System Manual
For
Maple Leaf University
System Design and Implementation CS 5910-HY2
SUNY Old Westbury
Dr. Naresh Gupta

Team Leader: Emiz Intriago: eintriag@oldwestbury.edu (631) 524-6448
Anibal Ruiz: aruiz23@oldwestbury.edu (516) 725-6020
Yesenia Fernandez: yferna16@oldwestbury.edu (516) 713-2249

Table of contents

Introduction

Hosting

The Website is hosted by hostsinger for both backend and front endpoint. The website is not able to hos

Visitor Users

The visitors of this system are restricted from using the registration system to a certain extent. These restrictions include trying to login to the server, trying to register for courses which only the students are allowed to do. The visitor will only be allowed to view courses being offered, the master schedule, majors and minors being offered, etc.

Student Users

The student will have complete access to the registration system, but will not be allowed to do faculty or admin things. In Maple Leaf University, students can access their account with the right credentials and register for courses they need in order to complete their major. Students can also be able to view all the details of the course that was

selected. To keep track of progress, students can view their degree audit. The students can also change their major or minor.The registration system takes into account being able to withdraw or drop a course during a given time window.

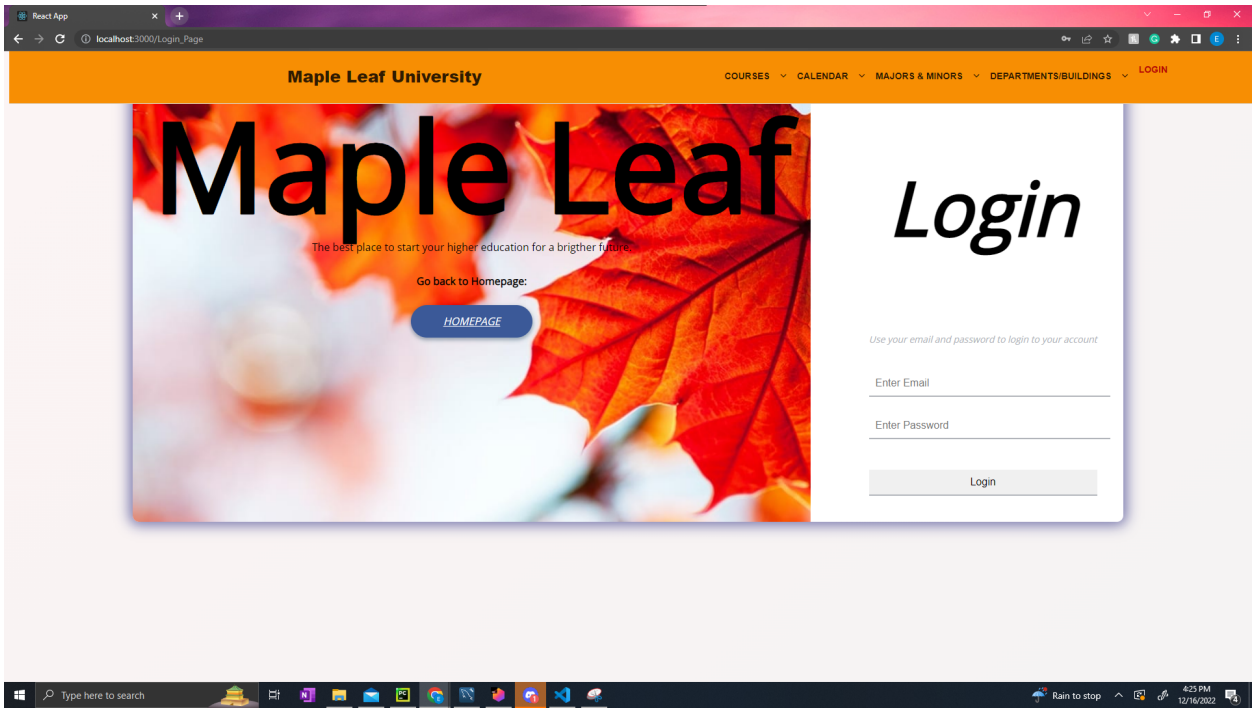
Faculty users

The faculty is the professor who is teaching the course. The faculty members can have control of their own course schedules and the student's course schedule that they advise. They can assign grades and keep attendance between the time window. Faculty can see everything students can see.

