSS is used to produce a solid design for a component, the TypeScript files encapsulate HTML which then calls the specific components needed from the imported CSS file. Our CSS files work in tandem with the system to produce a functional webpage that has the correct data and relevant formatting.

```css
.pagination {
    display: inline-block;
    font-family: Veranda, serif;
}

.pagination .page-button {
    color: black;
    float: left;
    padding: 8px 16px;
    text-decoration: none;
}

.pagination .page-button .active {
    background-color: #4CAF50;
    color: white;
}

.pagination a:hover:not(.active) {background-color: #ddd;}

.MuiTableSortLabel-root.MuiTableSortLabel-active,
.MuiTableSortLabel-button{
    color: #eeeeee !important;
}

.MuiButtonBase-root.MuiTableSortLabel-root:focus,
.MuiButtonBase-root.MuiTableSortLabel-root:hover {
    color: #aaaaaa !important;
}
```

```tsx
import React, {Fragment, useEffect, useRef, useState} from "react";
import axios from "axios";
import '../css/PeudoTable.css'
import '../css/TablePagination.css'
import {TablePagination} from "@mui/material";

function DisplayAllStudents() {

    const [activePage, setActivePage] = useState(0);
    const [allStudents, setAllStudents] = useState([]);

    const [selectedStudent, setSelectedStudent] = useState<number>();

    const studentCount = useRef(0);
    const [paginationPage, setPaginationPage] = useState(0);
    const [maxPaginationResults, setMaxPaginationResults] = useState(15);

    useEffect(() => {
        const request = async () => requestAllStudents();
        request().then(r=> console.log("Completed Request", allStudents));
    }, [paginationPage, maxPaginationResults])
```

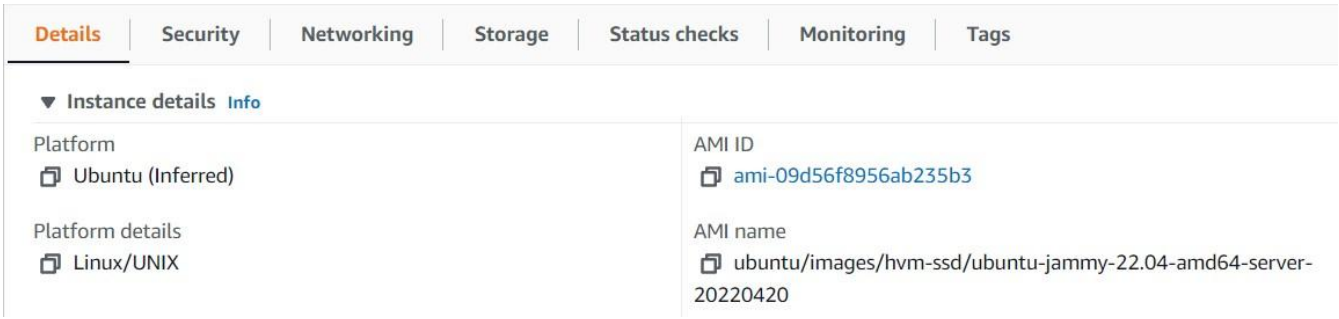
This is a TypeScript file for Displaying All Students. The File imports the relevant CSS files, i.e PeudoTable.css and the TablePagination.css that is pictured prior. DisplayAllStudents.tsx is able to utilize the CSS pagination design by calling the pagination components allowed from the specific import. The function DisplayAllStudents then accesses the requested data and reveals the information in a style of pagination to the user.

3. Subsystem Services:

The web based student registration system includes multiple subsystems, all working in tandem between the front and back ends. The system is composed of many subsystems, four of which are to follow: LAMP Stack, Dev Ops, Server, and Database.

3.1 LAMP Stack

Our implementation of the LAMP stack consists of the four components necessary to establish a fully functional web development environment.



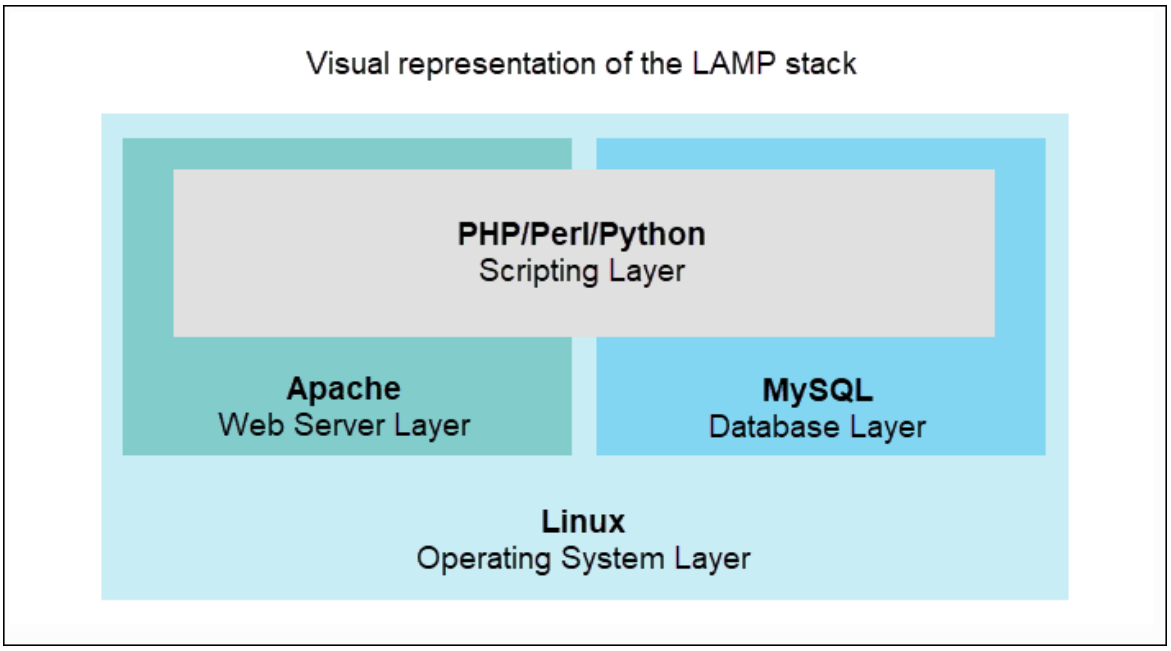
The running server instance.

Linux -(operating system layer) an operating system used to run the remaining components. Our instance of a Linux Operating System is presented by Ubuntu, a version of Linux. The Ubuntu OS is running on our AWS Server.

Apache HTTP Server -(web server layer) a web server software used to serve static web pages.

MySQL -(database layer) a relational database management system used for creating and managing the database

PHP - (scripting layer) language used for client-server communications through NodeJS.



Each component is used to create a layer of the stack.

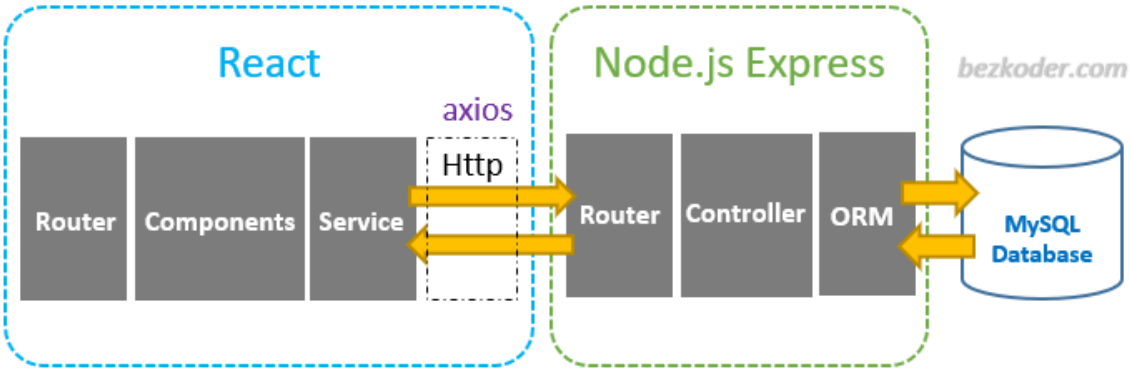
3.2 DevOps

3.2.1 What are DevOps?

DevOps is a set of practices that combine software development and IT operations (hence, DevOps). The purpose of this combination is to shorten the development lifecycle and provide continuous, high end software quality. DevOps and the Agile software development methodology pair well together in assuring that the process of development can be as smooth as possible for every member of the development team. Early in the planning for this project, the group decided to adopt the Agile methodology, as it is highly compatible with smaller teams that need to work in various different places at different times. Along with being the developers of the system, we must also act as the IT operators for said system by providing maintenance and any needed testing to it. These combinations of practices provide better communication, higher longevity and an increase in productivity for projects.

3.2.2 Node.js

Node.js is a JavaScript environment that allows developers to write command line tools using JavaScript. The main use of Node.js in our project is for communication between the database and the front end of the website. With a connection established to the MySQL database, Node.js can query the database by using any statements that would be used to query the database in the RDBMS. As will be explained in greater detail further below, Node.js works well with calls to the stored procedures saved within our database and uses them to pull data that is necessary for the views on the front end. This communication is reliable and efficient, requiring the connection to be established only once. After Node.js queries the database, the front-end can connect with Node.js via APIs in order to properly send that information to the user's view



Graphic detailing the role Node.js plays in the connection between the front end and the database

3.2.3 Creating The Website Using the LAMP Stack

The LAMP stack, which was explained above, is the basis of the web application that is this dynamic registration system. Linux, Apache, MySQL and PHP encompass the operating system, the web server, the database server and the programming language that many websites, including ours, are built with.

