

CPSC 471
Database Management System
Winter 2022

Final Report
My Better UCalgary

April 18th, 2022

Sadia Khan
Sam Farzamfar
Jarín Thundathil

Repository Link:

<https://github.com/sadiatasneemkhan/My-Better-UCalgary>

Abstract

The existing user interface for the MyUCalgary web portal lacks key features that could vastly improve the experience of students navigating course selection and enrollment. Here, we have developed a basic implementation of a scalable database system which addresses the shortcomings of the present MyUcalgary portal. Our system uses an Express Backend to handle server and API functions to access from a mySQL database, and a javascript front-end that utilizes the react framework. Our system uses mock data for a course enrollment database, and offers functionalities for two types of end users: students and system administrators. The aim of this project was to demonstrate that an improved system of course registration and schedule building could be developed for students at the UofC. Though not as full featured as a deployable system, the foregoing results show that our implementation is capable of handling the core functions of a course enrollment database, and remedies some of the issues with the present MyUCalgary experience.

Introduction

Our database was designed to implement a course selection and registration system that mirrors that of the existing MyUcalgary system. We sought to improve upon the user experience of the course selection tool and provide our end users - most of whom are students - with new, integrated tools including an improved GPA calculator and degree planner. We chose this project because the current MyUcalgary course registration system is not user-friendly, and many core functionalities are implemented in a non-intuitive fashion. There were also features not present in the current system, including a unified degree planner and advanced GPA calculator that we sought to implement in our database.

Of course, we understood that due to time and resource constraints our project would be limited in scope relative to a deployable course registration system. Instead, we designed our project as a proof of concept showing that improvements to the existing system were possible, and sought to implement new features we deemed useful.

Project Description

Our system uses an Express middleware layer and a React front-end to implement a basic mySQL database in a Node.js runtime environment. Our database stores mock course enrollment

data for our registration system that has different functionalities for two types of users: students and administrators. The mySQL backend includes tables for 10 entities and relationships as outlined in our relational model (See section titled 'Implementation'). The Express middleware layer handles the API and server-side functions for the database. There are a total of twelve API endpoints that are discussed in detail in the 'API Documentation' section of this report. The front end for the webpages are written in javascript using the React framework. There are a total of 20 pages on the front end. A basic schematic for the project can be seen in the figure below:

Project Design

Our project is designed with two different end-users in mind: students and system administrators. These users utilize much of the same data but have different core functionalities that they perform with it. Those functionalities are described for each below:

Student:

Students should have functionalities relative to only their individual student accounts, unlike system administrators who should be able to manage data and make changes across multiple accounts. The core functionalities include the following:

View Transcript:

Students should be able to view their individual transcripts, which are stored in the 'GRADES' table of the mySQL database. These can be access with the api grades/getGrades API.

Calculate Grade:

Students should be able to calculate their grades for an ongoing course. This page can be accessed through the student dashboard under the 'Grade Calculator' heading. When clicked, they are taken to a page where they can input the weights and grades achieved of ongoing courses and calculate the grades needed on remaining components. This functionality does not require accessing the database and the returned response can be computed directly by the server.

View Schedule:

The student can view their schedule by clicking on the 'schedule' tab of the student dashboard. The schedule will display the times and dates of only the courses which the student is currently enrolled in. This data is retrieved from the ENROLLED_IN table joined with the

COURSE table. This table will retrieve all courses of a given student that they are currently enrolled in. This feature is not functional because

Enroll into Course:

The student can enroll into a course from the 'Enroll in a Course' tab on their student dashboard. From here the student can manually enter the name of the course they wish to enroll in. If this matches the name of an ongoing course, the student can choose to enroll in it. The database table needed here is the ENROLLED_IN table.

Drop a Course:

This happens from the 'Drop a Course' tab on the student dashboard. Once clicked, the user will navigate to a page where they can input the name of the course that they want to Drop and this will call an API to drop the Course_name and S_UCID combo in the ENROLLED_IN table.

View Degree Tracker:

Degree trackers allow users to see a brief explanation of what major and minors they are currently in, and which courses of their degree they have already completed.

System Admin:

System administrators should have functionalities which can add and remove courses, enter grades for any students.

View student Transcript:

System administrators can do this from the 'View students grade' tab of the admin dashboard. The admin can then get the students transcript ID using their sys UCID and the students UCID with the API endpoint admin/studentTid.

Add Grade to Transcript:

System admins can do this from the add a student grade tab on the student dashboard. After getting the students Tid with the API described in the section above, they can then add a grade to the student's transcript using the endpoint admin/addGrade. The new grade value will be inserted into the table GRADE.

Create Course/Edit course/Delete course:

These functions can be accessed from the update course tab of the admin dashboard. The admin can navigate to this page and select one of the on-screen buttons to perform their desired function. New courses are created using the course/createCourse API endpoint, they are edited using the course/editCourse API endpoint, and deleted using the course/deleteCourse API endpoint. New courses will be inserted into the COURSE table, as well as into the OFFERS table of the database. Modifications will be performed on the COURSE table.

Transaction Collection

Admin.js

```
router.post('/createadmin',(req,res) =>{

  let fName = req.query.First_name;
  let lName = req.query.Last_name;
  let UCID = req.query.UCID;
  let pass = req.query.pass;
  let confpass = req.query.Confirmed_pass;
  let deptname = req.query.Dept_name;
  let depthead = req.query.Dept_head;
  let date = req.query.date;
  let properDate =
date.substring(0, 2) + "/" + date.substring(2, 4) + "/" + date.substring(4);

  let test = false;

  if (pass === confpass){

    let query = `INSERT INTO ACCOUNT VALUES (${UCID},'${pass}')`
    db.query(query, (err, result) => {
      if (err) throw err;
      test = true;
    });

    let query3 = `INSERT INTO SYSTEM_ADMIN VALUES
(${UCID},'${fName}','${lName}','${properDate}','${UCID}')`
    db.query(query3, (err, result) => {
      if (err) throw err;
      test = true;
    });
  }
})
```

```

});

let query2 = `INSERT INTO DEPARTMENT VALUES('${deptname}','${depthead}', ${UCID})`
db.query(query2, (err, result) => {
  if (err) throw err;
  if(test === true){
    res.send({'UCID': UCID});
  }
});
}

else {
  res.send('Passwords did not match');
}
});

router.get('/studentTid',(req,res) =>{
  let sUCID = req.query.Sid;
  let query = `SELECT TRANSCRIPT.T_ID FROM (TRANSCRIPT JOIN STUDENT ON
TRANSCRIPT.S_UCID = STUDENT.UCID)
WHERE UCID = ${sUCID}`;
  db.query(query, (err, result) => {
    if (err) throw err;
    else return res.json(result);
  });
});

router.get("/getadmins", (req, res) => {
  let query = `SELECT * FROM ACCOUNT`;
  db.query(query, (err, result) => {
    if (err) throw err;
    else return res.send(result);
  });
});

router.post('/addGrade',(req,res) => {
  let sysUCID = req.query.Adminid;
  let sUCID = req.query.Sid;
  let course = req.query.Course_name;

```

```

let Course_sem = req.query.Sem;
let letter = req.query.Lgrade;
let percent = req.query.Pgrade;
let t_id = 0;

let query0 = `SELECT TRANSCRIPT.T_ID FROM (TRANSCRIPT JOIN STUDENT ON
TRANSCRIPT.S_UCID = STUDENT.UCID)
WHERE UCID = ${sUCID}`;
db.query(query0, (err, result) => {
  if (err) throw err;
  var str = JSON.stringify(result[0]);
  str = str.replace(/[\^d.-]/g, "");
  t_id = str;

  if (t_id !== 0){

    let query = `INSERT INTO GRADE VALUES(${sUCID}, '${course}', '${Course_sem}', ${percent},
'${letter}', ${t_id}, ${sysUCID})`

    db.query(query, (err) => {
      if (err) throw err;
      else return res.send({'UCID': sysUCID});
    });
  }
});
});

```

Student.js

```

router.get("/checkLogin", (req, res) => {
  let ucid = req.query.UCID;
  let pass = req.query.password;
  let query = `SELECT * FROM ACCOUNT WHERE UCID='${ucid}' AND password='${pass}'`;
  db.query(query, (err, result) => {
    if (err) throw err;
    if (result.length > 0) {
      if(ucid.charAt(0) == 1){
        res.send({ account: 'Admin' });
      }
    }
  })
}

```

```

    else res.send({ account: 'Student' });
  } else {
    res.send({ account: 'ERROR' });
  }
});
});

router.get('/studentInfo',(req,res)=>{

  let ucid = req.query.UCID;

  let query = `SELECT COUNT(*) as appears FROM MINORS_IN WHERE UCID = ${ucid}`
  db.query(query, (err, result) => {
    if (err) throw err;
    else return res.json(result);
  });
});

router.get('/studentInfomm',(req,res)=>{

  let ucid = req.query.UCID;

  let query = `SELECT STUDENT.UCID, MAJORS_IN.Program as Major, MINORS_IN.Program as Minor,
MINORS_IN.Year_program as Minor_year, MAJORS_IN.Year_program as Major_year FROM (STUDENT
JOIN MAJORS_IN ON STUDENT.UCID= MAJORS_IN.UCID) JOIN MINORS_IN ON STUDENT.UCID =
MINORS_IN.UCID WHERE STUDENT.UCID= ${ucid}`
  console.log(query);
  db.query(query, (err, result) => {
    if (err) throw err;
    else {
      console.log(result);
      return res.json(result);
    }
  });
});

router.get('/studentInfom',(req,res)=>{

  let ucid = req.query.UCID;

```



```

    let query = `SELECT STUDENT.UCID, MAJORS_IN.Program as Major, MAJORS_IN.Year_program as
Major_year FROM (STUDENT JOIN MAJORS_IN ON STUDENT.UCID= MAJORS_IN.UCID)`
    db.query(query, (err, result) => {
      if (err) throw err;
      else return res.json(result);
    });
  });

```

// endpoint 3

```

router.post("/createStudent", (req, res) => {
  let fName = req.query.First_name;
  let lName = req.query.Last_name;
  let date = req.query.date;
  let addr = req.query.Address;
  let Postal = req.query.Postalcode;
  let City = req.query.City;
  let Province = req.query.Province;
  let Country = req.query.Country;
  let UCID = req.query.UCID;
  let Phone = req.query.Phone_number;
  let pass = req.query.pass;
  let confpass = req.query.Confirmed_pass;
  let prog = req.query.Program;
  let deptname = req.query.Dept_name;
  let yop = req.query.Year_of_program;
  let min = req.query.Minor;
  let dept_min = req.query.minor_dept;
  let yom = req.query.Year_of_minor;

  let properDate =
date.substring(0, 2) + "/" + date.substring(2, 4) + "/" + date.substring(4);

  if (pass === confpass){

    let query = `INSERT INTO ACCOUNT VALUES(${UCID},'${pass}')`;