Admin

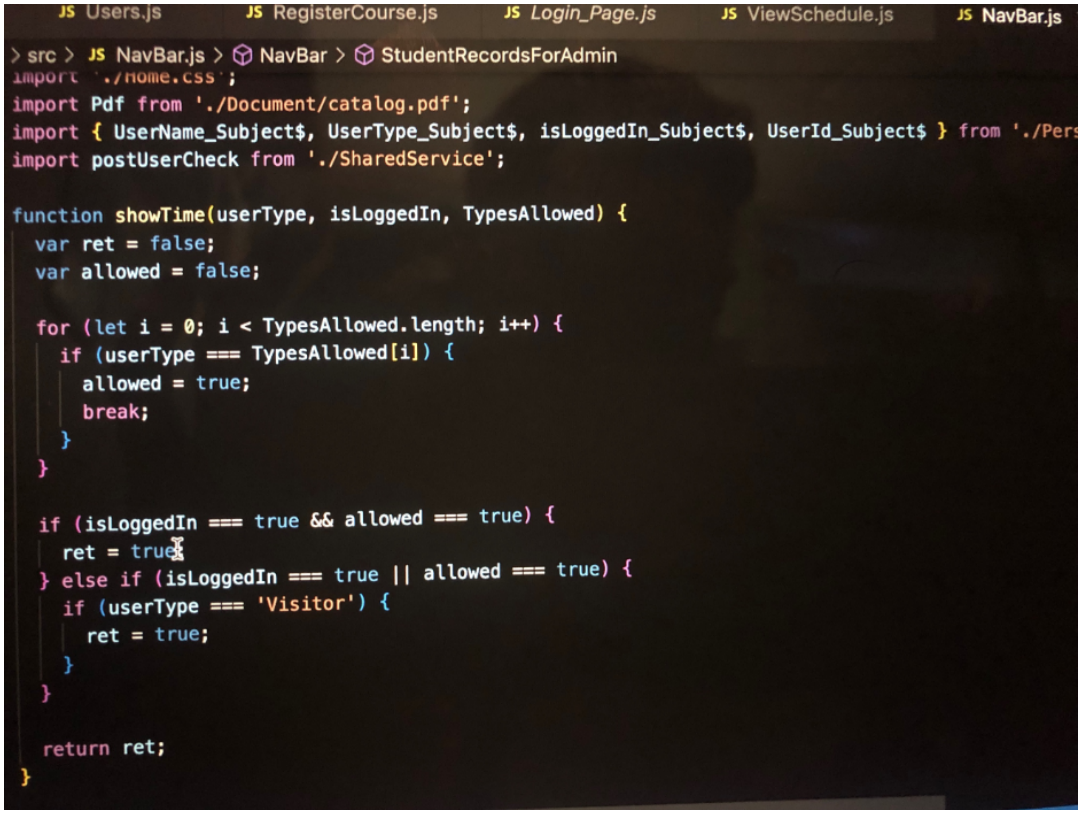
The administrator is the main person in charge of the Maple Leaf University system. The admin is responsible for creating new courses, adding a class to master schedule, creating new users, adding and removing holds for students, etc. the administrator can do everything that faculty and students can do.

Login

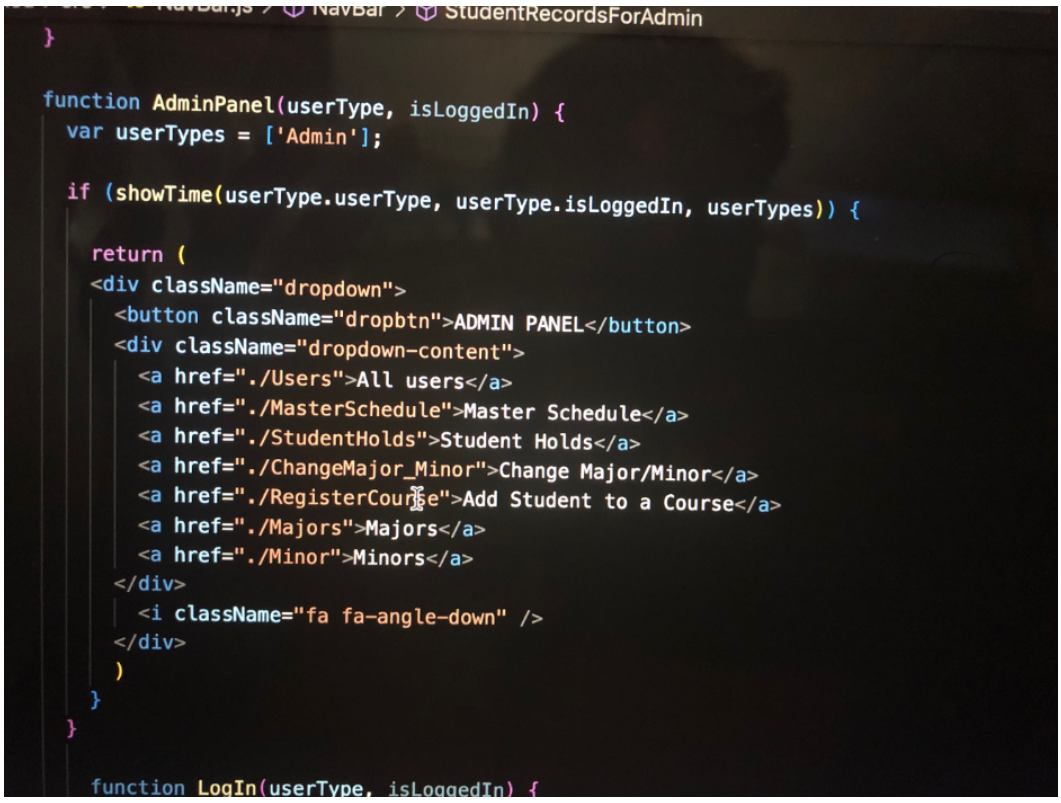


Users that have credentials to login can access their accounts using email and password. Each user in the backend has a UserType value which is essential for accessing the student, faculty, or admin page. When user logins, we send the info for email and password to the backend where the values are matched to the right credentials, then the usertype us checked to see what type of user is trying to access the system. A token is created at that moment with an ID which will keep the user logged in all the time the user requires. Also this token prevents the user from accessing pages they should not not be allowed to access. When logout is clicked, then this token is removed and the user is back to the homepage.
If the user exceeds the 3 attempts for login, the system will kick out the user from the login page.