As explained above, each of these softwares works in tandem with one another to cover all the bases necessary for the project. Linux is the base that every other component in the LAMP stack rests on. Linux allows for the installation and configuration of these components to best fit the needs of the application.

Apache is the web server that stores the website files and exchanges information with the web browser using HTTP. When a web browser requests a web page, the Apache server receives the request, processes it, and sends the necessary information to the browser.

MySQL is the database server that hosts all the user information for our website and manages it. This information can be accessed by the website using queries. User records, student histories, course requirements, login information, and much more are all stored in the MySQL database. The database is a very large and heavily worked on aspect of our project and will be explained in greater detail further below.

Finally, PHP is the scripting language which allows the website to run dynamic processes. With PHP, the database, web server, and operating system all run cohesively to provide the dynamic experience that a web application such as this one requires.

All of these components are open-source, which allows for the making of a web application in a cost effective way.

3.2.4 React and Packages

React, which will be further explained below, is a JavaScript library that is designed to build user interfaces. React creates what is known as a single-page application, or SPA. An SPA loads only a single HTML document on a request and then updates a specific portion or body of the web page through JavaScript. Through React's interactions with the packages explained above, such as Node.js and MySQL, the website is dynamically changed without the need to load a new HTML document every time. Node.js exports REST APIs and interacts with the MySQL database using Sequelize ORM. The React Client then consumes these REST APIs by using Axios (a promise-based HTTP client). The React Router is then used to navigate the pages. React Router is a declarative component based, client and server side Routing library for React. With React Router we are able to provide correct and safe Authentication for our users. As previously mentioned, this aids in providing structure to what a user is allowed to see when they are logged in.

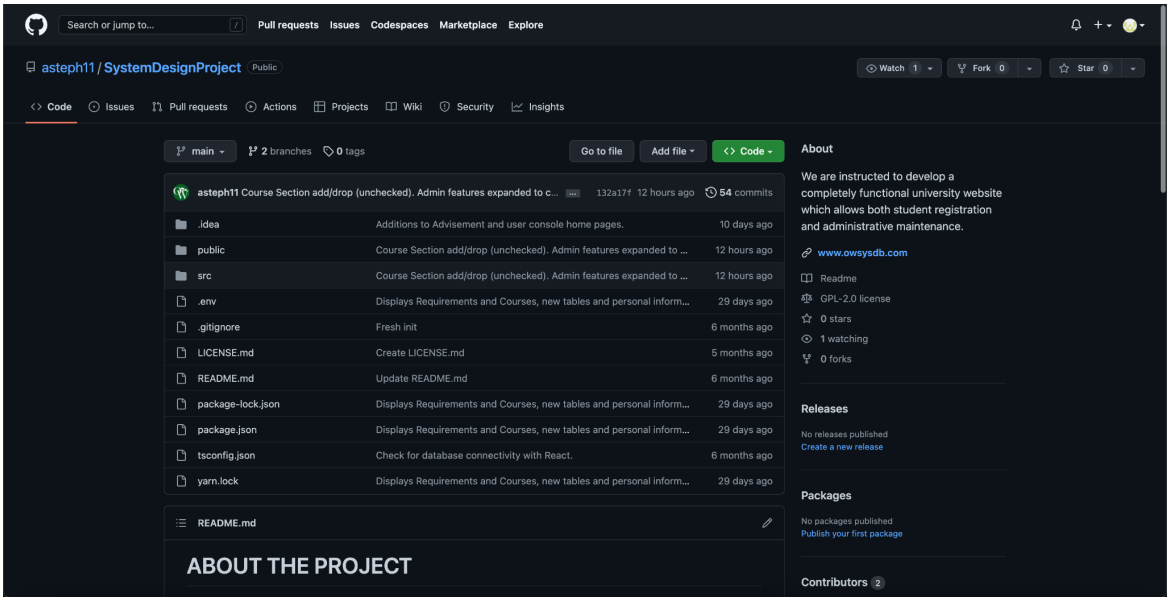
Another component utilized by and within React is Typescript. TypeScript is a superset of JavaScript that compiles to plain JavaScript. By using React and TypeScript

together, we build our UI using a typed version of JavaScript. Using React and TypeScript is beneficial because it is a statically typed language. Therefore, providing fewer bugs and more safety on the front end.

3.2.5 File Propagation

File propagation into the server is handled by FileZilla, which is an open-source FTP (file transfer protocol). Through a secure and protected connection to the web server, files that are written in an IDE such as IntelliJ can be placed into the folders that house the web page documents.

Beyond FileZilla, Git Bash is used for file propagation as well. Git Bash is used for pushing and pulling files from the GitHub page (where the code is housed and properly organized). Using Git Bash, code can easily be shared amongst all members of our group. It can be reviewed, modified, or deleted, with proper documentation and record keeping of every change made.

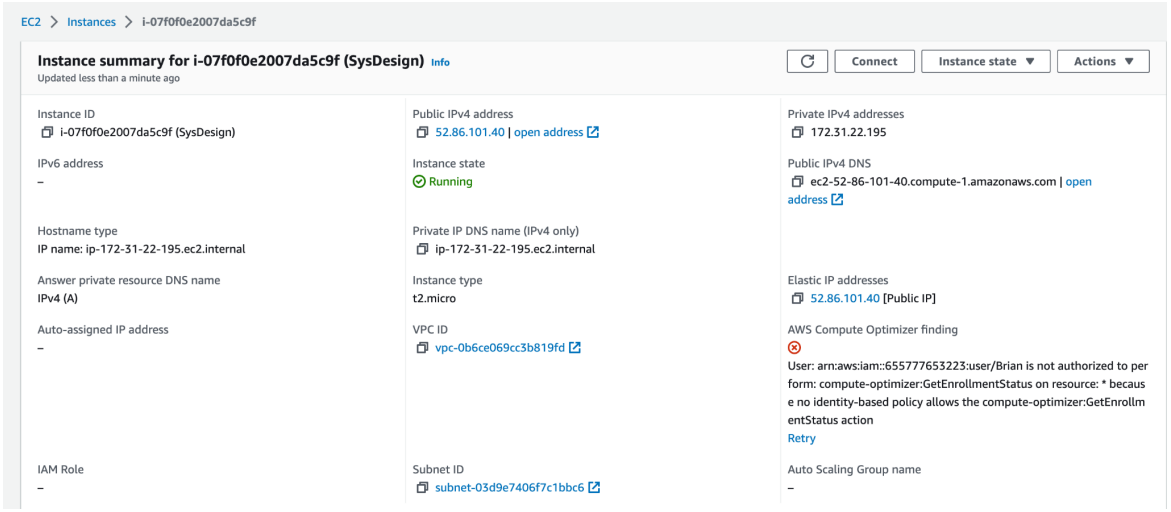


Screenshot of the GitHub page for this project

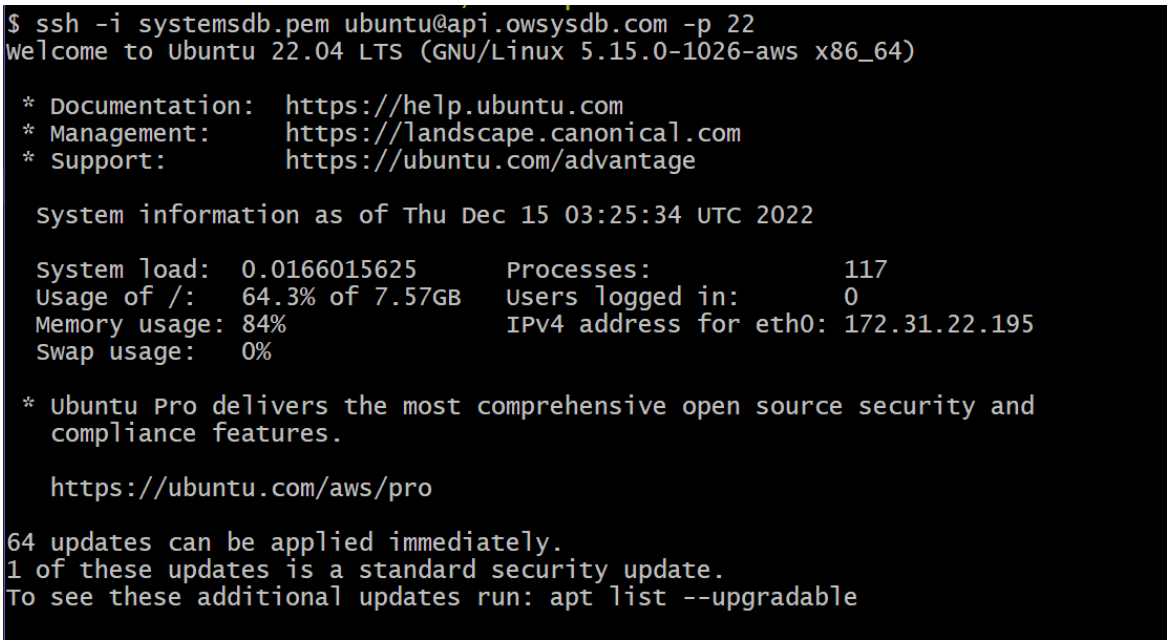
3.3 Server Hosting

The server is hosted on Amazon Web Services as an EC2 instance. The server used a Linux distribution called Ubuntu which, as stated above, runs the other components of the LAMP stack. The EC2 instance also uses Route 53, which allows for a set URL to be established for the website. Route 53 will take that URL and translate it to the IP

address given to us from Amazon. Elastic Beanstalk is also used, which assists in deploying and scaling web based applications, such as this dynamic registration system.