```

```

    let query2 = `INSERT INTO STUDENT VALUES
(${UCID},'${fName}','${lName}','${properDate}','${Phone}','${addr}','${Postal}','${City}','${Province}','${
Country}','${UCID})`;

    let query3 = `INSERT INTO MAJORS_IN VALUES(${UCID},'${deptname}','${yop}','${prog})`;

    // adds to account
    db.query(query, (err) => {
        if (err) throw err;
        else {
            console.log("Account added...");
        }
    });

    // adds to student
    db.query(query2, (err) => {
        if (err) throw err;
        else {
            console.log("Student added...");
        }
    });

    // adds to minor if applicable
    if (min !== "" || yom !== "" || dept_min !== "") {
        let query4 = `INSERT INTO MINORS_IN
VALUES(${UCID},'${dept_min}','${yom}','${min})`;

        db.query(query4, (err) => {
            if (err) throw err;
            else {
                console.log("Minor added...");
            }
        });
    }

    // adds to major
    db.query(query3, (err) => {
        if (err) throw err;
        else {
            console.log("Major added...");
        }
    });

```

```

    return res.send({ UCID: UCID });
  }

  else {
    return res.send('Passwords did not match' );
  }
});

// endpoint 9
router.get("/getGrades", (req, res) => {
  let UCID = req.query.UCID;

  let query = `SELECT GRADE.S_UCID, GRADE.Course_Name, GRADE.Letter_grade,
GRADE.course_semester FROM GRADE WHERE S_UCID = '${UCID}'`;

  //generic query
  db.query(query, (err, result) => {
    if (err) throw err;
    else return res.json(result);
  });
});

```

Course.js

```

router.get("/courses", (req, res) => {
  let query = "SELECT * FROM COURSE";
  db.query(query, (err, result) => {
    if (err) throw err;
    // output parsing
    console.log(result);
    res.json(result);
  });
});

// endpoint 6

router.post("/createCourse", (req, res) => {
  console.log(req.query);
  console.log('got to createCourse API.')
  let name = req.query.name;

```

```

let sem = req.query.semester;
let dept = req.query.dept;
let ifName = req.query.First_name;
let ilName = req.query.Last_name;
let des = req.query.desc;
let status = req.query.status;
let building = req.query.building;
let roomNum = req.query.room_num;
let day = req.query.day;
let time = req.query.time;
let size = req.query.Classsize;
let cap = req.query.cap;
let ID = req.query.Ucid;

let query = `
INSERT INTO COURSE

VALUES('${name}','${sem}','${des}','${ifName}','${ilName}','${building}','${roomNum}','${day}','${time}',
'${size}','${cap}','${status}','${ID}');

let query2 = `INSERT INTO OFFERS VALUES ('${name}','${dept}');
let query3 = `SELECT * FROM COURSE WHERE name = '${name}' AND Semester = '${sem}';

db.query(query, (err, result) => {
  if (err) throw err;
  else console.log("Course created...");
});

db.query(query2, (err, result) => {
  if (err) throw err;
  else console.log("Offers created...");
});
db.query(query3, (err, result) => {
  if (err) throw err;
  else return res.json(result);
});
});
// endpoint 7

```

```
router.put("/editCourse", (req, res) => {
  let orgname = req.query.orgName;
  let orgsem = req.query.orgSem;

  let query = `
UPDATE COURSE
SET`

  let name = req.query.name;
  if(name !== undefined){
    query += ` name='${name}',`
  }
  let sem = req.query.semester;
  if(sem !== undefined){
    query += ` Semester='${sem}',`
  }

  let dept = req.query.dept;
  if(dept !== undefined){
    query += ` Semester='${dept}',`
  }
  let ifName = req.query.First_name;
  if(ifName !== undefined){
    query += ` Instructor_fname='${ifName}',`
  }
  let ilName = req.query.Last_name;
  if(ilName !== undefined){
    query += ` Instructor_lname='${ilName}',`
  }
  let des = req.query.desc;
  if(des !== undefined){
    query += ` Description='${des}',`
  }
  let status = req.query.status;
  if(status !== undefined){
    query += ` Status='${status}',`
  }
  let building = req.query.building;
```

```
if(building !== undefined){
  query += ` Building='${building}',`
}
let roomNum = req.query.room_num;
if(roomNum !== undefined){
  query += ` Room_number='${roomNum}',`
}
let day = req.query.day;
if(day !== undefined){
  query += ` Day='${day}',`
}
let time = req.query.time;
if(time !== undefined){
  query += ` Time='${time}',`
}
let size = req.query.Classsize;
if(size !== undefined){
  query += ` Current_size='${size}',`
}
let cap = req.query.cap;
if(cap !== undefined){
  query += ` Max_capacity='${cap}',`
}
let ID = req.query.Ucid;
if(ID !== undefined){
  query += ` Semester='${ID}',`
}

query = query.substring(0,query.length-1);
query += ` \n WHERE Name ='${orgname}' AND Semester ='${orgsem}';
`;
let query2 = `UPDATE OFFERS
SET`
if(name !== undefined){
  query2 += ` name='${name}',`
}

if(sem !== undefined){
  query2 += ` Semester='${sem}',`
}
```

```

    }
    query2 = query2.substring(0,query2.length-1);
    query2 += `WHERE Course_name ='${orgname}' AND Dept_name ='${dept}';
    `;
let query3 = `SELECT * FROM COURSE WHERE name = '${name ? name: orgname}' AND Semester =
'${sem ? sem: orgsem}';`;
db.query(query, (err) => {
  if (err) throw err;
  else console.log("Course Updated!");
});
if(!(name === undefined) || !(sem === undefined)){
  db.query(query2, (err) => {
    if (err) throw err;
    else console.log("Offers updated!");
  });
}
db.query(query3, (err, result) => {
  if (err) throw err;
  else return res.json(result);
});
});

//endpoint 5
router.get("/courseByAdminID", (req, res) => {
  let ucid = req.query.UCID;

  let query = `
  SELECT Course_name,Semester,Instructor_lname,Status FROM (((SYSTEM_ADMIN JOIN
DEPARTMENT ON UCID=Sys_UCID) JOIN OFFERS ON DEPARTMENT.name= Dept_name) JOIN
COURSE ON COURSE.name=Course_name)
WHERE ${ucid} = SYSTEM_ADMIN.UCID;
  `;
  db.query(query, (err, result) => {
    if (err) throw err;
    // output parsing
    console.log(result);
    res.json(result);
  });
});
});

```

```

router.get("/courseByStudentID", (req, res) => {
  let ucid = req.query.UCID;
  console.log("Called")
  let query = `
    SELECT * FROM GRADE WHERE S_UCID=${ucid};
  `;
  db.query(query, (err, result) => {
    if (err) throw err;
    // output parsing
    console.log(result);
    res.json(result);
  });
});

//endpoint 8
router.delete("/deleteCourse", (req, res) => {
  let name = req.query.name;
  let sem = req.query.semester;

  let query = `DELETE FROM COURSE
    WHERE name = '${name}' AND Semester = '${sem}'`;
  db.query(query, (err, result) => {
    if (err) throw err;
    // output parsing

    console.log("Course Deleted...");
  });
  return res.json({"UCID": req.query.Id});
});

```

Schedule.js

```

router.post('/enroll',(req,res)=>{

  let name = req.query.Course_name;
  let sem = req.query.Semester;
  let S_UCID = req.query.Sucid;
  let sys_UCID = req.query.Sysucid;