Navigation Bar



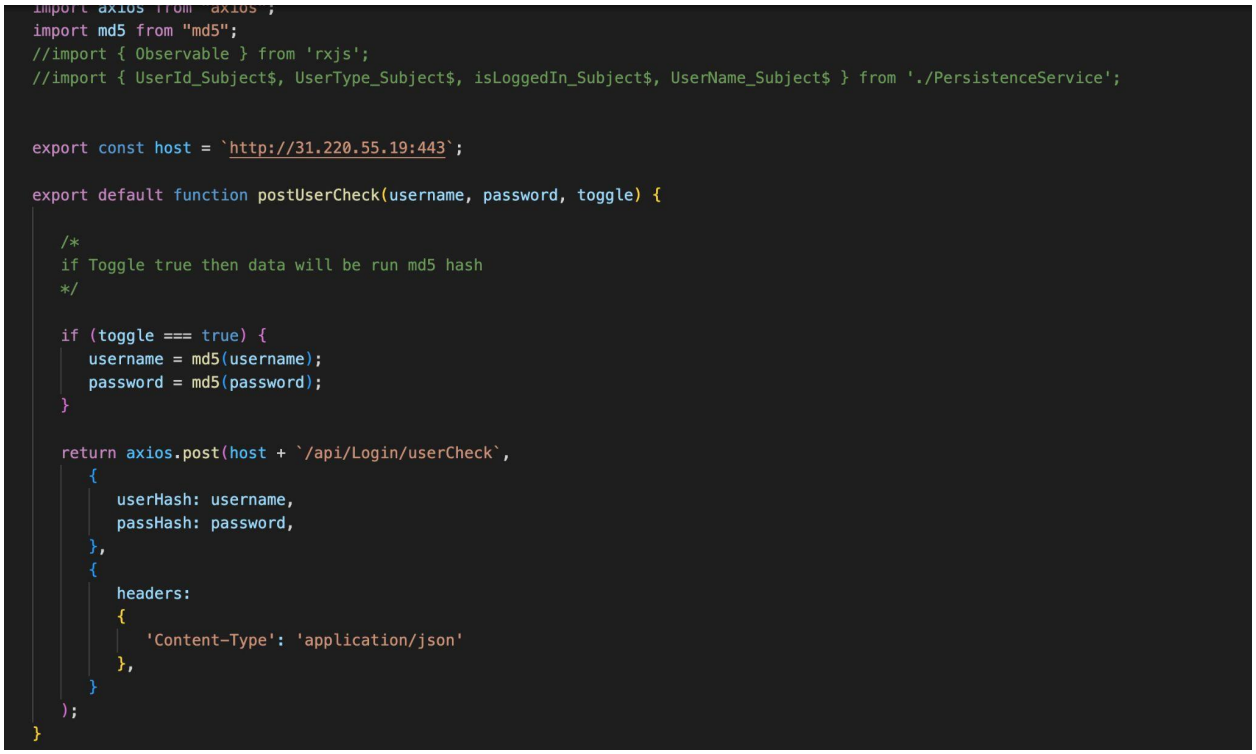
```
Users.js  RegisterCourse.js  Login_Page.js  ViewSchedule.js  NavBar.js
src > NavBar.js > NavBar > StudentRecordsForAdmin
import './home.css';
import Pdf from './Document/catalog.pdf';
import { UserName_Subject$, UserType_Subject$, isLoggedIn_Subject$, UserId_Subject$ } from './Pers
import postUserCheck from './SharedService';

function showTime(userType, isLoggedIn, TypesAllowed) {
  var ret = false;
  var allowed = false;

  for (let i = 0; i < TypesAllowed.length; i++) {
    if (userType === TypesAllowed[i]) {
      allowed = true;
      break;
    }
  }

  if (isLoggedIn === true && allowed === true) {
    ret = true
  } else if (isLoggedIn === true || allowed === true) {
    if (userType === 'Visitor') {
      ret = true;
    }
  }

  return ret;
}
```



```
NavBar.js > NavBar > StudentRecordsForAdmin
  }

  function AdminPanel(userType, isLoggedIn) {
    var userTypes = ['Admin'];

    if (showTime(userType.userType, userType.isLoggedIn, userTypes)) {

      return (
      <div className="dropdown">
        <button className="dropbtn">ADMIN PANEL</button>
        <div className="dropdown-content">
          <a href="./Users">All users</a>
          <a href="./MasterSchedule">Master Schedule</a>
          <a href="./StudentHolds">Student Holds</a>
          <a href="./ChangeMajor_Minor">Change Major/Minor</a>
          <a href="./RegisterCourse">Add Student to a Course</a>
          <a href="./Majors">Majors</a>
          <a href="./Minor">Minors</a>
        </div>
          <i className="fa fa-angle-down" />
        </div>
        )
      }
    }

    function LogIn(userType, isLoggedIn) {
```

Navigation bar was done similar to login. For each user panel (student, faculty, admin). We create a token using the Usertype and pass the value to the NavBar page to check what to show depending on the user that is logged in. For example if the user is an admin, the user can see all users, master schedule, student holds, change major/minor or add a student to a course, and others. In this way we prevent other users as students or faculty from accessing any admin page.