Instance Summary of AWS Server on AWS account



Ubuntu Instance System Information

3.3.1 AWS Simple Email Service

There exists a functionality within the system that allows a user to request a password change from an admin. This function contributes to the “Request Password Change” use case. In order for this request to

be processed, there must be an event triggering an email exchange to and from the requestor and the admin. In order to accomplish this use case, we integrated Amazon Simple Email Service (SES) in our EC2 Instance. Amazon SES is a cloud email service provider that can integrate into any application for bulk email sending.The SES account resides in the AWS Sandbox, where all features of SES can be accessed with the following restrictions: You can send a maximum of 200 email per 24 hour period, you can only send emails to verified email addresses and domains, and you can send maximum of one message per second. Due to the request for this email use case to only be executed for testing purposes, there is no need to request Amazon Support Center to move the SES account from the Sandbox instance.

UID	FirstName	LastName	BirthDate	Gender	Email	username	UserType	PriorityLevel
324706	Bunny	Mattosoff	1994-01-18	F	bmatto.aiu@gmail.com	bmatto	Administ...	1
345650	Gwen	Alessi	1997-06-04	F	galessi.aiu@gmail.com	galessi	Administ...	0
380119	Brian	Mejia	1996-05-12	M	bmejia.aiu@gmail.com	bmejia	Administ...	0
394466	Andrew	Stephens	1993-01-04	M	asteph.aiu@gmail.com	asteph	Administ...	0
302200	Dick	Rozalski	1974-12-29	M	drozal.aiu@gmail.com	drozal	Faculty	NULL
352980	Missie	Sambeck	1995-10-22	F	msambe.aiu@gmail.com	msambe	Researcher	NULL
302103	Waldemar	Matushevich	1998-05-13	M	wmatus.aiu@gmail.com	wmatus	Student	NULL

The email addresses above are verified as the Email Address Identifiers, the accounts associated with these emails will be used for testing this functionality.

SMTP credentials were created through the same AWS account, these credentials are needed to target sending and receiving. PHP Mailer library has been installed and accessed in order to send the mail. Within the file ses_mailer.php resides the necessary code and information to activate the SES Mailer function.

```
<?php
header( header: "Access-Control-Allow-Origin: *");
header( header: "Access-Control-Allow-Headers: Content-Type");

use PHPMailer\PHPMailer\PHPMailer;
use PHPMailer\PHPMailer\SMTP;
use PHPMailer\PHPMailer\Exception;
```

Within the function sendEmail, the SMTP Credentials are referenced in the .php file to access the SMTP Instance:

```php
function sendEmail($sender, $recipient, $header, $message){
//Load Composer's autoloader

    require '/home/ubuntu/vendor/autoload.php';
    //Create an instance; passing `true` enables exceptions
    $mail = new PHPMailer(true);

    try {
        //Server settings
        $mail->SMTPDebug = SMTP::DEBUG_SERVER;
        $mail->isSMTP();
        $mail->Host = 'email-smtp.us-east-1.amazonaws.com';
        $mail->SMTPAuth = true;
        $mail->Username = 'AKIAZRL3E4HTZDTE6JPY';
        $mail->Password = 'BDbjkdO2cDhuMkRYOQHXbpZbz+PZ7FTgJszlJaf1shrT';
        $mail->SMTPSecure = PHPMailer::ENCRYPTION_SMTPS;
        $mail->Port = 465;
```

3.4 Database:

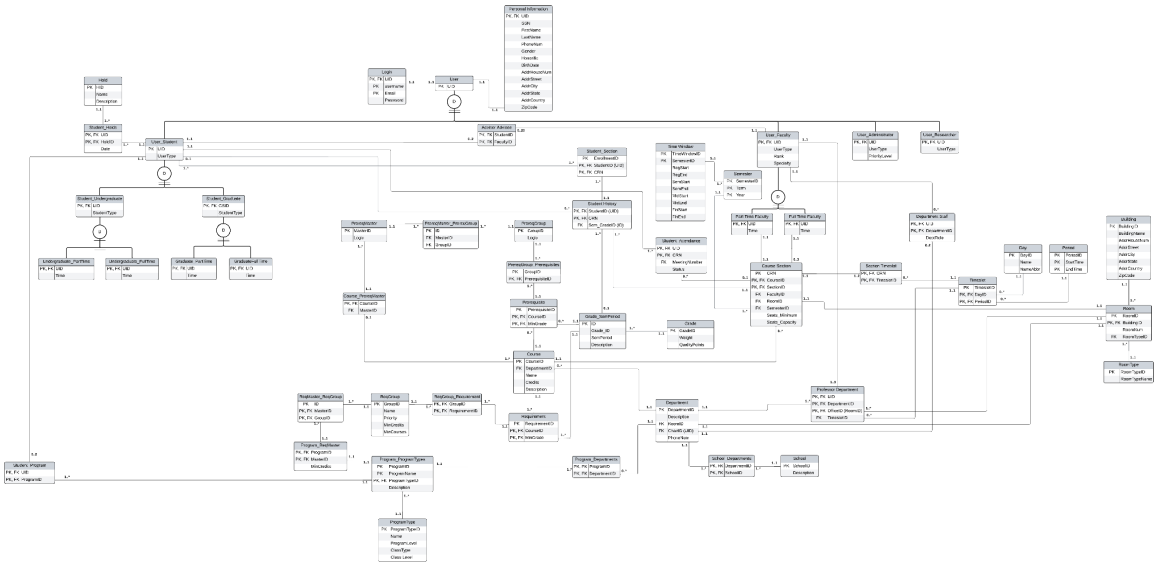
The database is managed using the RDBMS name MySQL. MySQL was required for this project because it is a Relational Database Management System. A relational database organizes data into one or more data tables in which data may be related to each other; these relations help structure the data. With a student registration system, many different tables of data must be used in unison with one another. This allows for the proper storage and handling of personal information, login credentials, academic histories and other data that a website like this entails.

3.4.1 Database Structure

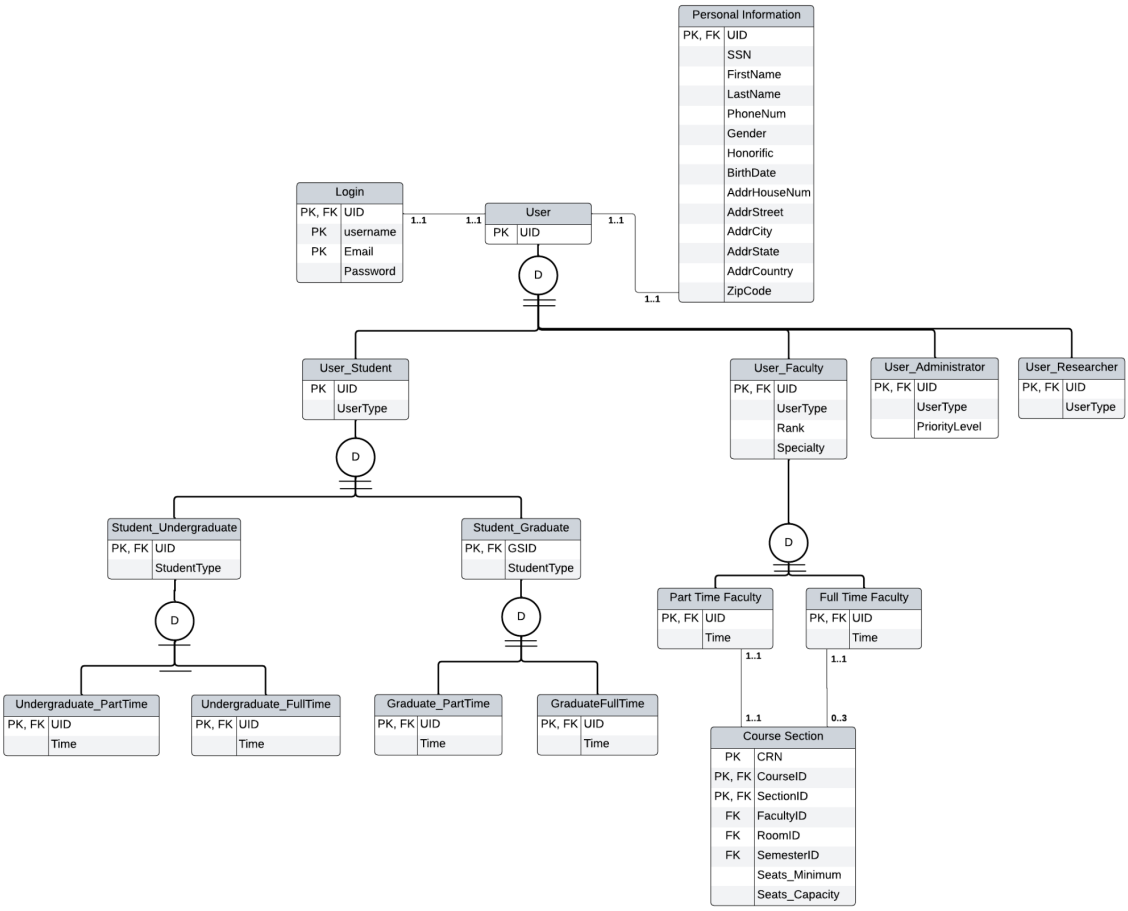
The SQL language is different from other programming languages in the sense that it does not contain a built in way to establish a clear inheritance between two tables. As stated above, a relational database is required when dealing with data that relies on other data to function correctly. With MySQL being a RDBMS, it makes use of keys to establish relationships between the different tables within the schema. With the Primary and Foreign keys, connections can be made.