```



```

let query = `INSERT INTO ENROLLED_IN VALUES (${S_UCID},'${name}','${sem})`

db.query(query,(err,result)=>{
  if(err) throw err;
  else return res.send({'UCID' : sys_UCID});
});

})

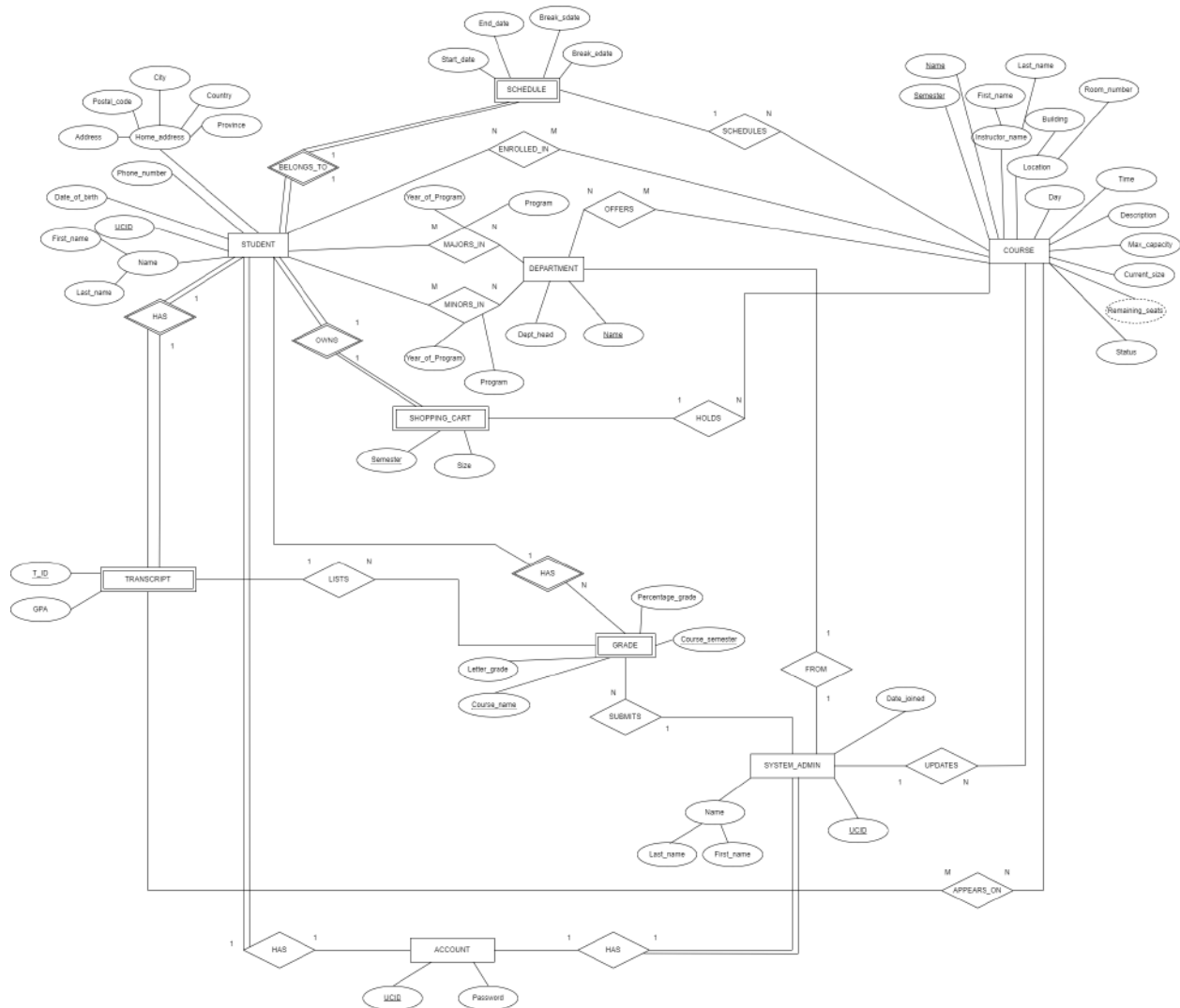
router.delete('/drop',(req,res)=>{

  let name = req.query.Course_name;
  let sem = req.query.Semester;
  let S_UCID = req.query.Sucid;
  let sys_UCID = req.query.Sysucid;
  let query = `DELETE FROM ENROLLED_IN WHERE S_UCID=${S_UCID} AND Course_name =
'${name}' AND Semester = '${sem}'`

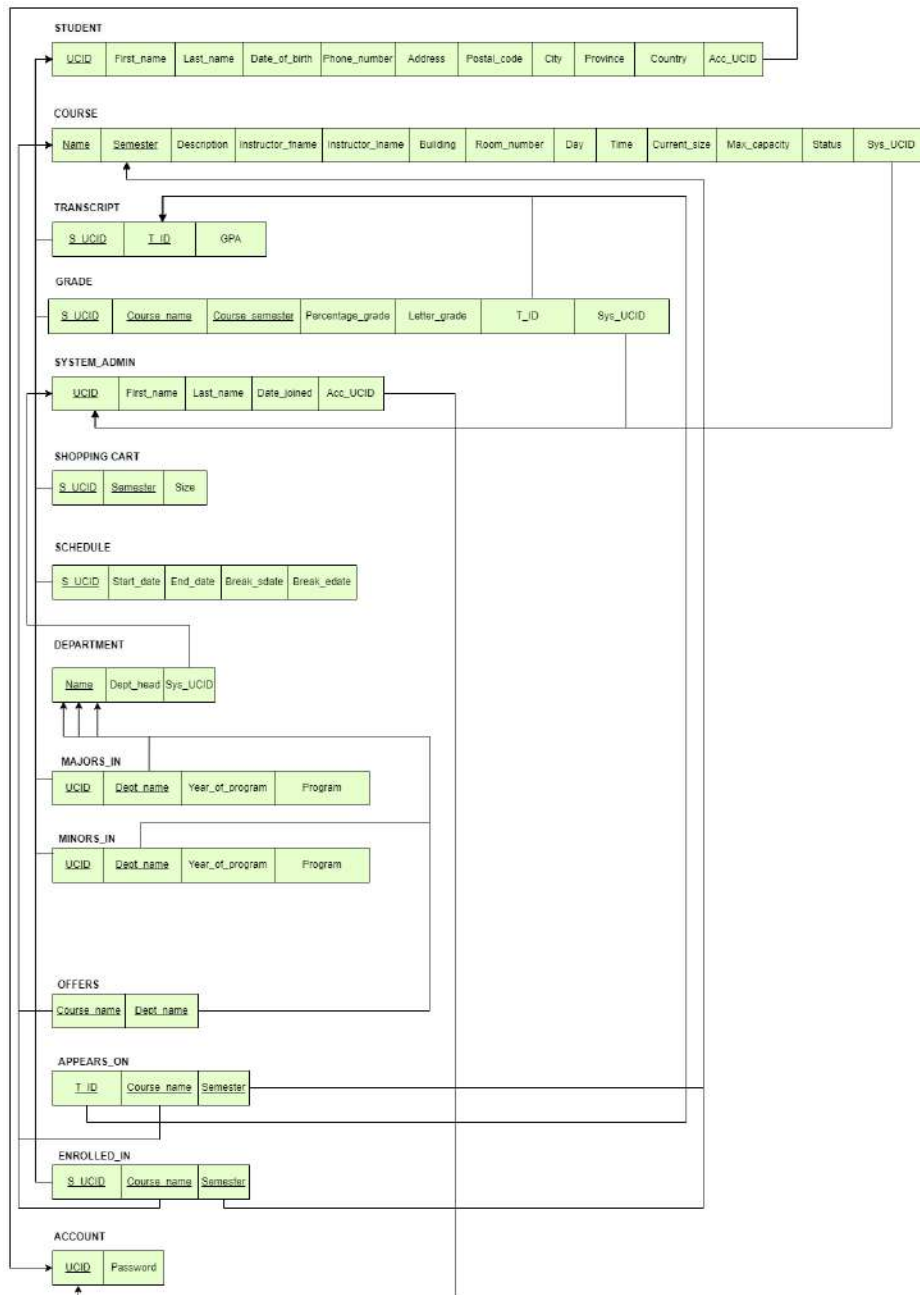
  db.query(query,(err,result)=>{
    if(err) throw err;
    else return res.send({'UCID' : sys_UCID});
  });
})

```

EERD:



The email attribute from STUDENT and grade scale from COURSE were removed due to errors we were facing in mySQL. Also ACCOUNT entity was added in order to keep track of a user's ID and password.



DBMS and Queries:

We chose MySQL as our DBMS because of the familiarity with the system. All of our group members were familiar with how to write the query statements because of the lecturer notes, and some members have used MySQL for previous projects. Also, MySQL allows easy viewing for each table which is very useful.

The SQL statements are as follows. These statements were reused for each entity as needed, with the variable and table names changed as necessary. All SQL statements are included at the bottom of the document for reference. The example is shown from the Student entity:

Matching student account and password:

```
`SELECT * FROM ACCOUNT WHERE UCID ='${ucid}' AND password='${pass}'`
```

Getting a student's information (minors):

```
`SELECT COUNT(*) as appears FROM MINORS_IN WHERE UCID = ${ucid} `
```

Getting a student's minor and major:

```
SELECT STUDENT.UCID, MAJORS_IN.Program as Major, MINORS_IN.Program as Minor,  
MINORS_IN.Year_program as Minor_year, MAJORS_IN.Year_program as Major_year FROM (STUDENT  
JOIN MAJORS_IN ON STUDENT.UCID= MAJORS_IN.UCID) JOIN MINORS_IN ON STUDENT.UCID =  
MINORS_IN.UCID WHERE STUDENT.UCID= ${ucid} `
```

Getting only a student's major:

```
`SELECT STUDENT.UCID, MAJORS_IN.Program as Major, MAJORS_IN.Year_program as Major_year  
FROM (STUDENT JOIN MAJORS_IN ON STUDENT.UCID= MAJORS_IN.UCID)`
```

Getting a student's grades:

```
`SELECT GRADE.S_UCID, GRADE.Course_Name, GRADE.Letter_grade, GRADE.course_semester FROM  
GRADE WHERE S_UCID = '${UCID}';
```

Creating a student:

```
`INSERT INTO ACCOUNT VALUES(${UCID},'${pass}')`;  
`INSERT INTO STUDENT VALUES  
(${UCID},'${fName}','${lName}','${properDate}','${Phone}','${addr}','${Postal}','${City}','${Province}','${  
Country}','${UCID});`;  
`INSERT INTO MAJORS_IN VALUES(${UCID},'${deptname}','${yop}','${prog}')`;
```

Create an admin account:

```
`INSERT INTO ACCOUNT VALUES(${UCID},'${pass}')`
`INSERT INTO SYSTEM_ADMIN VALUES (${UCID},'${fName}','${lName}','${properDate}','${UCID}')`
`INSERT INTO DEPARTMENT VALUES('${deptname}','${depthead}', ${UCID})`
```

Admin getting a students transcript:

```
`SELECT TRANSCRIPT.T_ID FROM (TRANSCRIPT JOIN STUDENT ON TRANSCRIPT.S_UCID = STUDENT.UCID)
```

Getting a list of all admins:

```
`SELECT * FROM ACCOUNT`;
```

Admin adding a grade to a student's transcript:

```
`SELECT TRANSCRIPT.T_ID FROM (TRANSCRIPT JOIN STUDENT ON TRANSCRIPT.S_UCID = STUDENT.UCID)
```

```
`INSERT INTO GRADE VALUES(${sUCID}, '${course}', '${Course_sem}', ${percent}, '${letter}', ${t_id}, ${sysUCID})`
```

Getting all courses:

```
"SELECT * FROM COURSE";
```

Create a course:

```
INSERT INTO COURSE
VALUES('${name}','${sem}','${des}','${ifName}','${ilName}','${building}','${roomNum}','${day}','${time}',
'${size}','${cap}','${status}','${ID}');';
let query2 = `INSERT INTO OFFERS VALUES ('${name}','${dept}')`;
let query3 = `SELECT * FROM COURSE WHERE name = '${name}' AND Semester = '${sem}';`;
```

Edit a course (if statements are appended if those attributes of a course are changed):

```
UPDATE COURSE
SET`
let name = req.query.name;
if(name !== undefined){
  query += ` name='${name}',`
}
let sem = req.query.semester;
if(sem !== undefined){
  query += ` Semester='${sem}',`
```

```
}

let dept = req.query.dept;
if(dept !== undefined){
  query += `Semester='${dept}',`
}
let ifName = req.query.First_name;
if(ifName !== undefined){
  query += `Instructor_fname='${ifName}',`
}
let ilName = req.query.Last_name;
if(ilName !== undefined){
  query += `Instructor_lname='${ilName}',`
}
let des = req.query.desc;
if(des !== undefined){
  query += `Description='${des}',`
}
let status = req.query.status;
if(status !== undefined){
  query += `Status='${status}',`
}
let building = req.query.building;
if(building !== undefined){
  query += `Building='${building}',`
}
let roomNum = req.query.room_num;
if(roomNum !== undefined){
  query += `Room_number='${roomNum}',`
}
let day = req.query.day;
if(day !== undefined){
  query += `Day='${day}',`
}
let time = req.query.time;
if(time !== undefined){
  query += `Time='${time}',`
}
let size = req.query.Classsize;
```

```

if(size !== undefined){
  query += ` Current_size='${size}',`
}
let cap = req.query.cap;
if(cap !== undefined){
  query += ` Max_capacity='${cap}',`
}
let ID = req.query.Ucid;
if(ID !== undefined){
  query += ` Semester='${ID}',`
}

query = query.substring(0,query.length-1);
query += `
WHERE Name ='${orgname}' AND Semester ='${orgsem}';
`;
let query2 = `UPDATE OFFERS
SET`
if(name !== undefined){
  query2 += ` name='${name}',`
}

if(sem !== undefined){
  query2 += ` Semester='${sem}',`
}
query2 = query2.substring(0,query2.length-1);
query2 += `
WHERE Course_name ='${orgname}' AND Dept_name ='${dept}';
`;
let query3 = `SELECT * FROM COURSE WHERE name = '${name ? name: orgname}' AND Semester =
'${sem ? sem: orgsem}';`;