POST AND GET REQUESTS



```
import axios from "axios";
import md5 from "md5";
//import { Observable } from 'rxjs';
//import { UserId_Subject$, UserType_Subject$, isLoggedIn_Subject$, UserName_Subject$ } from './PersistenceService';


export const host = `http://31.220.55.19:443`;

export default function postUserCheck(username, password, toggle) {

   /*
   if Toggle true then data will be run md5 hash
   */

   if (toggle === true) {
      username = md5(username);
      password = md5(password);
   }

   return axios.post(host + `/api/Login/userCheck`,
      {
         userHash: username,
         passHash: password,
      },
      {
         headers:
         {
            'Content-Type': 'application/json'
         },
      }
   );
}
```

```
}

export function createUser(firstName, lastName, dob, street_address, state, city, zipCode, country, userType) {
   //call api to create user
   //console.log("HELLLOOOOOOOO", firstName);
   return axios.post(host + `/api/createUser`,
      {
         //UserId: props.UserId,
         userFname: firstName,
         userLname: lastName,
         DOB: dob,
         Address: street_address,
         State: state,
         City: city,
         ZipCode: zipCode,
         Country: country,
         UserType: userType,
         //Email: props.Email,
         //Password: props.Password,
      },
      {
         headers:
         {
            'Content-Type': 'application/json'
         },
      }
   );
}

export function editUser(userId, firstName, lastName, dob, street_address, state, city, zipCode, country, userType) {
   //call api to create user
```

```
export function getBuildings() {

   return axios.get(host + `/api/getBuilding`)
      .then(x => { return x; });

}

export function getTimeWindow() {

   return axios.get(host + `/api/getTimeWindow`)
      .then(x => { return x; });

}

export function getDepartment() {

   return axios.get(host + `/api/getDepartment`)
      .then(x => { return x; });

}

export function getStudents() {
   return axios.get(host + `/api/getStudents`)
      .then(x => { return x; });
}

export function getStudent() {
   return axios.get(host + `/api/edit/getStudent`)
      .then(x => { return x; });
}
```

We used a GET request to get all the info for a specific table in our backend using MySQL and API from Hostinger. This api is the direct

contact of the frontend with all the data in the backend. This way we have an easy way to request info for certain tables.

The POST request was used to insert or update data in the backend.

React functions: UseEffect and useState

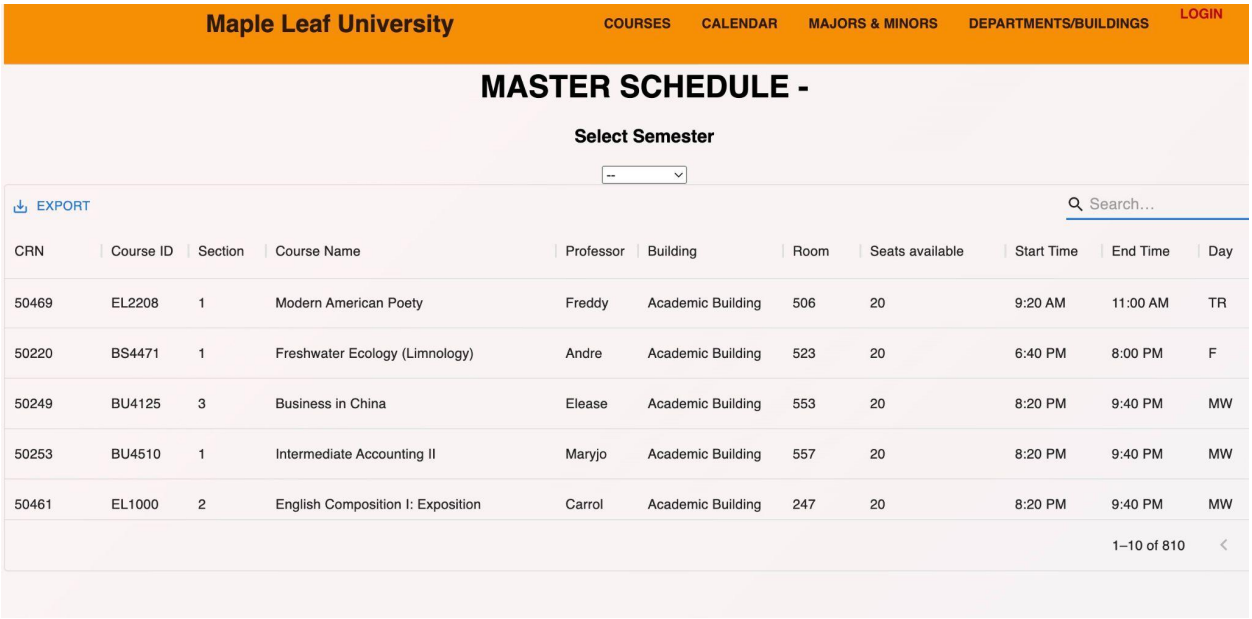
```
useEffect(() => {
    getStudent().then(x => {
        // filter with userId_us
        let StudentID = localStorage.getItem("StudentID");
        if(userType_us == "student"){

            let filteredData = x.data.filter(x => x.StudentID == userId_us);
            setStudent(filteredData)
        }
        else{
            let filteredData = x.data.filter(x => x.StudentID == StudentID);
            setStudent(filteredData)
        }
    });
}, [userId_us]);

useEffect(() => {
    getCourse_Prerequisites().then(x => {
        setCoursePrerequisites(x.data)
    });
}, []);

useEffect(() => {
    getCourses().then(x => {
        setCourses(x.data)
    });
}, []);

useEffect(() => {
    getUndergraduateStudentHistory().then(x => {
```