The data used by the website is housed in their own individual tables, named and structured for efficiency in retrieving it when calls are made to the database. Most calls made to the database require the joining of the data within these tables to bring together one complete view. The joining of these tables is done by including a Foreign Key that references a Primary Key of another table. We further improved on this with implementing a rule of naming a Foreign Key in one table the same thing as the Primary Key that it references.

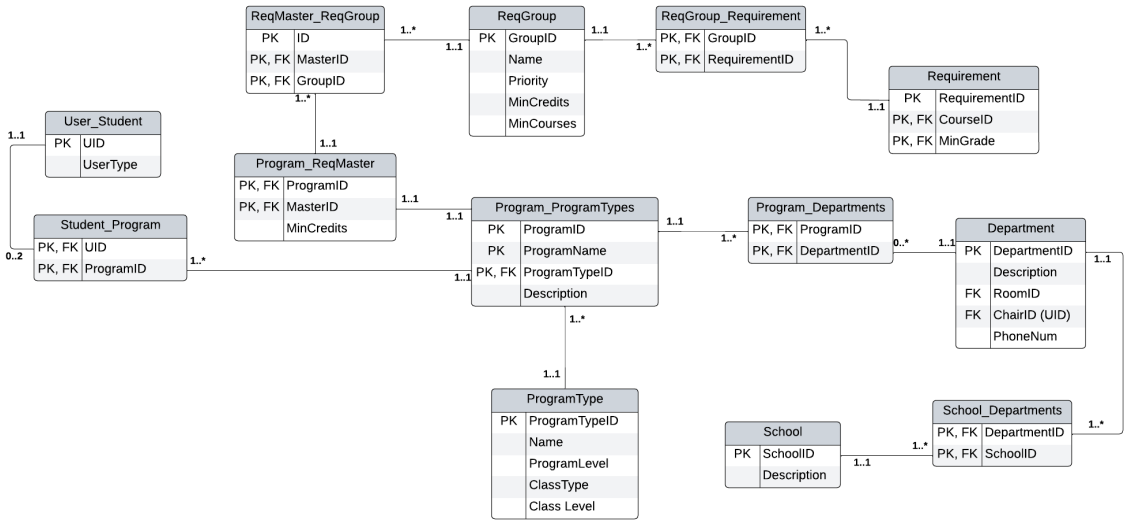
Full System EERD



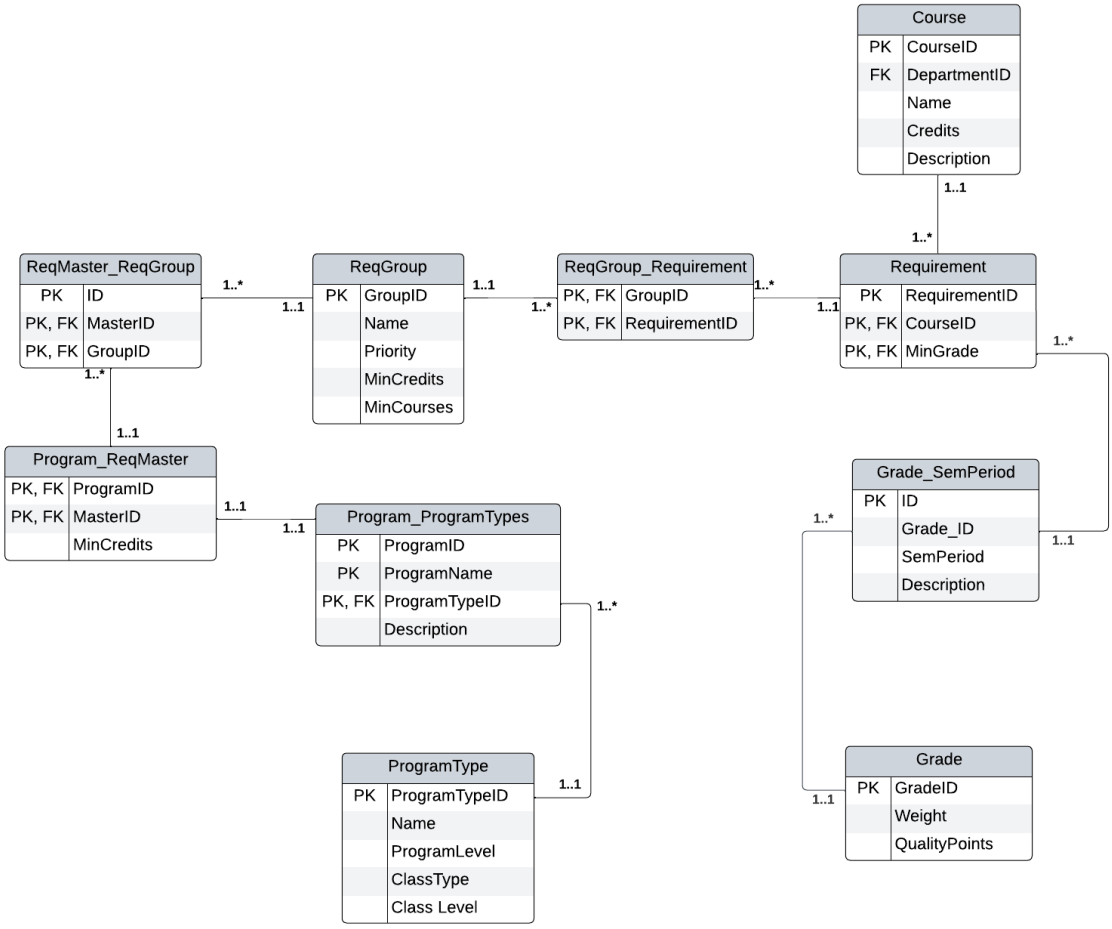
Users Subsystem



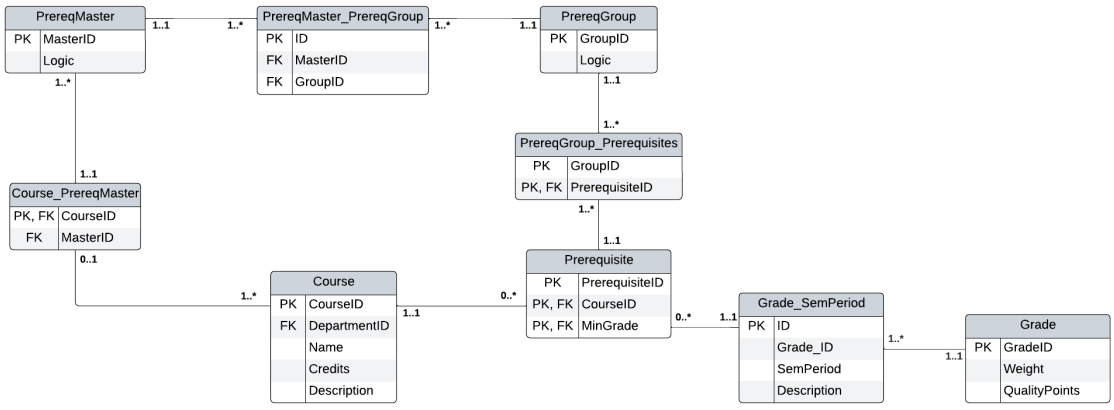
Program Requirements Subsystem



Course Section Subsystem



Course Prerequisites Subsystem



3.4.2 Stored Procedures

Stored Procedures are pre-compiled SQL statements that are stored within a database. We create these stored procedures with different statements, be it SELECT, INSERT, UPDATE or DELETE. These stored procedures can take in information as well as output it. Once they are made, stored procedures can be called in the front end to get information from the database and display it on the website or make changes to the data in the database based on user input. Examples of these in our database include calls for selecting a master schedule based on semester, inserting a class into a student's schedule, and combining data from different tables to form a complete degree audit.

Stored procedures expedite the process of making calls to the database from the front end. They make it so that each individual call does not have to be a complete SQL statement in itself, but rather just a call to that specific saved procedure.

```
function updateUserPersonalInformation($conn, $params) {

    echo "Entered function -> UID = ". ($params['uid']);

    if(!(isset($params['uid']))) {
        echo "Incomplete request";
    }

    $uid = $params['uid'];

    $fName = $params['fName']===""?$params['fName']:null;
    $lName = $params['lName']===""?$params['lName']:null;
    $phone = $params['phone']===""?$params['phone']:null;
    $gender = $params['gender']===""?$params['gender']:null;
    $honorific = $params['honorific']===""?$params['honorific']:null;
    $birthdate = $params['birthdate']===""?$params['birthdate']:null;
    $houseNum = $params['houseNum']===""?$params['houseNum']:null;
    $street = $params['street']===""?$params['street']:null;
    $city = $params['city']===""?$params['city']:null;
    $state = $params['state']===""?$params['state']:null;
    $country = $params['country']===""?$params['country']:null;
    $zip = $params['zip']===""?$params['zip']:null;

    $stmt = $conn->prepare("CALL updatePersonalInformation(?,?,?,?,?,?,?,?,?,?,?,?,?)");
    $stmt->bind_param("sssssssssssss", $uid, $fName, $lName, $phone, $gender, $honorific, $birthdate, $houseNum, $street, $city, $state, $country, $zip);
    $status = $stmt->execute();
    if($status === false)
        trigger_error($stmt->error, E_USER_ERROR);

    $arr ['status'] = "Success!";

    echo(json_encode($arr));

    mysqli_close($conn);

}
```