```

Get courses by admin ID (all courses managed by a given admin):

```

SELECT Course_name,Semester,Instructor_lname,Status FROM (((SYSTEM_ADMIN JOIN DEPARTMENT
ON UCID=Sys_UCID) JOIN OFFERS ON DEPARTMENT.name= Dept_name) JOIN COURSE ON
COURSE.name=Course_name)
WHERE ${ucid} = SYSTEM_ADMIN.UCID;

```

Get courses by student ID (get all courses of a given student):

```

SELECT * FROM GRADE WHERE S_UCID=${ucid};

```

Delete a course:

```
DELETE FROM COURSE
WHERE name = '${name}' AND Semester = '${sem}';
```

Enroll in a course:

```
INSERT INTO ENROLLED_IN VALUES (${S_UCID},${name},'${sem}')
```

Drop a course:

```
DELETE FROM ENROLLED_IN WHERE S_UCID=${S_UCID} AND Course_name = '${name}' AND
Semester = '${sem}'
```

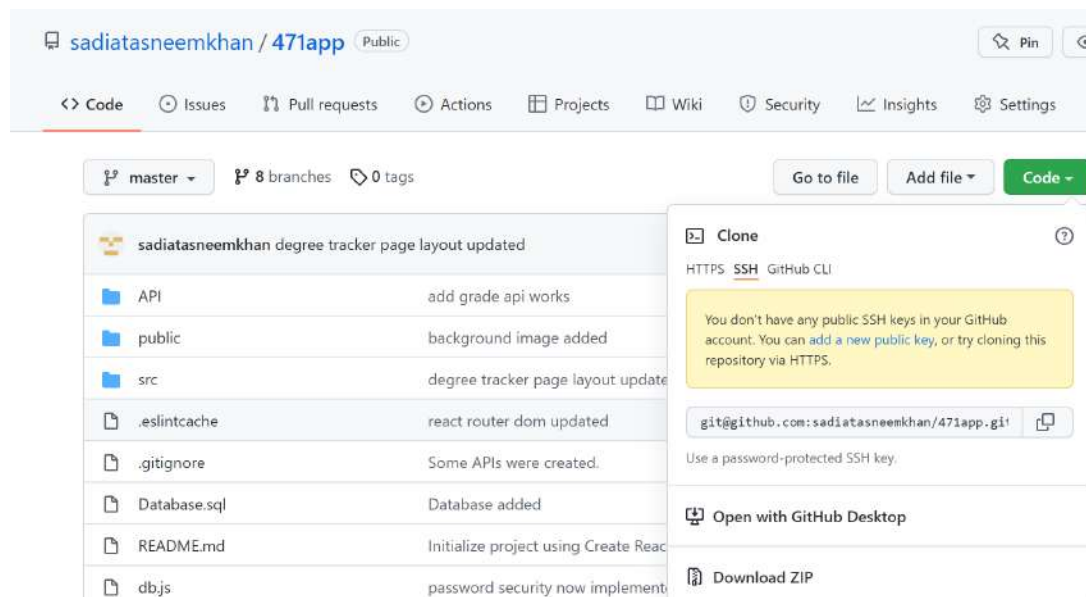
API Documentation:

The documentation for the APIs can be viewed here:

<https://documenter.getpostman.com/view/20541974/Uyr7HJX1>

User Guide:

In order to run our software directly from your device, you must have Node.js and mySQL installed beforehand, along with an IDE of your choosing. You can then download the repository from our github by pressing the green “Code” button and downloading the ZIP.



Extract the zip to any location of your choosing. Locate the Database.sql file and add into your MySQL program. To connect the MySQL database, go into the localuser.json file and edit the “user” and “password” fields to match your SQL server.

Example:

```
{  
  "user": "root",  
  "password": "password",  
  "port": 5001  
}
```

Although node-modules are included, if you face any troubles with them, you should install the following packages in your server folder:

To install any missing modules/packages:

In your IDE, make sure you are in the 471app folder. If not, please change your directory using the “cd” command.

Example: C:\Users\sadia\Documents\GitHub\471app>

Then open a new terminal and enter the following command to install the required dependencies:

npm i

Starting the application:

Within the 471app directory, open a new terminal and enter the following command to open the website (frontend):

npm start

Then in another new terminal, enter the following command to connect the server (database):

node server.js

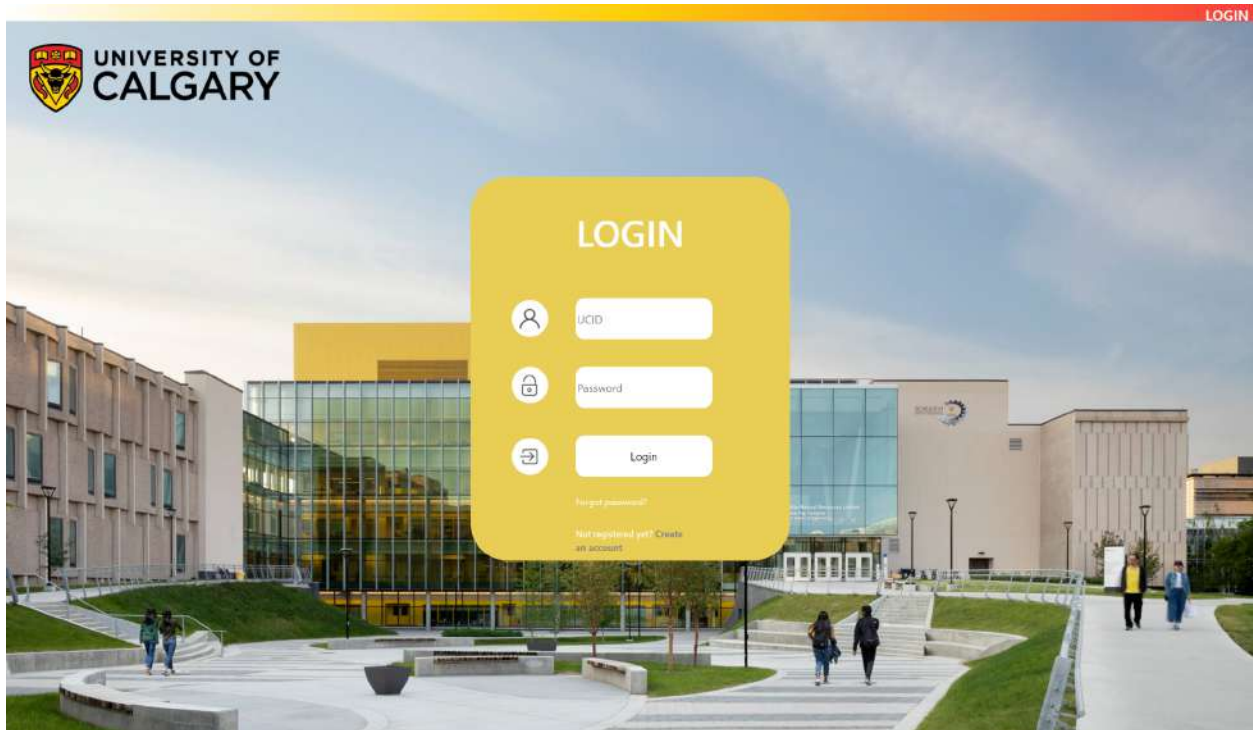
You should now be able to access our website on your local browser. The frontend (client) should be available at <http://localhost:3000> (or any other port) and the backend (server) should be available at <http://localhost:5001>.

Homepage

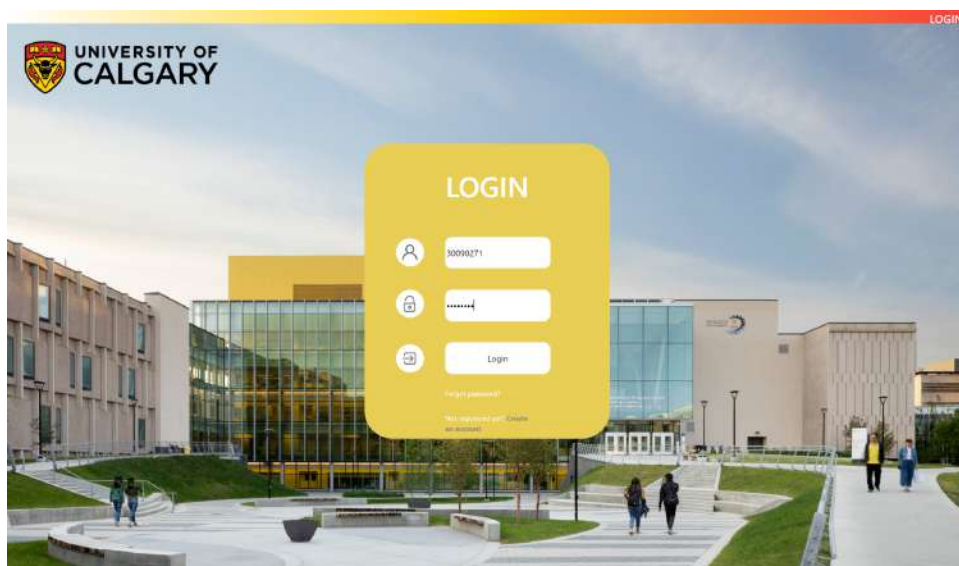


This is the homepage that all users, whether student or admin, are introduced to when opening our web application. From here, they can press the “login” button in the upper right corner.

