STARFLEET ACADEMY REGISTRATION SYSTEM

System Manual

FOR

Web Based Registration System

SYSTEM DESIGN AND IMPLEMENTATION

CS 5510

FALL 2017

Professor Naresh Gupta

Benjamin Xerri	Philip Witkin	Kwame Mensah
Number: (631)278-0034 Email: bxerri@oldwestbury.edu	Number: (516) 647-1824 Email: pwitkin@oldwestbury.edu	Number: (917)605-0043 Email: kmensah2@oldwestbury.edu

Mission statement

This project was created to provide the necessary features of a course registration system for a university; allowing services related to the Office of the Registrar in a safe and secure manner. The system will be a web based application. Users of the system will be students, faculty, administration, and research departments. Each set of users will have their own unique set of features and some common features.

Table of Contents

Mission Statement……..……..1
Student Requirements…….……..……...3
Faculty Requirements……..……....5
Administration Requirements…….……..……...7
Research Requirements…….……..9
System Functions……..……10
Relationships……..……14
Constraints…….……..…….…….……....16
ERD Diagram……..……18
Golang……..……...19
Implementation / Use Cases / Code……20
Public Features……..……..……...……..20
System Login……..……..……...……....30
Admin……33
Researcher……75
Faculty……...80
Student……..……..86
Group Meeting……..……..……...101
Final Notes…..……..……..……102

Student

Students make up the majority of users in the system. A student must be able to login and use the system properly to be a part of the school. Therefore, the system must be user friendly to students of varying technical backgrounds. A student in the system will be able to perform certain functions at any time in the system, such as viewing their transcript and viewing previous semester schedules. Other functions will be limited within a certain time period, such as registering for courses.

Any student must be logged into the system to achieve certain goals. The system will be designed in a way that makes it easy for students achieve these goals. Upon logging into the system, the student will see a sub-navigation bar that will help the student understand the functions he is able to perform. (See figure 1B).

Student Requirements

- There are two types of students in our system, fulltime and part-time.
- Full-time students must register for no less than 3 courses and no more than 5 courses.
- Part-time students will only be allowed to register for 1 or 2 courses.
- The system will provide many features for students that are related to course registration.
- All students can search for courses to add to their schedule for the upcoming semester.
- A student can view grades and classes for many past semesters.

- A student's transcript will show a student the details of all the courses they have taken up until the current semester.
- Details of a transcript include, the courses they have taken for each semester and the grade received for each course.
- Students will be able to view any holds that they may have on the account.
- A course may have many prerequisite courses.
- A prerequisite may have many courses.
- A student has many advisors.

Students Forbidden

- The system will enforce that a student is registered the correct number of courses, based on their part-time or full-time status.
- The system will forbid students from adding or removing courses outside of the allotted course registration period.
- The system will prevent students from registering for any concurrent courses.
- If a student has a hold, then system will forbid the student from registering for any courses.
- Examples of holds that may prevent a student from registering for courses include; Financial, academic, health, or disciplinary.
- The system will forbid students from registering for any course if they lack the prerequisite courses.

Faculty

The faculty of the school can be thought of as those who are educating and advising the students. Because they are the ones who are educating the students, it is important that they are able to view any student's schedule and current transcript. This will allow for faculty advisors to appropriately advise their students when the time arises.

Similar to students, upon logging into the system, the faculty will see a customized sub navigation bar that will make it easier to accomplish their daily goals.

Faculty Requirements

- There are two types of faculty members in our system, part time and full time.
- Part-time faculty members may teach only 1 - 2 classes.
- Full-time professors may teach anywhere from 3-5 classes.
- Faculty will be allowed to see the courses they are teaching the current semester.
- Faculty is also allowed to view all the current students registered to their courses and view their transcripts.
- Faculty is the only role in the system that is allowed to give students their course grades within a limited time.
- A student may receive only one singular grade for the course.

Faculty Forbidden

- The system will forbid faculty members from teaching concurrent courses.

- Faculty is forbidden from altering grades outside the time allotted by administration.

Administration

Administration users have the least restrictions placed on what they can do. Many of the features of Administrators are granted to other users as well. They can perform all the actions which students can do on behalf of a student in the system, such as enrolling or dropping a student from a course, checking student schedules, viewing holds placed on a student, and viewing student grades and transcripts. Administrators can also perform actions otherwise granted only to faculty, such as viewing a course roster. Admins can also view the courses being taught by a faculty member.

Administrators also have some specialized privileges which no other user type has. They can alter the master schedule for the school, lift holds applied to any student,override student prerequisites, and change grades which have already been submitted. The Admin must also open up the system to allow faculty to submit grades.

In the same manner as students and faculty, upon logging into the system, the faculty will see a customized sub navigation bar that will group their large number of possible goals.

Administration Requirements

As stated above, Administrators share requirements with both students and faculty. Those requirements will not be restated as the requirements are the same.

- The requirements of students and faculty can be performed by the system as if the Administrator were any specific student or faculty member.

Administration will be allowed to do certain actions not shared by other users.

- They are allowed to add courses to the master schedule to make new courses available.
- The Administrator must supply all of the course information, including a description, department, location, time, teacher, and any prerequisites.
- Administrators can remove existing courses from the master schedule.
- Administrators can eliminate a hold placed on a student record.
- Administrators must open the system for accepting grades by the faculty.