A call made in the front end to a stored procedure that exists within the database to update a user's personal information

```
CREATE DEFINER=`admin`@`localhost` PROCEDURE `getUserPersonalInformation`(IN UID VARCHAR(11))
BEGIN
    SELECT
        UID, SSN, FirstName, LastName, PhoneNum, Gender, Honorific, BirthDate,
        AddrHouseNum, AddrStreet, AddrCity, AddrState, AddrCountry, ZipCode, Email,
        CONCAT(IFNULL(s.UserType,""), IFNULL(f.UserType,""), IFNULL(a.UserType,""), IFNULL(r.UserType,"")) AS UserType,
        CONCAT(IFNULL(su.StudentType,""), IFNULL(sg.StudentType,""), IFNULL(f.Rank, "")) AS "Rank",
        CONCAT(IFNULL(upt.Time,""), IFNULL(ugft.Time,""), IFNULL(ppt.Time,""), IFNULL(pft.Time,""), IFNULL(mpt.Time,""), IFNULL(mft.Time,""), IFNULL(fpt.Time,""), IFNULL(fft.Time,"")) AS Time
    FROM User u
        JOIN PersonalInformation USING(UID)
        JOIN Login USING(UID)
        LEFT JOIN User_Student s USING(UID) LEFT JOIN User_Faculty f USING(UID) LEFT JOIN User_Administrator a USING(UID) LEFT JOIN User_Researcher r USING(UID)
        LEFT JOIN Faculty_FullTime fft USING(UID) LEFT JOIN Faculty_PartTime fpt USING(UID)
        LEFT JOIN Student_Undergraduate su USING(UID) LEFT JOIN Student_Graduate sg USING(UID)
        LEFT JOIN Undergraduate_PartTime upt USING(UID) LEFT JOIN Undergraduate_FullTime ugft USING(UID)
        LEFT JOIN PhD_PartTime ppt USING(UID) LEFT JOIN PhD_FullTime pft USING(UID)
        LEFT JOIN Masters_PartTime mpt USING(UID) LEFT JOIN Masters_FullTime mft USING(UID)
    WHERE(
        UID = u.UID
    );
END
```

The Stored Procedure being called

3.4.3 Data Testing

The consistency of data was also included as a top priority when developing the database aspect of the project. The overall structure of the database depends heavily on the data that populates it being accurate. In the beginning, the data we gathered from the SUNY Old Westbury Course Catalog had errors in it that could potentially lead to a much larger issue if left unchecked, which led to us implementing data checks before importing anything into the main database. This included testing for duplicates, maintaining the integrity of Primary Keys and Foreign Keys when not working in MySQL itself, and generating complicated data combinations by hand when generating it automatically was not an option.

A1:A2 | fx | Courses

	A	B	C	D	E	F	G	H	I	J
1	Courses	Master ID	Master Logic	Master Groups			Group Logic	Group Prereq		Prereq
2				ID	Master ID	Group ID		Group ID	Prereq ID	
462	PY4760	20	AND	41	20	119	AND	119	264	PY2010
463								119	269	PY3010
464				42	20	121	OR	121	266	PY2530
465								121	267	PY2720
466								121	282	PY3710
467	PY4240	21	AND	43	21	119	AND	119	264	PY2010
468								119	269	PY3010
469				44	21	123	OR	123	265	PY2400
470								123	270	PY3215
471								123	271	PY3230
472								123	272	PY3240
473	PY5210	22	AND	45	22	119	AND	119	264	PY2010
474								119	269	PY3010
475				46	22	124	OR	124	270	PY3215
476								124	271	PY3230
477								124	272	PY3240
478	PY4420	23	AND	47	23	119	AND	119	264	PY2010
479								119	269	PY3010
480				48	23	125	OR	125	270	PY3215
481								125	272	PY3240
482	PY4520	24	AND	49	24	119	AND	119	264	PY2010
483								119	269	PY3010
484				50	24	126	OR	126	266	PY2530
485								126	273	PY3310
486								126	282	PY3710
487	PY4340	25	AND	51	25	119	AND	119	264	PY2010
488								119	269	PY3010
489				52	25	127	OR	127	273	PY3310
490								127	274	PY3311
491	PY4550	25	AND	51	25	119	AND	119	264	PY2010
492								119	269	PY3010

```
1  call systemsdb.displayAllPrerequisiteData('py4760');
2
```

100% 1:1

Result Grid Filter Rows: Search Export:

Course	MasterID	MasterLogic	Master/GroupID	GroupID	GroupLogic	PrereqCourse	MinGrade
PY4760	20	AND	41	119	AND	PY2010	7
PY4760	20	AND	41	119	AND	PY3010	7
PY4760	20	AND	42	121	OR	PY2530	11
PY4760	20	AND	42	121	OR	PY2720	11
PY4760	20	AND	42	121	OR	PY3710	11

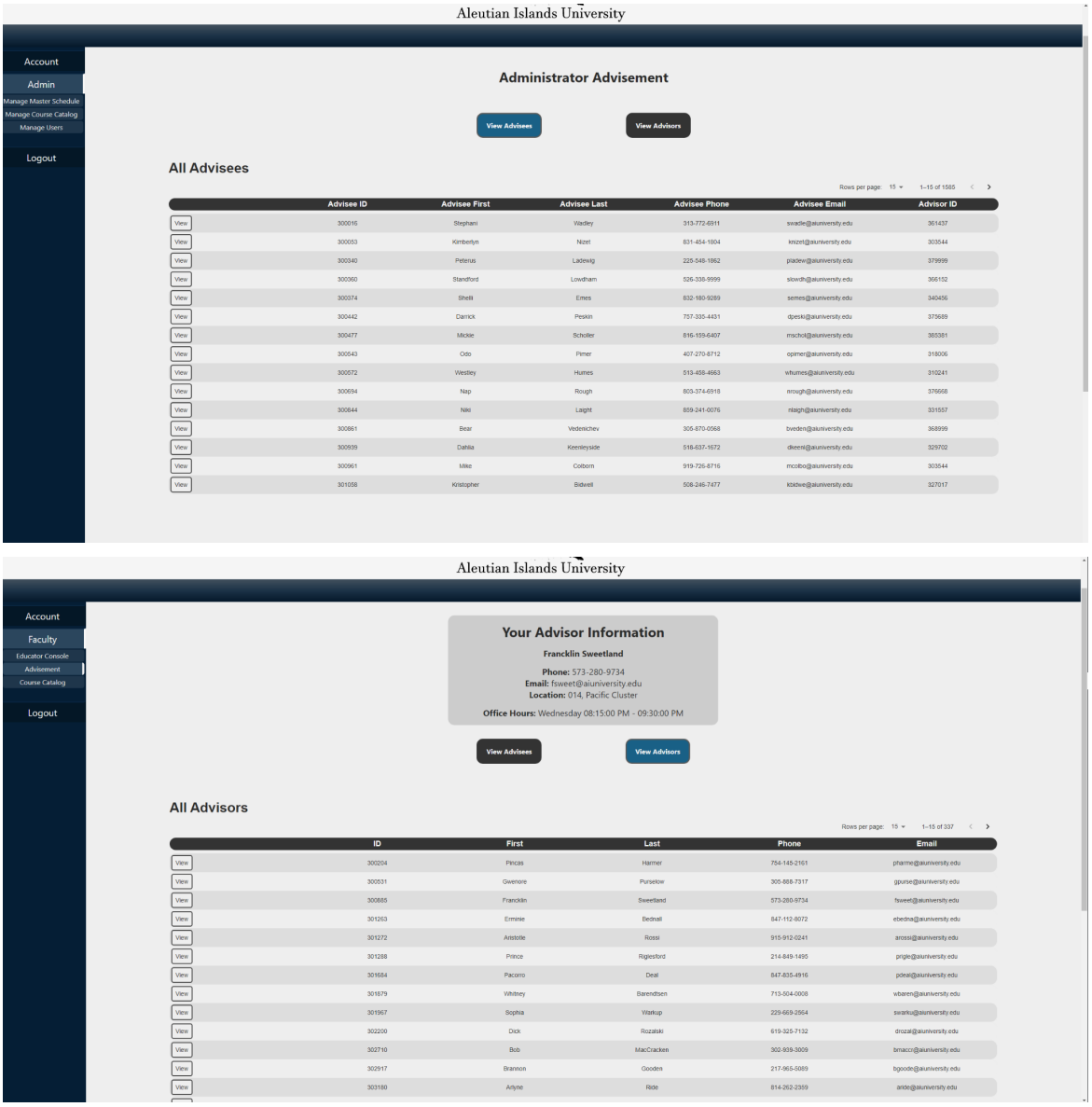
A testing sheet (top) made to test the integrity of the prerequisite tables (bottom)

4. Low-Level Design

4.1 React Framework

React is a free and open-source front-end JavaScript library for building user interfaces based on UI components. The version of React we are using is v18.1.0. React cuts down on the amount of code needed when developing a user interface in vanilla JavaScript. React also differs from the back and forth aspect of generating web pages using traditional JavaScript, React takes a different approach by letting you build a single-page application (SPA). A single-page application loads only a single HTML document on the first request. Then, it updates the specific portion, content, or body of the webpage that needs updating using JavaScript. This has proven beneficial when developing a dynamic student registration system as most pages will remain the same for displaying information to a wide array of users. The data changes, however the structure of that data is maintained. This is the same system that Facebook uses for its website

The reuse of these components can be seen throughout our website, which aided in the criteria we set for usability. The data displayed for a Faculty member viewing the master schedule is the same as an Administrator or Student. While manipulation of that data is limited to only the Administrator, the structure of the view is consistent thanks to React. Other examples on the website are advisement contacts, course section rosters and course catalog information.