Login Page

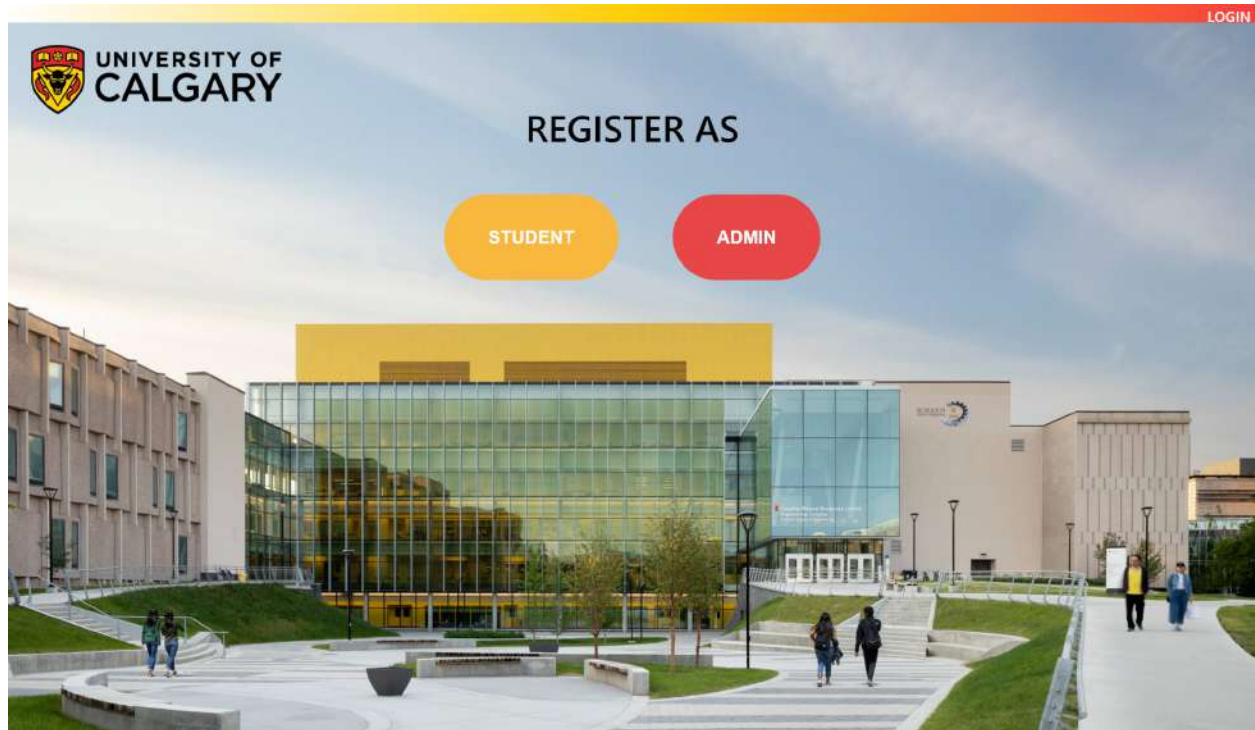


Once the login button is pressed from the homepage, the login menu appears. Here, users can login if they have an account already. Users are required to input their UCID and password into the correct boxes. If a user successfully logs in, they will be directed to their respective dashboard (either student or admin). If the user does not have an existing account, they can make one by pressing on the “Create an account” grey text.



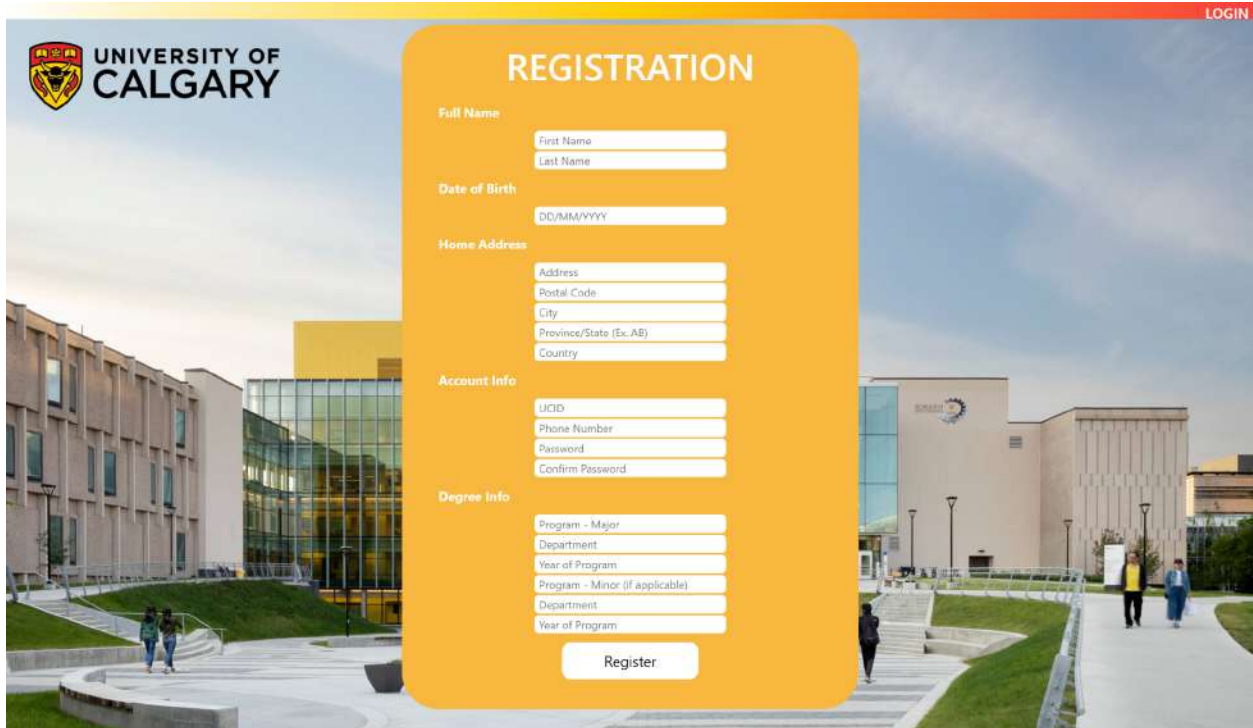
An example of a student, who already has an account stored in the database, logging in.

Registration Options Page



Before creating a new account, the user is required to pick what type of account they are registering for. Student accounts will be made by pressing the orange “Student” button, and the red button for admin.

Student Registration Page



The image shows a web page for the University of Calgary with a registration form overlay. The form is titled "REGISTRATION" and includes sections for Full Name, Date of Birth, Home Address, Account Info, and Degree Info. A "Register" button is at the bottom of the form. The background shows a campus scene with a large building and a walkway.

UNIVERSITY OF CALGARY

REGISTRATION

Full Name
First Name
Last Name

Date of Birth
DD/MM/YYYY

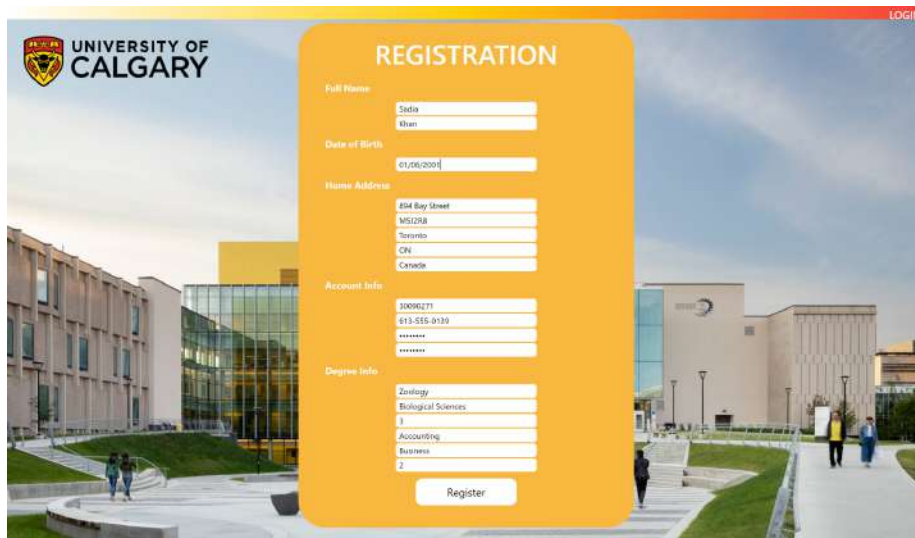
Home Address
Address
Postal Code
City
Province/State (Ex. AB)
Country

Account Info
UCID
Phone Number
Password
Confirm Password

Degree Info
Program - Major
Department
Year of Program
Program - Minor (if applicable)
Department
Year of Program

Register

After pressing the orange “Student” button on the registration option page, a student registration form will appear. Student’s are required to fill out every text box except the last three boxes, since not everyone has a minor. Once the required boxes are filled, they can press register. Then they will be taken to the student dashboard page. Next time when they come back to our application, they will not have to register again, and can just simply login using their UCID and their chosen password.



The image shows the same registration form as above, but with example data filled in. The data is as follows:

UNIVERSITY OF CALGARY

REGISTRATION

Full Name
Sadia
Khan

Date of Birth
01/08/2000

Home Address
254 Bay Street
M5S2R8
Toronto
ON
Canada

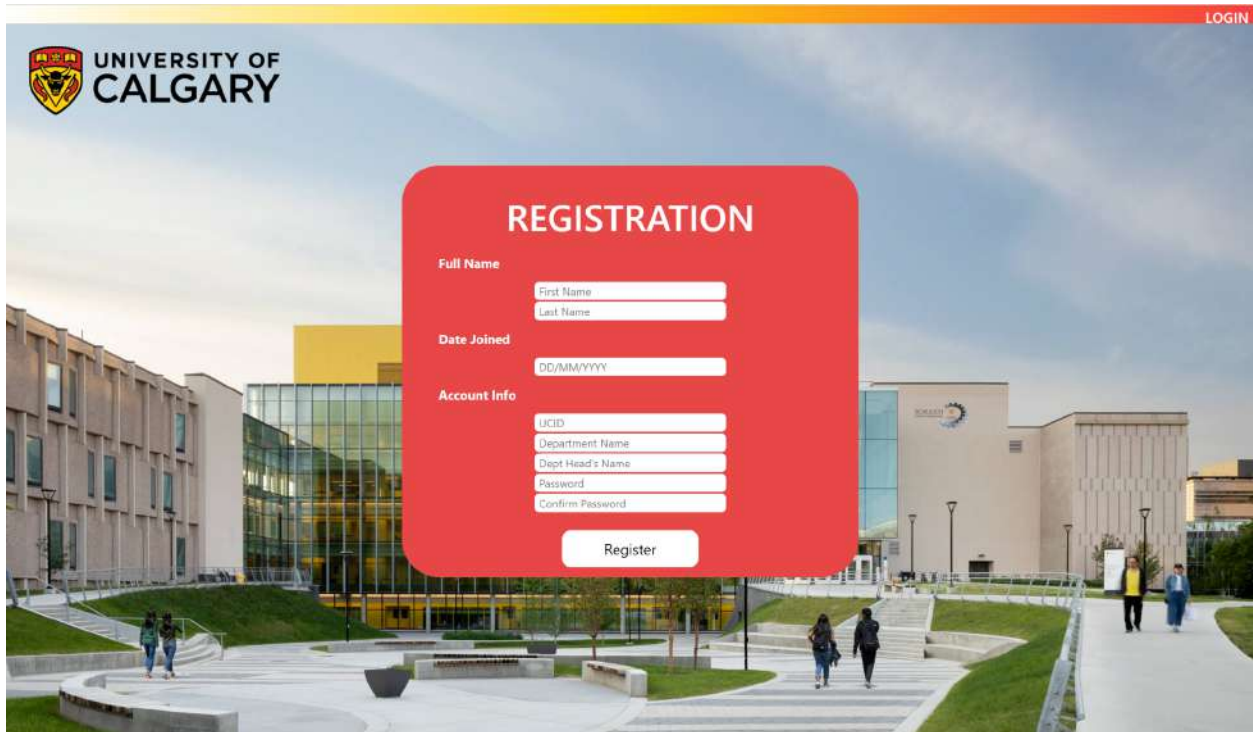
Account Info
10090271
613-555-0130

Degree Info
Zoology
Biological Sciences
1
Accounting
Business
2

Register

An example of a student filling in all the required information with the correct syntax. This student also has a minor, thus included it.