The useState and useEffect were used to render the pages each time the user made a change or selected an option. This way we keep the pages dynamic and the data running all the time.

Dynamic Boxes

```
return (
    <>
        <div className="slides2" id="top">
            <img src="images/fall22.png" alt="bacfacile" className="center" />
        </div>
        <h1> Register {studentData[0]?.FirstName} for a Course - SPRING23 </h1>

        <div style={{ height: 650, width: '100%' }}>
            <Box sx={{ height: 400, width: 1 }}>
                <DataGrid
                    rows={NextSemester}
                    columns={columns}
                    pageSize={10}
                    rowsPerPageOptions={[10]}
                    getRowId={(row) => row.CRN}

                    disableColumnFilter
                    disableColumnSelector
                    disableDensitySelector
                    disableColumnMenu
                    components={{ Toolbar: GridToolbar }}
                    componentsProps={{
                        toolbar: {
                            showQuickFilter: true,
                            quickFilterProps: { debounceMs: 500 },
                        },
                    }}
                />
            </Box>
        </div>
```

MUI reactions were used to create the dynamic tables that show the data. This MUI import have the following functions incorporated:
Filter data, refresh data, edit row, page size, etc.



Local Storage:

```
            Registration(userId_us, e)
            alert("Course added")
            window.location.reload();
        }
    }
    else {
        let localStudentID = localStorage.getItem("StudentID");
        // check if any student in studentHolds is equal to the userId_us and if it is, alert the user
        if (studentHolds.some(x => x.FacultyID == localStudentID)) {
            alert("You have a hold, please check your holds.");
        }
        // now chenct if that CRN is already in the NextSchedule
        else if (NextSchedule.some(x => x.CRN == e)) {
            alert("You are already registered for this course.");
        }
        // now if the Day and Time is already in the NextSchedule
        else if (NextSchedule.some(x => x.WeekDay == w && x.StartTime == s && x.EndTime == en)) {
            alert("You have a time conflict.");
        }
        // now if student.Year is full time and the number of courses is greater than 4
        else if (student.some(x => x.Year == "Full_Time" && NextSchedule.length >= 4)) {
            alert("Full Time conflict. You are already registered for 4 courses.");
        }
        // now if student.Year is part time and the number of courses is greater than 2
        else if (student.some(x => x.Year == "Part_Time" && NextSchedule.length >= 2)) {
            alert("Part Time conflict. You are already registered for 2 courses.");
        }


        // check if the course has prerequisites and if it does, check if the student has taken the prerequisites
        else if (coursePrerequisites.some(x => x.CourseID == course)) {
```

The local storage was used to save any studentID to use over the server. For example, to register a student for a course, the admin chose any student and his/her ID was saved in the local storage for the web. The

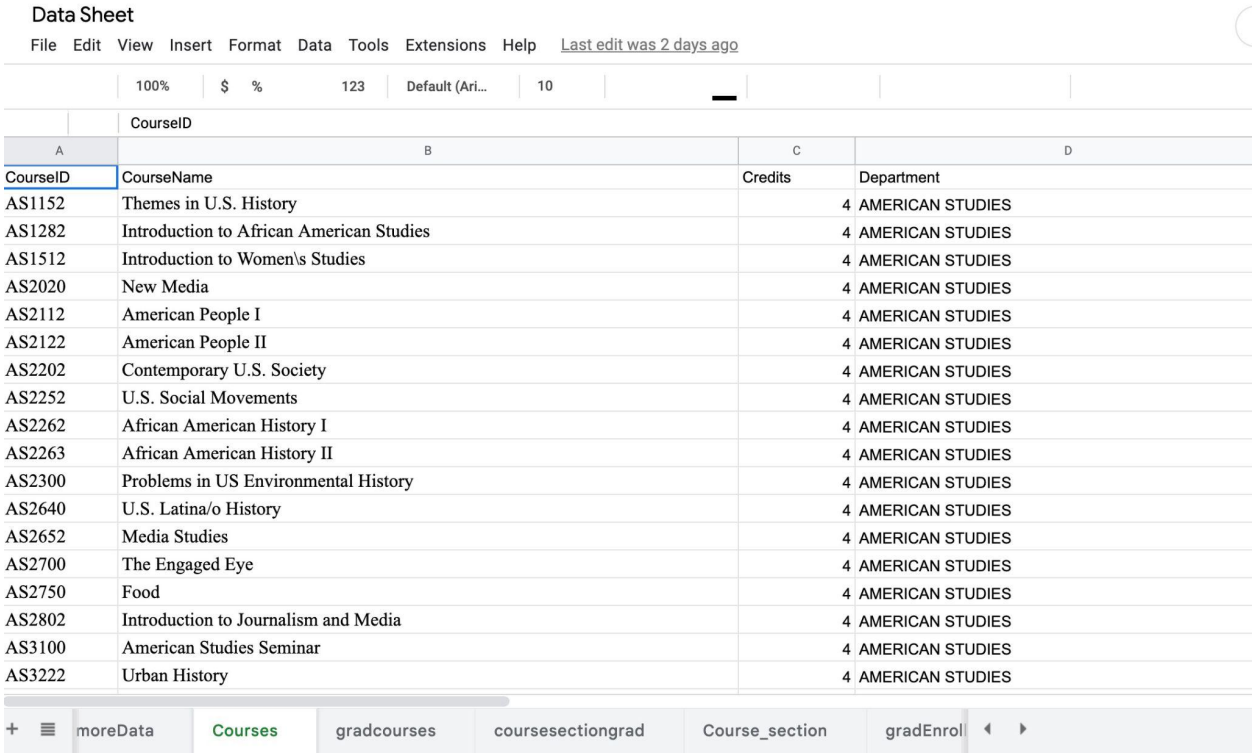
admin now has the ability to view or change anything about that student, until another student is chosen and a new local storage is stored.