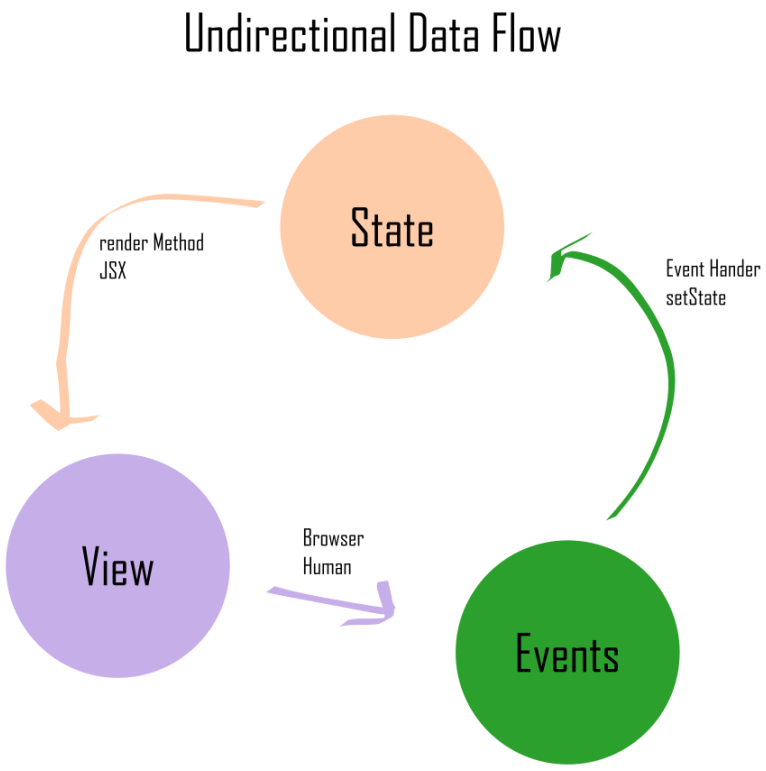


The advisement information for an admin (above) and a faculty member (below)

As seen above, the data being presented changes from user to user. However, how it is displayed remains the same. Not only does this apply to the presentation of data in our website, but the overall structure of the site. The left tabs and menu are changed from user to user, displaying their role and the actions that go along with it. What does not change though is the Account tab, which remains the same. The account tab displays personal information for any type of user, so the same structure is maintained.

4.2 Data flow

React uses a unidirectional flow of data, meaning that the data can move in only one pathway when being transferred between different parts of the program. The parent data, known as a prop, can only be transferred down to a child, not vice versa. The data that is being passed in from the database, or state, is stored in preparation for a view. The view is where a state is converted into the user interface. When a user wants to interact with the data, they trigger an event. A button clicked, a tab selected, etc. These actions can call a code, a stored procedure for example, and execute whatever needs to be done.



A diagram of a unidirectional dataflow

Unidirectional data flow makes debugging much easier. Since the origin and destination of the data is known, we can find the problems more efficiently. It also stops extra resources from being wasted, leading to greater efficiency, and allows for better control.

4.3 Object Oriented and Function Oriented Design

Function-oriented design is useful for a system such as this because of the importance of real-world users. We start with a high level description of what the program does. Then, in each step, we take one part of our high level description and refine it. Each component of the website is designed with the higher level description in mind. However, the importance of data and the manipulation of that data forces some aspects of object-oriented to be implemented as well. Object-oriented design focuses on the data that is to be manipulated by the program. It takes the real-world users and converts them into objects, software packages that are designed and developed to correspond with real world entities that contain all the data and services to function as their associated entities messages. The combination of these two design techniques equates to a website that takes into consideration the real world applications of it, but also treats the data with the level of importance it requires.

Our website makes a request on the front end that is sent to the database, for example a need for the first name and last name of a student. The database sends back a response, the student's names, as a JSON. The JSON is then destructured.

COMPARISON FACTORS	FUNCTION ORIENTED DESIGN	OBJECT ORIENTED DESIGN
Abstraction	The basic abstractions, which are given to the user, are real world functions.	The basic abstractions are not the real world functions but are the data abstraction where the real world entities are represented.
Function	Functions are grouped together by which a higher level function is obtained.	Function are grouped together on the basis of the data they operate since the classes are associated with their methods.
execute	carried out using structured analysis and structured design i.e, data flow diagram	Carried out using UML
State information	In this approach the state information is often represented in a centralized shared memory.	In this approach the state information is not represented is not represented in a centralized memory but is implemented or distributed among the objects of the system.
Approach	It is a top down approach.	It is a bottom up approach.
Begins basis	Begins by considering the use case diagrams and the scenarios.	Begins by identifying objects and classes.
Decompose	In function oriented design we decompose in function/procedure level.	We decompose in class level.
Use	This approach is mainly used for computation sensitive application.	This approach is mainly used for evolving system which mimics a business or business case.

Comparisons made between Function Oriented Design and Object Oriented Design

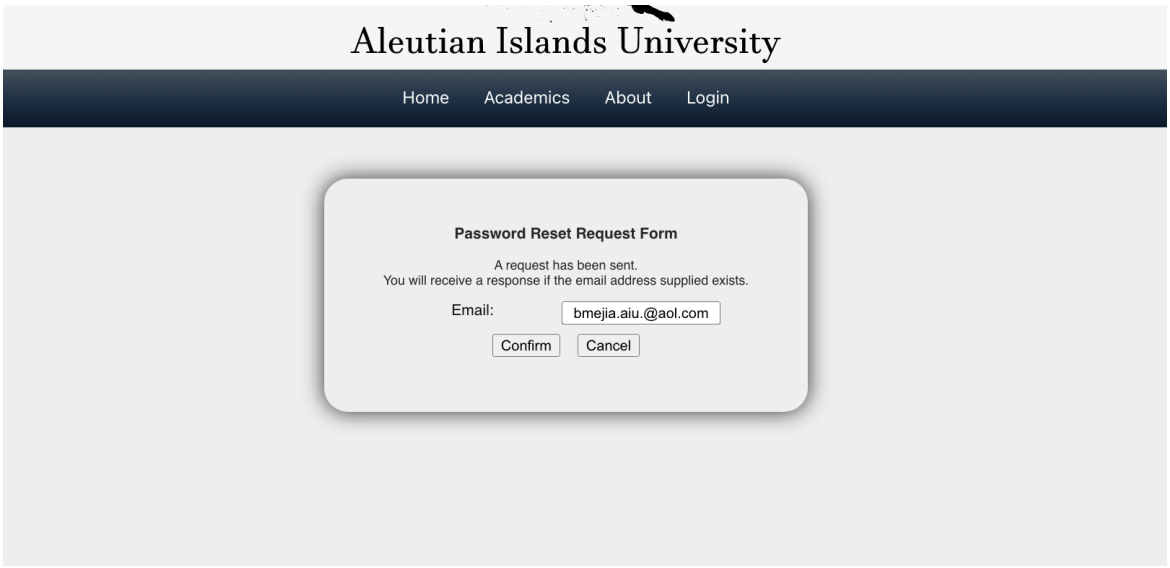
5. Extras

In here you will find areas of interest in our project that do not meet the criteria for being mentioned in the main document. Extra information about how a certain aspect of the web application works and supplemental tools used in developing code and data for the AIU Dynamic Registration System

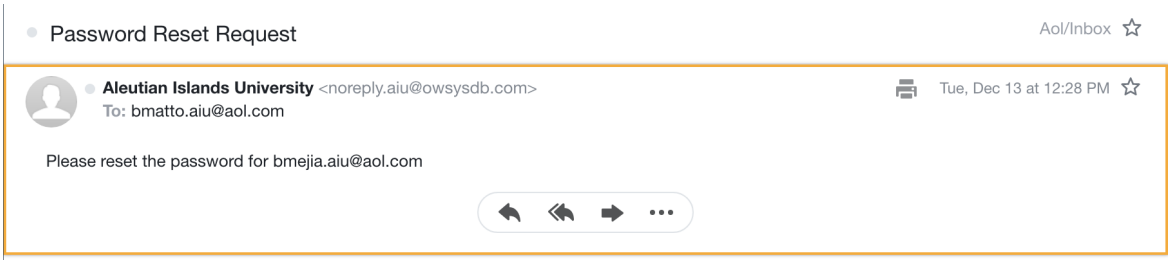
5.1 Requesting to Update Password Through Email

Since the primary admin has the ability to update a different user's password when requested, there needs to be a way to process those requests and give them to the primary admin to be processed. The solution we decided on was establishing an email address for the primary admin in order to receive requests in an automatically filed and organized manner.