Admin Registration Page

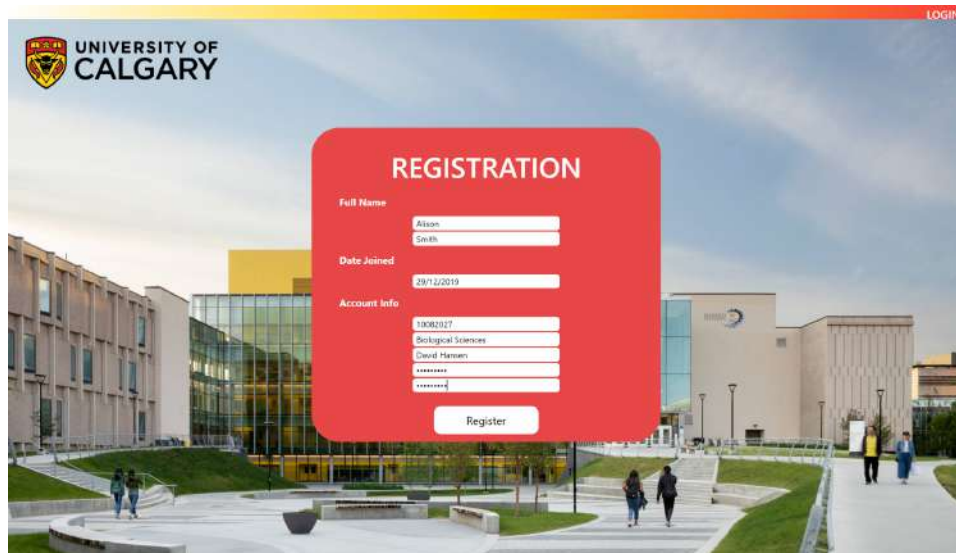


The screenshot shows the University of Calgary Admin Registration form. The form is a red overlay on a background image of a university campus. The form has a header with the University of Calgary logo and the word "UNIVERSITY OF CALGARY". The form is titled "REGISTRATION" and contains the following fields:

- Full Name:** First Name, Last Name
- Date Joined:** DD/MM/YYYY
- Account Info:** UCID, Department Name, Dept Head's Name, Password, Confirm Password

A "Register" button is located at the bottom of the form. A "LOGIN" link is visible in the top right corner of the page.

After pressing the red “Admin” button on the registration option page, an admin registration form will appear. Admin’s are required to fill out every text box and then press register. Each system admin is in charge of a unique department, thus the department name cannot already exist for another admin. Once they have been successfully registered, they will be taken to the admin dashboard page. Next time when they come back to our application, they will not have to register again, and can just simply login using their UCID and their chosen password.



The screenshot shows the University of Calgary Admin Registration form with example data filled in. The form is a red overlay on a background image of a university campus. The form has a header with the University of Calgary logo and the word "UNIVERSITY OF CALGARY". The form is titled "REGISTRATION" and contains the following fields:

- Full Name:** Alison Smith
- Date Joined:** 28/12/2019
- Account Info:** 10002027, Biological Sciences, David Hansen, [password], [password]

A "Register" button is located at the bottom of the form. A "LOGIN" link is visible in the top right corner of the page.

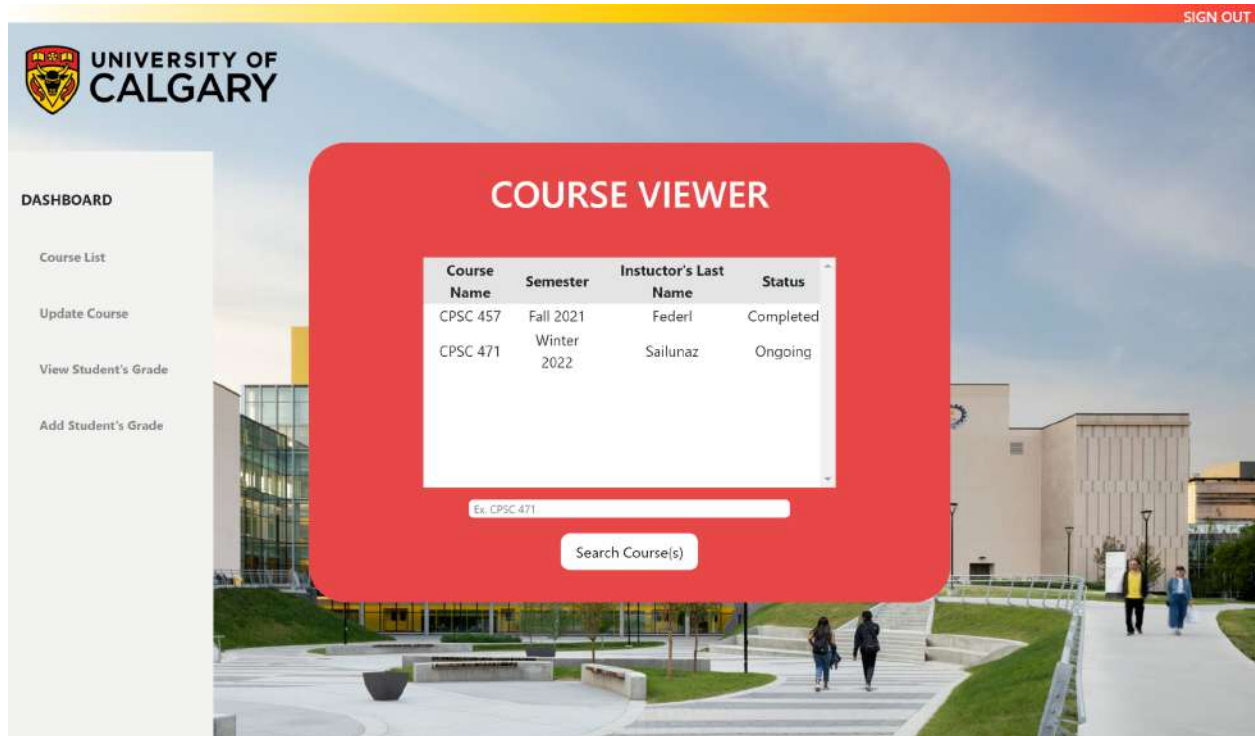
An example of an admin registering. This admin will be in charge of the biological sciences department.

Admin Dashboard Page



This is the admin dashboard page that every admin sees once they register or login. On the left hand side panel, they are given options of admin functionalities. Pressing on any of those will redirect them to the page for the specific function. For example, pressing on “Course List” will allow them to view all the courses that are in the database for the department the admin is in charge of. If the admin wishes to sign out while on this page, they can press the “Sign out” button in the upper right corner.

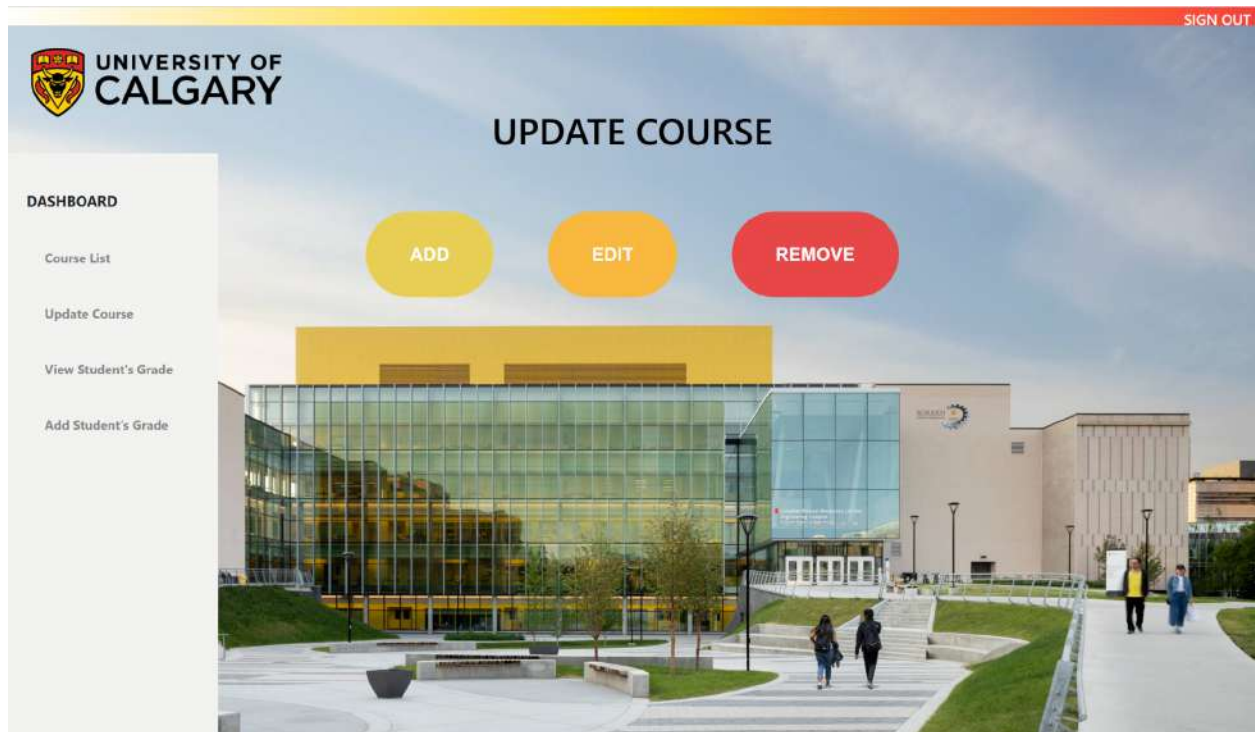
View Courses Page



When “Course List” is pressed from the left side panel, a list of all the courses stored in the database respective to the admin’s department appear. Here, the admin can search for a specific course to filter out from the total list. If the admin wishes to sign out while on this page, they can press the “Sign out” button in the upper right corner.

*Note: This feature was not properly implemented but can be implemented later as another functionality.

Update Course Page



The “Update Course” option from the left side panel brings the admin to this page. Here, they can choose whether they would like to add a new course, edit a single or multiple fields of an existing course or remove a course entirely from the database. If the admin wishes to sign out while on this page, they can press the “Sign out” button in the upper right corner.

Add Course Page

UNIVERSITY OF CALGARY

ADD COURSE

Name and Department

Ex: CPSC 471
Ex: Computer Science

Semester

Ex: Winter 2022

Description and Status

Description
Ongoing, Completed or Future

Instructor's Name

First Name
Last Name

Location

Building
Room Number

Day and Time

Ex: MWF
Ex: 12:00 - 14:30

Class Size and Capacity

Number of Enrolled Students
Max Capacity

Save

SIGN OUT

On this page, the admin can add a course that does not currently exist in the database. For this functionality, all the fields need to be filled in properly for the course to be saved into the database. If the admin wishes to sign out while on this page, they can press the “Sign out” button in the upper right corner.