APP.js

```
import CreateCourseSection from "./Panel/CreateCourseSection";
import MajorRequirements from "./MajorRequirements";
import MinorRequirements from "./MinorRequirements";
import AddDeptToFaculty from "./Panel/AddDeptToFaculty";
import ViewStudentScheduleForFaculty from "./Panel/ViewStudentScheduleForFaculty";
import CreateMajor from "./Panel/CreateMajor";


function App() {

  return (
    <div className="App">
      <NavBar />
      <Routes>
      <Route exact path="/" element={<Home />} />
      <Route exact path="/Buildings" element={<Buildings />} />
      <Route exact path="/Calender" element = {<Calender />} />
      <Route exact path = "/Courses" element = {<Courses />} />
      <Route exact path = "/Departments" element = {<Departments />} />
      <Route exact path = "/Majors" element = {<Majors />} />
      <Route exact path = "/Minor" element = {<Minor />} />
      <Route exact path = "/MasterSchedule" element = {<MasterSchedule />} />
      <Route exact path = "/Login_Page" element = {<Login_Page />} />
      <Route exact path = "/Course_Prerequisites" element = {<Course_Prerequisites />} />
      <Route exact path="/Profile" element={<Profile />} />
      <Route exact path="/Advisement" element={<Advisement />} />
      <Route exact path="/Attendance" element={<Attendance />} />
      <Route exact path="/ChangeMajor_Minor" element={<ChangeMajo_rMinor />} />
      <Route exact path="/Grades" element={<Grades />} />
      <Route exact path="/RegisterCourse" element={<RegisterCourse />} />
      <Route exact path="/Transcript" element={<Transcript />} />
      <Route exact path="/SearchFaculty" element={<SearchFaculty />} />
```

App is the main page for calling all the pages we need for the web. In this way we can make our server fully dynamic and avoid the repetition of the same pages for different roles. For example, the admin can view the master schedule, also students and faculty can view the master schedule, but only Admin can create a new course. Because of App.js and the tokens, we can do everything in one page called MasterSchedule.

Data



Data Sheet

File Edit View Insert Format Data Tools Extensions Help Last edit was 2 days ago

100% $ % 123 Default (Ari... 10

CourseID

	A	B	C	D
1	CourseID	CourseName	Credits	Department
2	AS1152	Themes in U.S. History	4	AMERICAN STUDIES
3	AS1282	Introduction to African American Studies	4	AMERICAN STUDIES
4	AS1512	Introduction to Women\s Studies	4	AMERICAN STUDIES
5	AS2020	New Media	4	AMERICAN STUDIES
6	AS2112	American People I	4	AMERICAN STUDIES
7	AS2122	American People II	4	AMERICAN STUDIES
8	AS2202	Contemporary U.S. Society	4	AMERICAN STUDIES
9	AS2252	U.S. Social Movements	4	AMERICAN STUDIES
10	AS2262	African American History I	4	AMERICAN STUDIES
11	AS2263	African American History II	4	AMERICAN STUDIES
12	AS2300	Problems in US Environmental History	4	AMERICAN STUDIES
13	AS2640	U.S. Latina/o History	4	AMERICAN STUDIES
14	AS2652	Media Studies	4	AMERICAN STUDIES
15	AS2700	The Engaged Eye	4	AMERICAN STUDIES
16	AS2750	Food	4	AMERICAN STUDIES
17	AS2802	Introduction to Journalism and Media	4	AMERICAN STUDIES
18	AS3100	American Studies Seminar	4	AMERICAN STUDIES
19	AS3222	Urban History	4	AMERICAN STUDIES