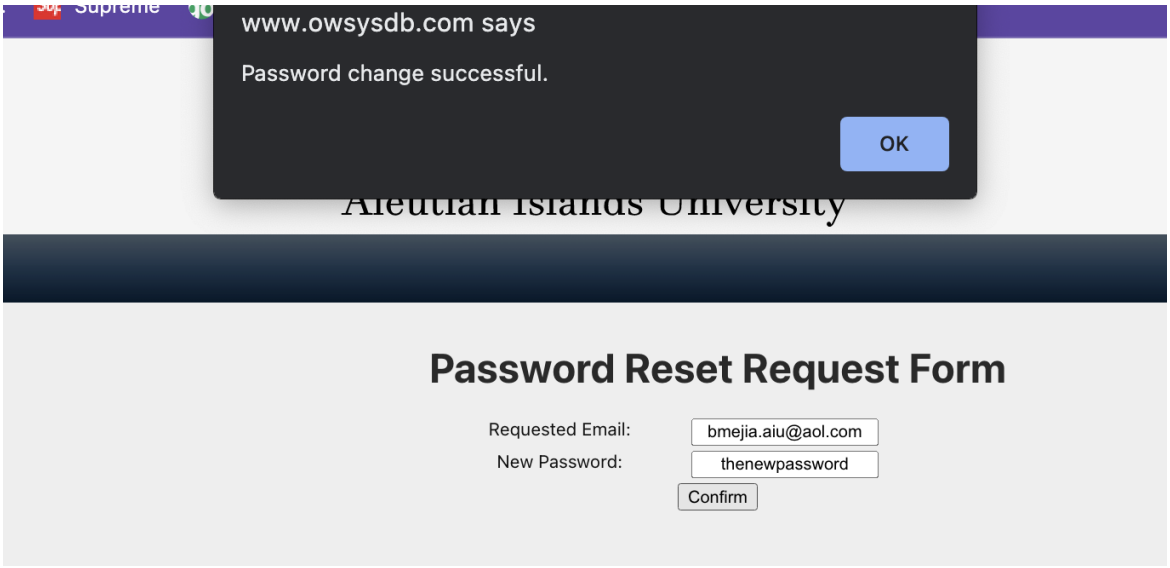
A no-reply email was established using the website that would send out emails when a user selected the "Can't Sign In?" button at sign in

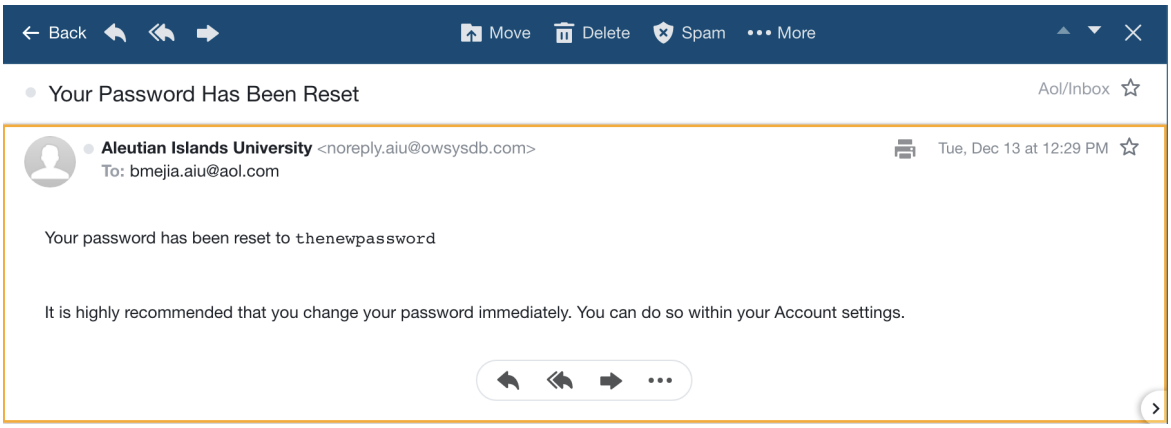


With this, the user would input their email and submit the request. An email is then generated and sent to the Primary Admin for processing.



The Primary Admin will be able to process this request when signed in to the website.





Once the reset is completed, an email will be sent back to the requestor to let them know that their password has been reset and they can now sign in with their new password.

5.2 GitHub

GitHub is an Internet hosting service used primarily for software development. As stated above, GitHub works in tangent with Git Bash, an application for Microsoft Windows environments which provides an emulation layer for a Git command line experience. We host our code in GitHub and then Git Bash is used to push or pull revisions made to the code.

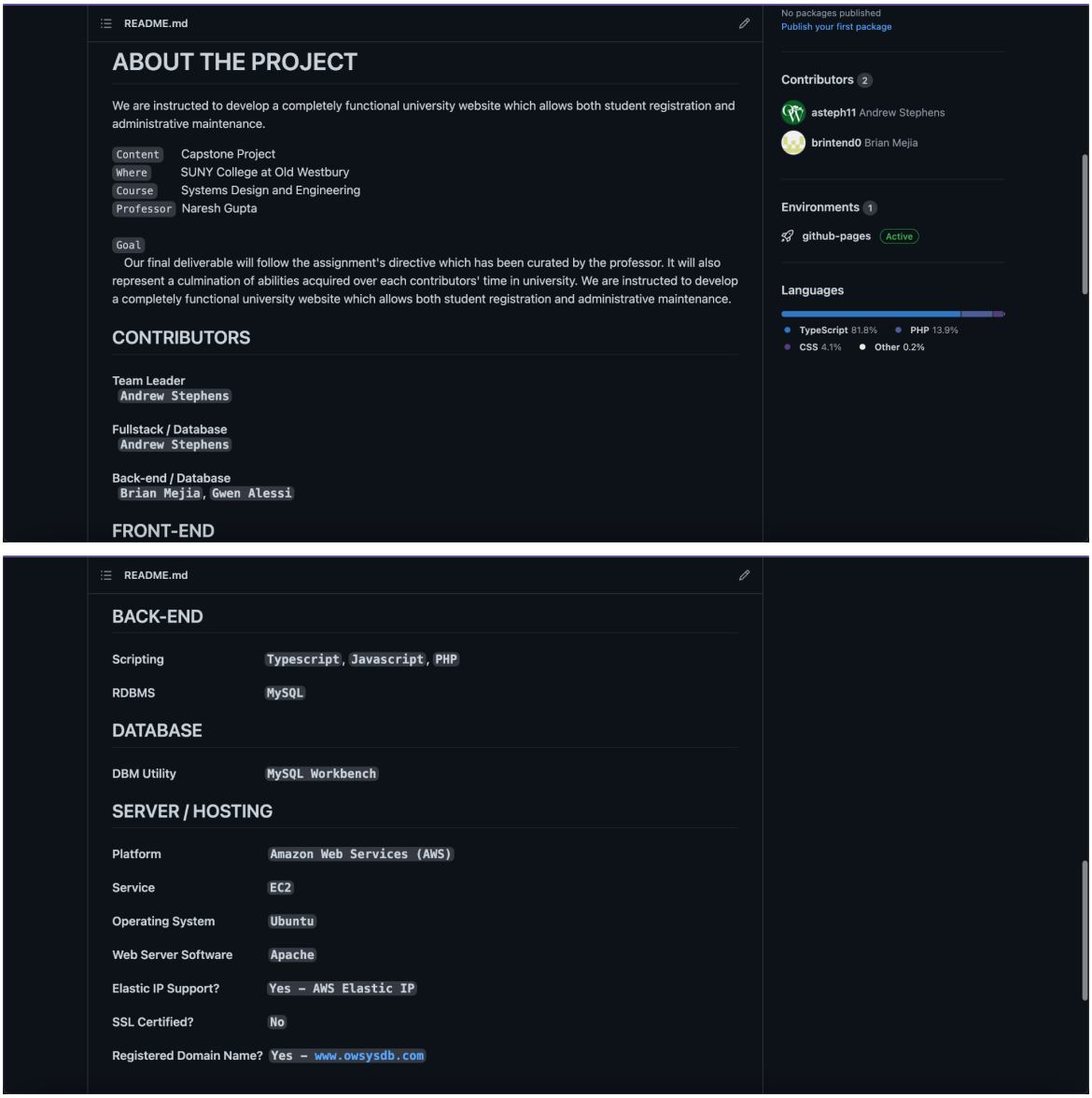


Fig 5.2a & b An overview of the GitHub page where the code for the website is hosted

GitHub provided an excellent environment for the code heavy aspects of the project, particularly the front end, to be well organized and accessible to all members of the team. The following is a direct link to this project's GitHub page for the Professor's use. Here is a URL if viewing this document physically: github.com/asteph11/SystemDesignProject

5.3 Lucidchart

Charts and figures proved very helpful to our project, especially when dealing with the intricacies of data and ensuring it was all proper and as accurate as possible. Lucidchart proved very helpful to the team, keeping a concise and shareable space where any one of us could develop a visual that would help in the development of the web application. Lucidchart was used primarily in developing the Enhanced Entity-Relationship Diagram, or EERD, of our database and developing the student histories for our users.