Edit Course Page

UNIVERSITY OF CALGARY

DASHBOARD

- Course List
- Update Course
- View Student's Grade
- Add Student's Grade

EDIT COURSE

Choose a course to edit:

Ex. CPSC 471

Modify one or more attributes:

Name

Ex. CPSC 471

Semester

Ex. Winter 2022

Description

Description

Instructor's Name:

First Name Last Name

Location

Building Room Number

Day and Time

Ex. MWF Ex. 12:00 - 14:30

Current Size and Capacity

Number of Enrolled Students Max Capacity

Save

SIGN OUT

On this page, the admin can edit a course that already exists in the database. For this functionality, all the fields do not need to be filled in. Even if only one attribute is changed, this function will work. If the admin wishes to sign out while on this page, they can press the “Sign out” button in the upper right corner.

Remove Course Page

UNIVERSITY OF CALGARY

DASHBOARD

- Course List
- Update Course
- View Student's Grade
- Add Student's Grade

REMOVE COURSE

Choose the course you wish to remove:

Ex: CPSC 471

Ex: Fall 2021

Save

SIGN OUT

On this page, the admin can remove a course that exists in the database. For this functionality, all the fields need to be filled in. Since the same course can be offered in different semesters, the semester field is required alongside the course name. If the admin wishes to sign out while on this page, they can press the "Sign out" button in the upper right corner.

View Grade Page

UNIVERSITY OF CALGARY

STUDENT GRADES

Student ID	Course Name	Semester	Letter GPA
30090257	ENGG 200	Fall 2019	A+
30071766	ENGG 200	Spring 2020	B-
30045153	CPSC 471	Winter 2022	A+

Enter Student's UCID

Search Student

DASHBOARD

- Course List
- Update Course
- View Student's Grade
- Add Student's Grade

SIGN OUT

When “View Student’s Grade” is pressed from the left side panel, a list of all the department related course grades appear. Here, the admin can search for a specific student to filter out from the total list.

*Note: This feature was not properly implemented but can be implemented later as another functionality. If the admin wishes to sign out while on this page, they can press the “Sign out” button in the upper right corner.

Add Grade Page

UNIVERSITY OF CALGARY SIGN OUT

DASHBOARD

- Course List
- Update Course
- View Student's Grade
- Add Student's Grade

ADD GRADE

Student's UCID

Course Name

Course Semester

Percentage Grade (%)

Letter Grade

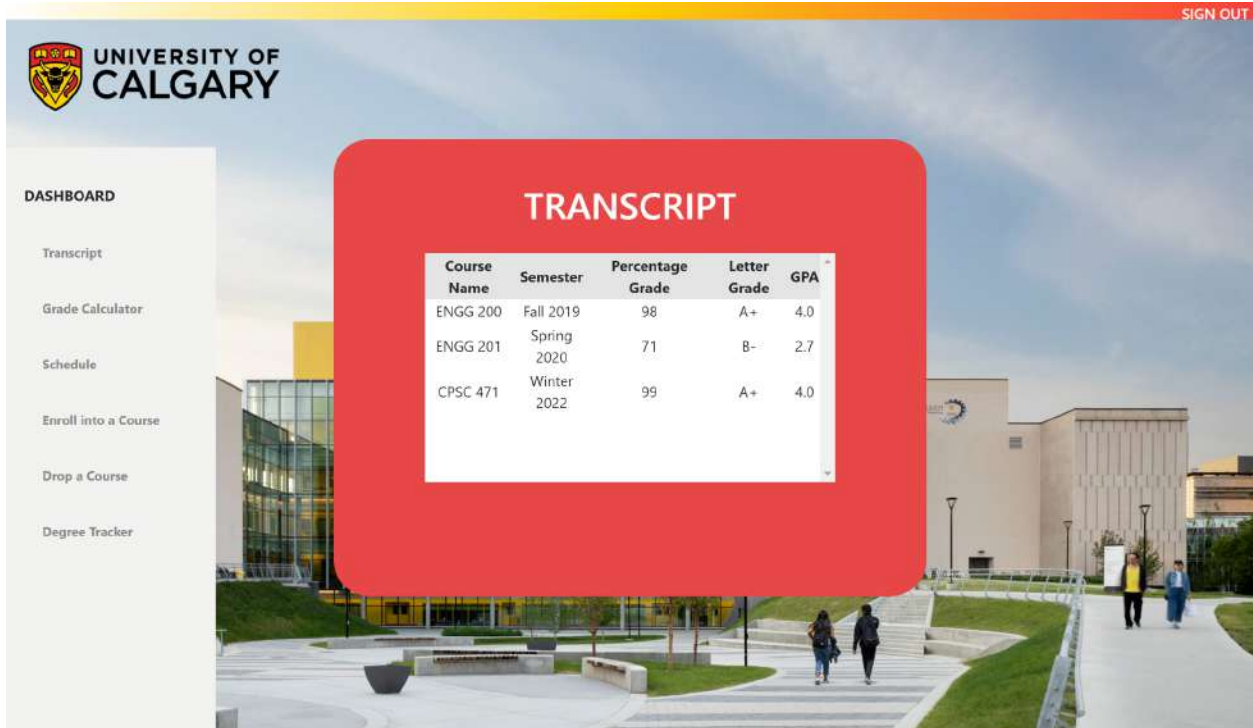
On this page, the admin can add a student's grade to a course. For this functionality, all the fields need to be filled in properly for the grade to be saved into the database. If the admin wishes to sign out while on this page, they can press the "Sign out" button in the upper right corner.

Student Dashboard Page



This is the student dashboard page that every student sees once they register or login. On the left hand side panel, they are given options of student functionalities. Pressing on any of those will redirect them to the page for the specific function. For example, pressing on “Drop Course” will allow the student to drop out of a course that they are currently enrolled in. If the student wishes to sign out while on this page, they can press the “Sign out” button in the upper right corner.

Transcript Page



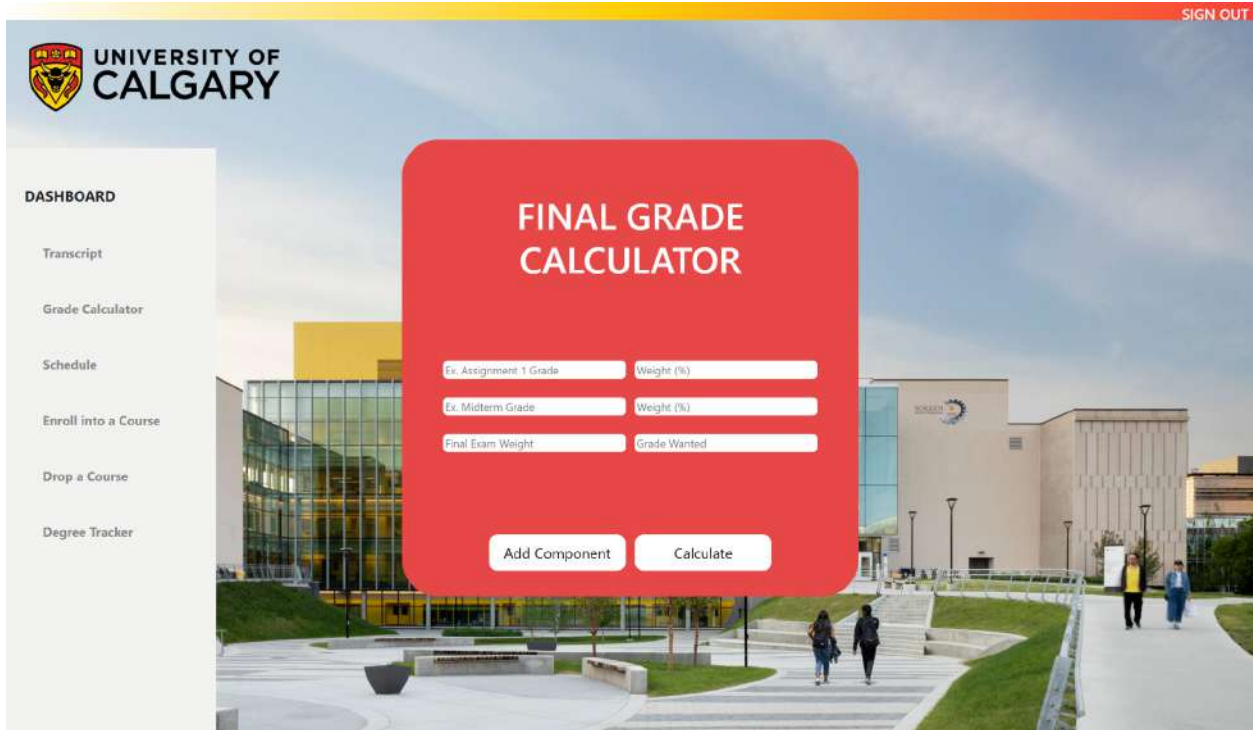
The screenshot shows the University of Calgary student dashboard. The background is a photograph of a modern university building with a glass facade and a paved walkway where several students are walking. In the top right corner, there is a red button labeled "SIGN OUT". On the left side, there is a vertical sidebar with the heading "DASHBOARD" and several menu items: "Transcript", "Grade Calculator", "Schedule", "Enroll into a Course", "Drop a Course", and "Degree Tracker". The "Transcript" menu item is highlighted. In the center of the screen, a large red rounded rectangle contains the word "TRANSCRIPT" in white capital letters. Below this, a white table displays the student's transcript data.

Course Name	Semester	Percentage Grade	Letter Grade	GPA
ENGG 200	Fall 2019	98	A+	4.0
ENGG 201	Spring 2020	71	B-	2.7
CPSC 471	Winter 2022	99	A+	4.0

When “Transcript” is pressed from the left side panel, a list of all the courses and corresponding grade appears for the student. Here, the student can see their completed course history. If the student wishes to sign out while on this page, they can press the “Sign out” button in the upper right corner.

*Note: This feature was not properly implemented but can be implemented later as another functionality.

Calculator Page



The image shows a web application for the University of Calgary. At the top left is the University of Calgary logo. At the top right is a "SIGN OUT" button. On the left is a sidebar menu with the following items: DASHBOARD, Transcript, Grade Calculator, Schedule, Enroll into a Course, Drop a Course, and Degree Tracker. The main content area features a large red rounded rectangle titled "FINAL GRADE CALCULATOR". Inside this rectangle are three input fields for "Ex. Assignment 1 Grade", "Ex. Midterm Grade", and "Final Exam Weight", each paired with a "Weight (%)" or "Grade Wanted" field. Below these fields are two buttons: "Add Component" and "Calculate". The background of the page is a photograph of a modern university building with a glass facade and a paved walkway with people.