moreData Courses gradcourses coursesectiongrad Course_section gradEnrol

Data Sheet

File Edit View Insert Format Data Tools Extensions Help Last edit was 2 days ago

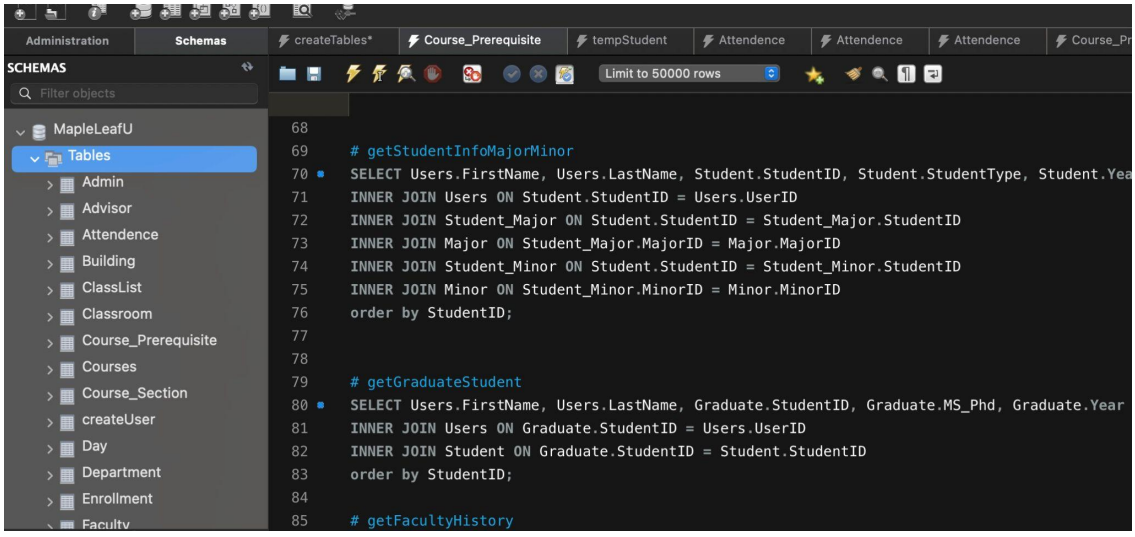
100% $ % 123 Default (Ari... 10

UserID

	A	B	C	D	E	F	G	H	
	UserID	FirstName	LastName	DOB	Street_address	ZipCode	City	State	Cour
	700001	Bernadette	Reed	1995-04-07	6548 Alysson Mountain Blvd.	87495	Port Cristopherport	Nevada	Unit
	700002	Randal	Bellamy	1995-04-28	04628 Hand Mountain court	36691	Port Lavernburgh	Minnesota	Unit
	700003	Carmelo	Cagle	1995-05-03	754 Anthony Hills St.	24636	New Keshaun	Nebraska	Unit
	700004	Ellen	Fontaine	1995-06-14	926 Dax Stream Dr.	34658	Gottliebmouth	Ohio	Unit
	700005	Ferdinand	Marx	1995-08-10	8739 Rohan Inlet Rd.	84238	Lake Carmelo	Florida	Unit
	700006	Noah	Patrick	1995-09-29	2161 Mante Rapid Ln.	2751	Runteview	Nevada	Unit
	700007	Maia	Morrow	1985-11-21	868 Friesen Rue place	94669	Hesselfurt	Illinois	Unit
	700008	Helene	Sawyer	1982-02-09	86531 Hiram Burgs ave.	51670	New Deondreburgh	Massachusetts	Unit
	700009	Melia	Mosier	1996-03-29	8248 Auer Bridge Blvd.	66661	West Maciefort	Illinois	Unit
	700010	Mariann	Mcclain	1997-12-26	008 Simeon Manors court	22799	Beaulahville	Massachusetts	Unit
	700011	Easter	Moran	1998-01-13	305 Konopelski Rapids St.	76107	South Austenton	Minnesota	Unit
	700012	Verline	Rankin	1967-01-16	27223 Calista Village Dr.	8769	West Mosefurt	North Carolina	Unit
	700013	Regine	Borden	1989-05-15	38452 Hammes Neck Rd.	72087	Mattview	West Virginia	Unit
	700014	Man	Bannister	1998-08-18	62235 Corkery Highway Ln.	41818	Hortenseberg	Kentucky	Unit
	700015	Heidy	Andrews	1999-02-03	799 Cecil Park place	61460	North Adrienneland	Florida	Unit
	700016	Margrett	Sammons	1999-04-30	043 Willa Roads ave.	99823	East Levi	Georgia	Unit
	700017	Velma	Blackmon	2000-04-26	6427 Hintz Fort Blvd.	57722	Lebsackside	Pennsylvania	Unit
	700018	Maude	Brink	2000-05-22	9189 Keeling Lane court	29544	Nikolaushaven	Rhode Island	Unit