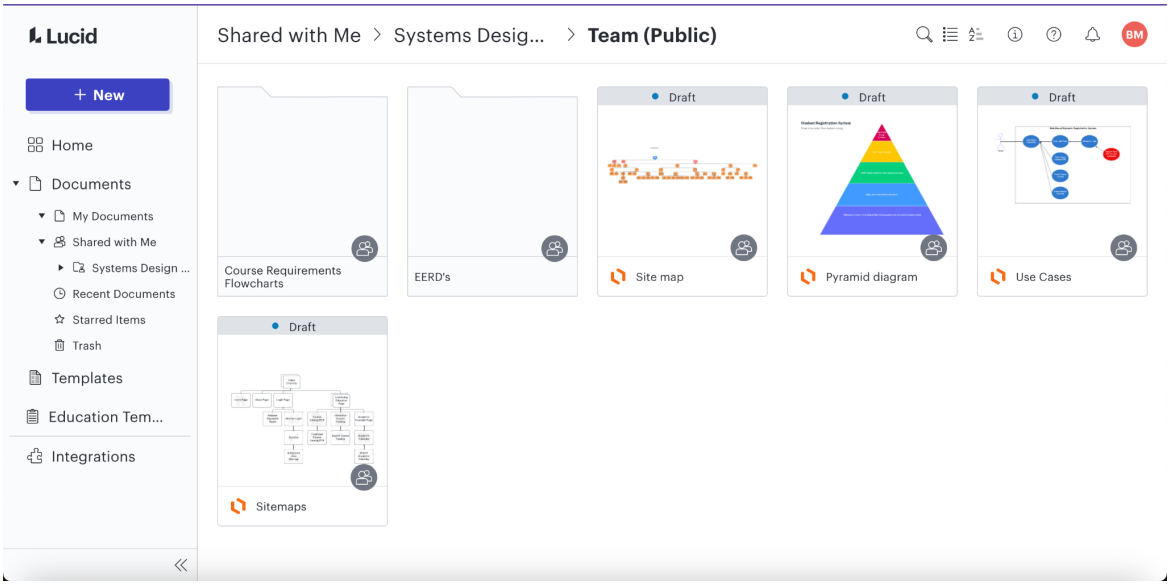


Fig 5.3a The homepage of our shared Lucidchart account

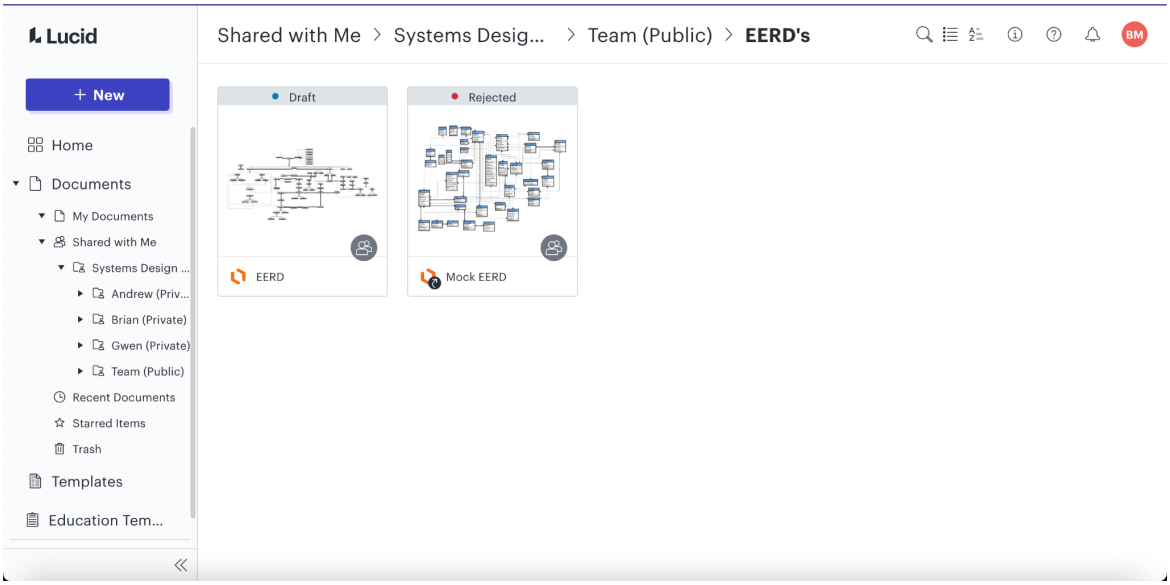


Fig 5.3b The EERD folder of our Lucidchart, along with the rejected and draft of our final EERD

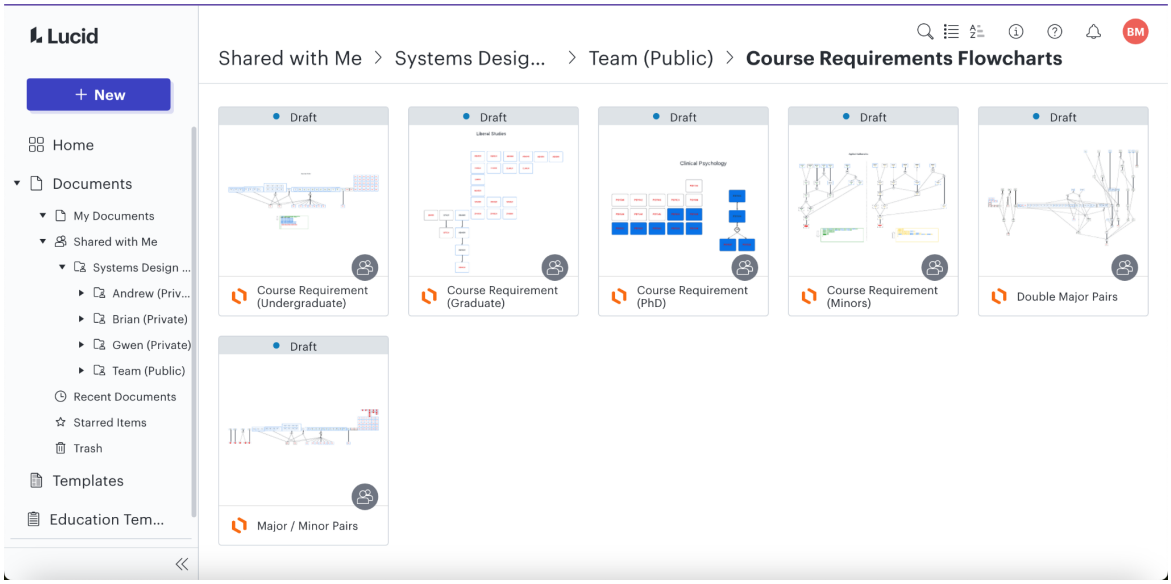


Fig 5.3c Preliminary visuals of how student histories would be structured for different types of students

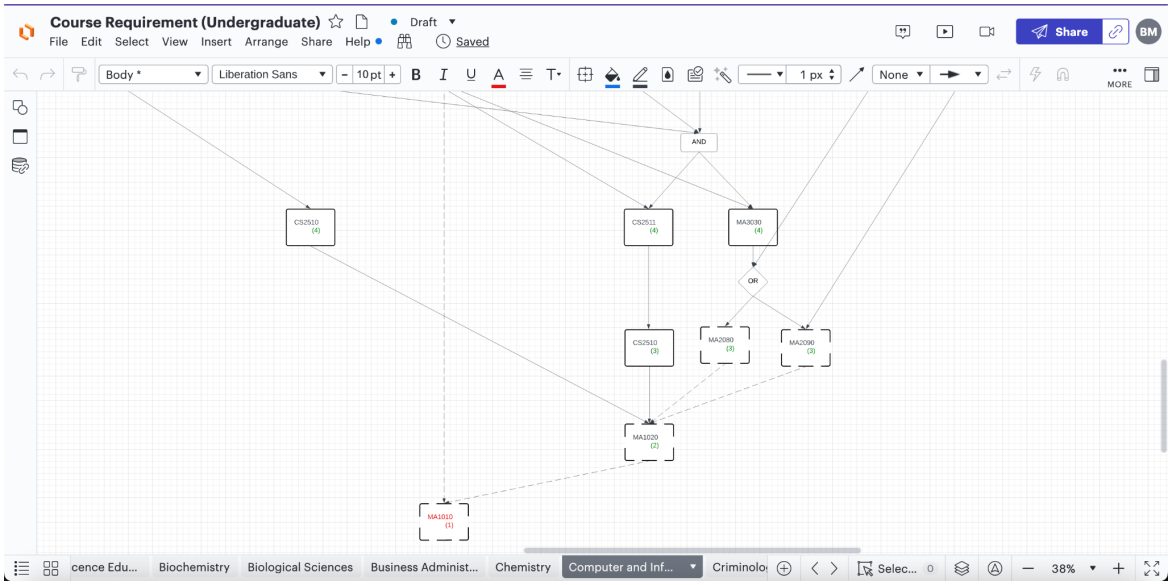


Fig 5.3d Section of the Computer and Information Science program requirements.

Appendix

1. Information Acquisition

1.1 Copied Old Westbury Catalog Information:
https://docs.google.com/spreadsheets/d/10Z9UGwUCkBEvw2harC1WxvzFL3dYVXgsGRbuRFvJk5A/edit#gid=1643004219

2. Data Preparation

2.1 Program Requirements - Preliminary:
https://docs.google.com/spreadsheets/d/1WG0XvKxpN27j9mFkiWNka2xr_0t4o-PbptMcXI_xsXY/edit#gid=0

2.2 Program Requirements - Refined:
https://docs.google.com/spreadsheets/d/1jqStkW_gKdt382gWan3dX2tLm6QVsbe3pmRRPqbG2v0/edit#gid=0

2.3 Stored Procedure - Brainstorm:
https://docs.google.com/spreadsheets/d/1P06kf0f9lF3T-mUq-TbYFw3RYwks4mJly0fMtkaWtDk/edit#gid=548051931

2.4 Prerequisites - Old Westbury Logic Conversion:
https://docs.google.com/spreadsheets/d/1e_jx1k4oeCR1hBB9QNGHArHeazDTt8UEp5tojAQiCQM/edit#gid=0

2.5 EERD:
https://lucid.app/lucidchart/b681d5e0-1337-4081-8337-3c4867abaee0/edit?viewport_loc=-777%2C638%2C2739%2C3133%2CYBq5jwsBG0cR&invitationId=inv_41d3e60b-2981-4d0b-b769-859ffa55506e

2.6 Program Requirements Trees:
https://lucid.app/folder/invitations/accept/inv_34b503b1-8dbd-4bd1-a6f8-85548c58e163

3. Data Creation

3.1 Preliminary Database Data:
https://docs.google.com/spreadsheets/d/1ZI0laUdkyIesBPi6dCq98HNx1jTLpUs4ypWlRerMZDw/edit?usp=sharing

3.2 Original Student Histories:
https://docs.google.com/spreadsheets/d/13n9iNzy3UACwhFE0Gkcv1HjNcQQ1DNZbWc3JJwqzJCg/edit#gid=468704262

4. Project Files

4.1 Github Repository:
https://github.com/asteph11/SystemDesignProject