UNIVERSITY OF CALGARY

FINAL GRADE CALCULATOR

Ex. Assignment 1 Grade Weight (%)

Ex. Midterm Grade Weight (%)

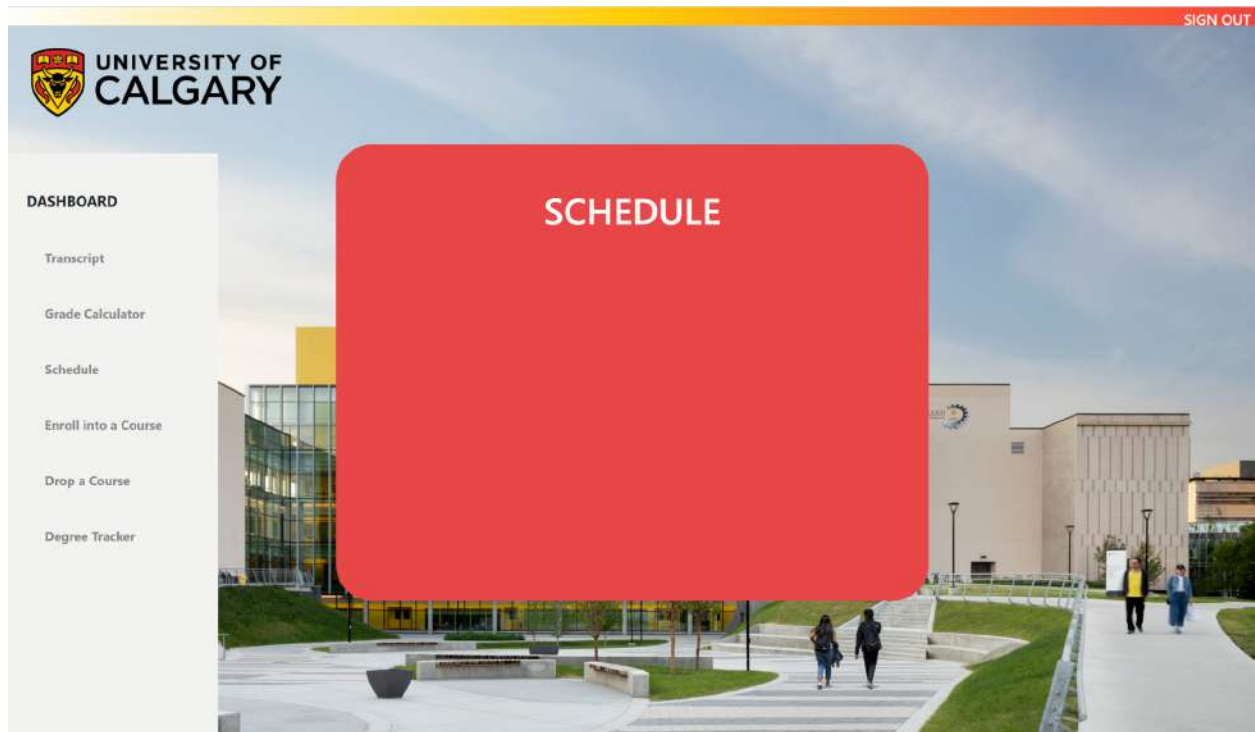
Final Exam Weight Grade Wanted

Add Component Calculate

If the student wishes to sign out while on this page, they can press the “Sign out” button in the upper right corner.

This page is supposed to take in a varied amount of values and their weights and calculate how much is needed on the final for the wanted grade.

Schedule Page



When “Schedule” is pressed from the left side panel, a weekly calender showing the student’s schedule was supposed to be shown. If the student wishes to sign out while on this page, they can press the “Sign out” button in the upper right corner.

*Note: Currently, this page does not work due to being unable to import the scheduling calender we had intended. In the future however, this functionality can be added to allow students to view all their current courses in a calendar format, with the correct time and date for each course.

Enroll into a Course Page

UNIVERSITY OF CALGARY SIGN OUT

DASHBOARD

- Transcript
- Grade Calculator
- Schedule
- Enroll into a Course
- Drop a Course
- Degree Tracker

ENROLL INTO COURSE

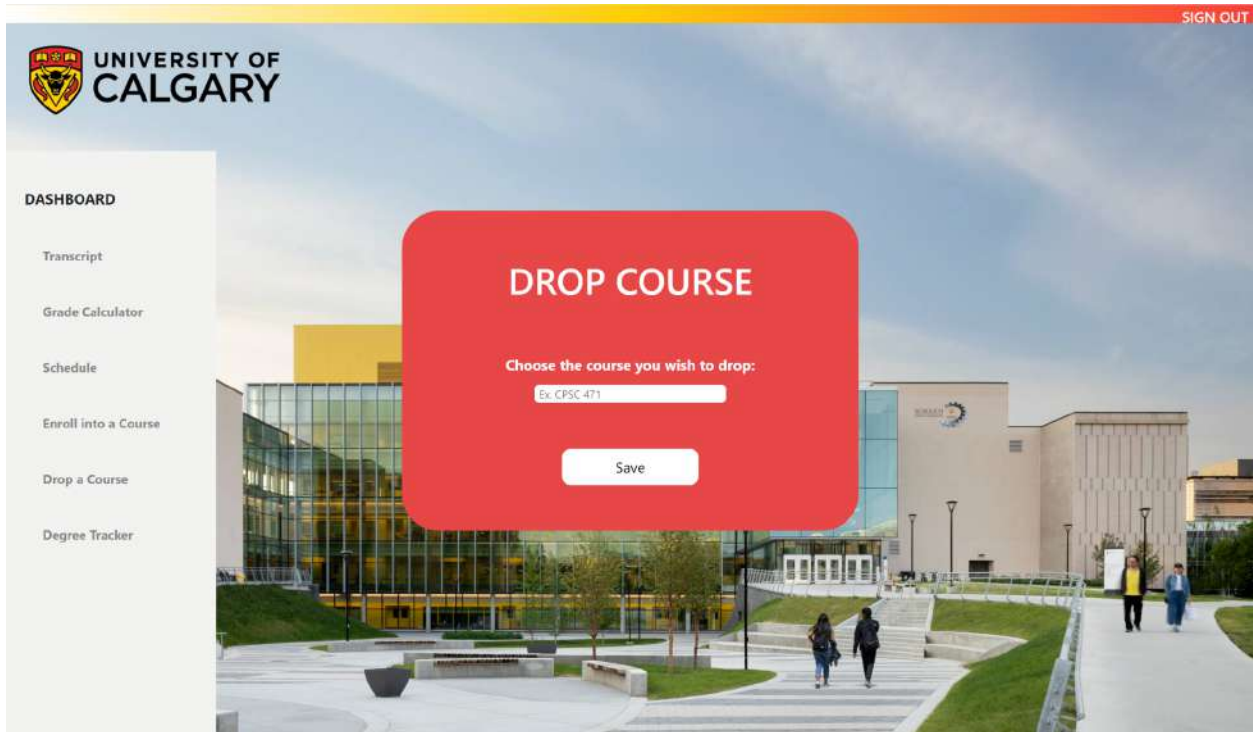
Choose the course you wish to enroll into:

Ex. CPSC 471

Save

On this page, the student can enroll into a course. For this functionality, the student enters which course they would like to enroll in and then it is stored in the database. The chosen course has to be already existing in the database. If the student wishes to sign out while on this page, they can press the “Sign out” button in the upper right corner.

Drop Course Page



UNIVERSITY OF CALGARY

DASHBOARD

- Transcript
- Grade Calculator
- Schedule
- Enroll into a Course
- Drop a Course
- Degree Tracker

DROP COURSE

Choose the course you wish to drop:

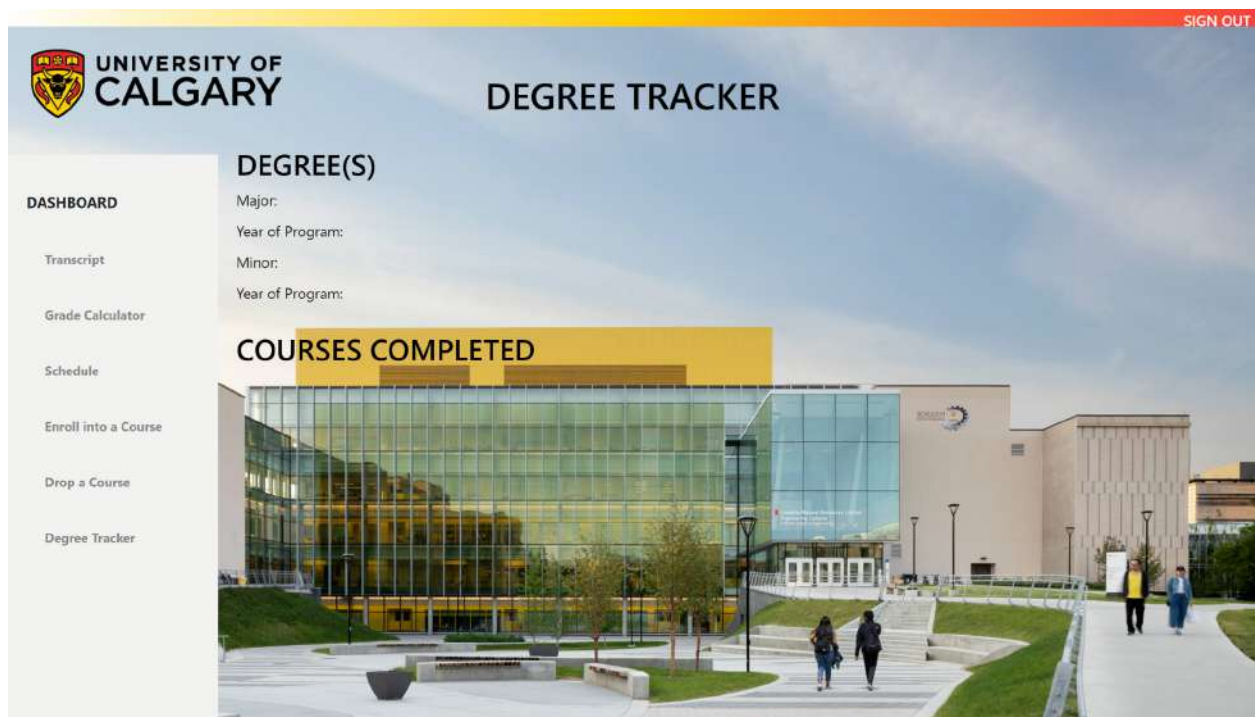
Ex: CPSC 471

Save

SIGN OUT

On this page, the student can drop out of a course that they are enrolled in. For this functionality, all the fields need to be filled in. Since the same course can be offered in different semesters, the semester field is required alongside the course name. If the student wishes to sign out while on this page, they can press the “Sign out” button in the upper right corner.

Degree Tracker Page



On this page, the student can view their program details and which courses they have completed for their degree. If the student wishes to sign out while on this page, they can press the “Sign out” button in the upper right corner.

*Note: This feature was not properly implemented but can be implemented later as another functionality.