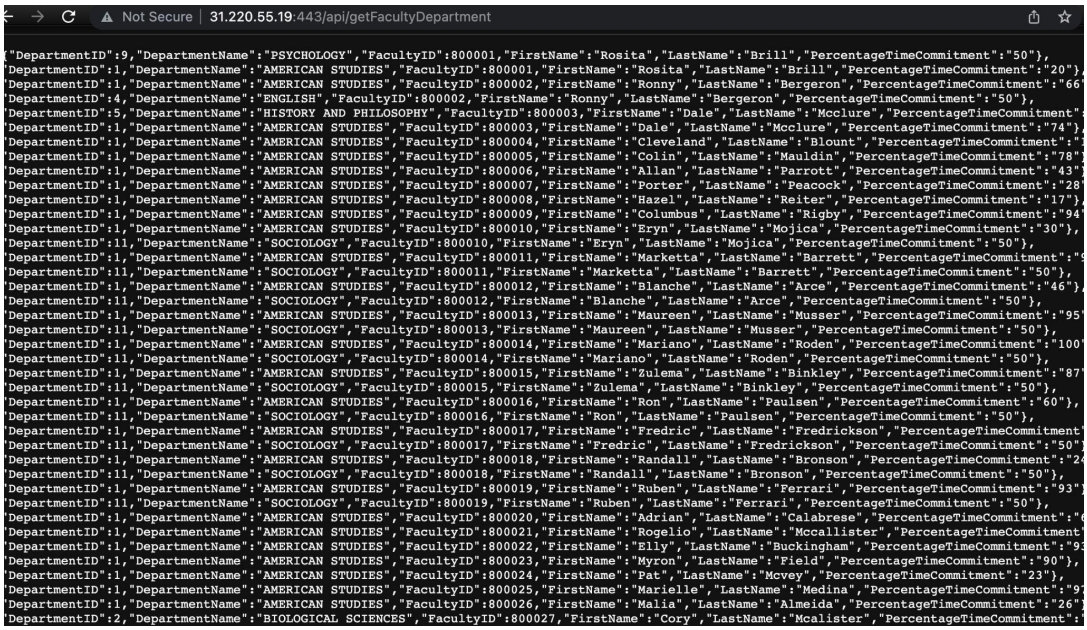
users | Login | Student | standing | Undergraduate | FullTime_UndergraduateStud

We used excel datasheets for storing the main part of our data for courses, users, login, majors/minors, time slot, Students, Faculty, etc. After creating these datasheets, they were converted to csv files and imported into the MySQL database for their specific table.

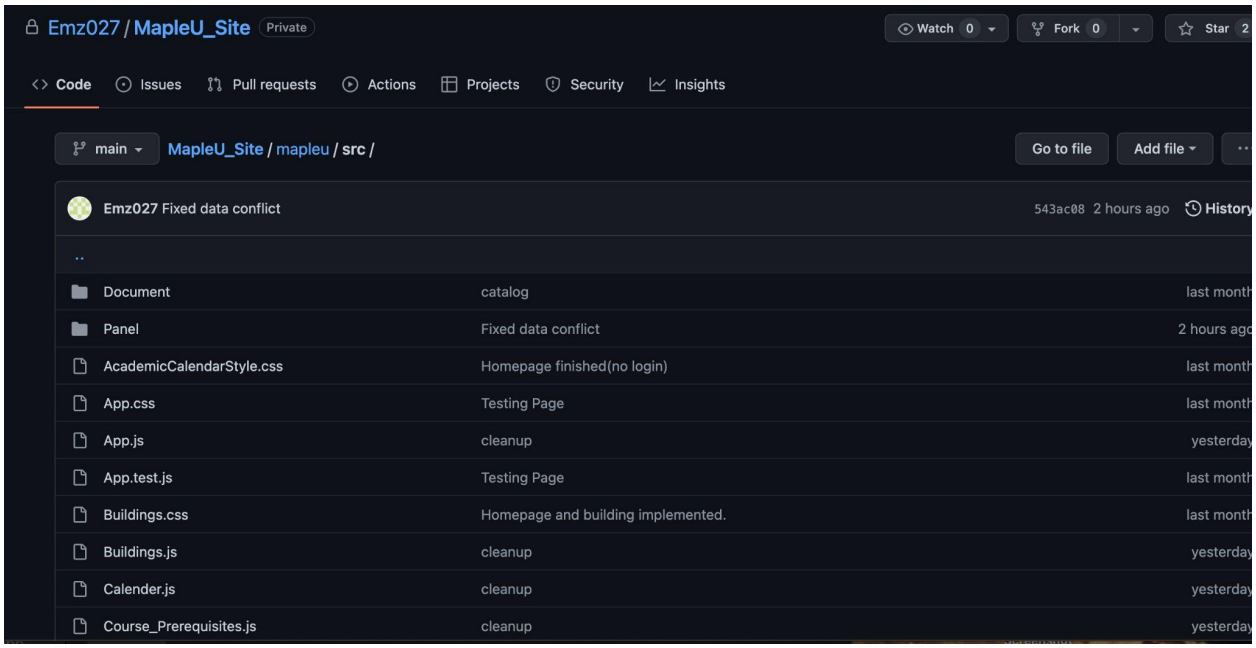


On MySQL we used INNER JOIN statements to join tables and create the api needed for the development of the web. In this way we created the master schedule, the Student major/student minor, and added students to courses.

Example of api/getFacultyDepartment:



Team Work:



GitHub was the principal tool in working on this project as a group. The entire frontend with every single file is in github, all the history of all the Push and Pull requests we made during the semester. As in the beginning of the semester, each member of the group worked locally for simpler coding and testing. Therefore, github was the connection and communication for all of us in the development of the web.