Administration Forbidden

- The system will forbid administrators having the ability to change a grade already in the system.
- The system will forbid administration from creating of more than one course in the same place at the same time.

Researcher

Research Department Requirements

- The research department has access to and is able to gather all of the data produced by the system.
- Examples of this data include enrollment data, grade data and graduation rate.
- The researcher's end goal is to perform many kinds of analysis or visualization.
- The goal of this analysis is to make conclusions about patterns within the data of the system.
- The site serves as a source of data assembled from the site's database, with minimal to no processing or additional organization.
- The faculty of the research department can login to the system but has limited access to site features.
- The faculty can gather sets of data based on selected parameters.
- The user will be able to select from a few different data sources, such as course offerings, enrollment data, or grades.
- They must also select boundary conditions, such as a timeframe or department.
- This data produced by the faculty of the research department is delivered to many researchers as a file.
- The researchers can download many files analyze however they see fit.

System Functions

Entity - Attributes

User Attributes

- User_ID
- Email
- Password
- First Name
- Last Name
- Type

Student Attributes

- Student_ID
- Email
- Password
- First Name
- Last Name
- Type

Full-Time Student Attributes

- Student_ID
- Email
- Password
- First Name
- Last Name

Part-Time Student Attributes

- Student_ID
- Email
- Password
- First Name
- Last Name

Students Schedule History

- Student ID
- Enrollment ID
- Status

Faculty

- Faculty ID Number
- Department ID
- First Name
- Last Name
- Email
- Type

Full-Time Faculty Attributes

- Faculty ID Number
- Department ID
- First Name
- Last Name
- Email

Part-Time Faculty Attributes

- Faculty ID Number
- Department ID
- First Name
- Last Name
- Email

Admin Attributes

- ID Number
- First_Name
- Last_Name
- Email_Address

Researcher Attributes

- Researcher ID
- First Name
- Last Name

- Email

Courses

- Course ID
- Department ID
- Course Name
- Description
- Number of Credits for Course

Prerequisite Course

- Course Required By
- Course Requirement

Attendance

- Enrollment ID
- Student ID
- Date
- Present/Absent

Sections

- Course ID
- Faculty ID
- Section ID
- Capacity
- Time Slot ID
- Building
- Room Number

Enrollment

- Student ID
- Course ID
- Grade

Reports

- Report ID
- Date Created

- Description
- Location on Disk

Department

- Department ID
- Department Chair
- Department Name
- Building
- Department Phone Number
- Department Room Number

Major

- Major ID
- Department ID
- Major Name

Minor

- Major ID
- Department ID
- Minor Name

Holds

- Hold ID
- Hold Name

Timeslot

- Timeslot ID
- Day ID
- Period ID
- Semester ID

Day

- Day ID
- Meeting Day

Period

- Period ID

- Start Time
- End Time

Semester

- Semester ID
- Season
- Year

Relationships

Part Time Faculty Teaches 1-2 Sections

- Faculty ID
- Course ID

Full Time Faculty Teaches 2-4 Sections

- Faculty ID
- Course ID

A Student Has Many Advisors

- Faculty ID
- Student ID

An Advisor Advisor Has Many Students

- Faculty ID
- Student ID

A Student Has Many Major

- Student ID
- Major ID

A Major Has Many Students

- Student ID
- Major ID

A Student Has Many Minors

- Student ID

- Minor ID

A Minor Has Many Students

- Student ID
- Minor ID

A Full Time Student Enrolls in 3-5 Courses

- Student ID
- Course ID
- Grade Earned

A Part Time Student Enrolls in 1-2 Courses

- Student ID
- Course ID
- Grade Earned

A Course has many Students

- Student ID
- Course ID
- Grade Earned

A Course Has Many Sections

- Course ID
- Section ID

A Course Has Many Prerequisites

- Course Required
- Course Required By

A Prerequisite Has Many Courses

- Course Required
- Course Required By

A Student Has Many Holds

- Student ID
- Hold ID

A Hold Can Be Held by Many Students

- Student ID

- Hold ID

Constraints

Student
- Primary Key - Student Email
- Foreign Key- Major ID
- Foreign Key - Minor ID

Full-Time Student
- Primary Key - Student Email
- Foreign Key- Major ID
- Foreign Key - Minor ID

Part-Time Student
- Primary Key - Student Email
- Foreign Key- Major ID
- Foreign Key - Minor ID

Department
- Primary Key - Department ID

Major
- Primary Key- Major ID
- Foreign Key- Department ID

Minor
- Primary Key- Minor ID
- Foreign Key- Department ID

Section
- Primary Key - Section ID
- Foreign Key - Course ID

Course
- Primary Key - Course ID

- Foreign Key - Department ID

Enrollment

- Primary Key - Student ID
- Foreign Key - Course ID

Faculty

- Primary Key - Faculty Email
- Foreign Key - Department ID

Full-Time Faculty

- Primary Key - Faculty Email
- Foreign Key - Department ID

Part-Time Faculty

- Primary Key - Faculty Email
- Foreign Key - Department ID

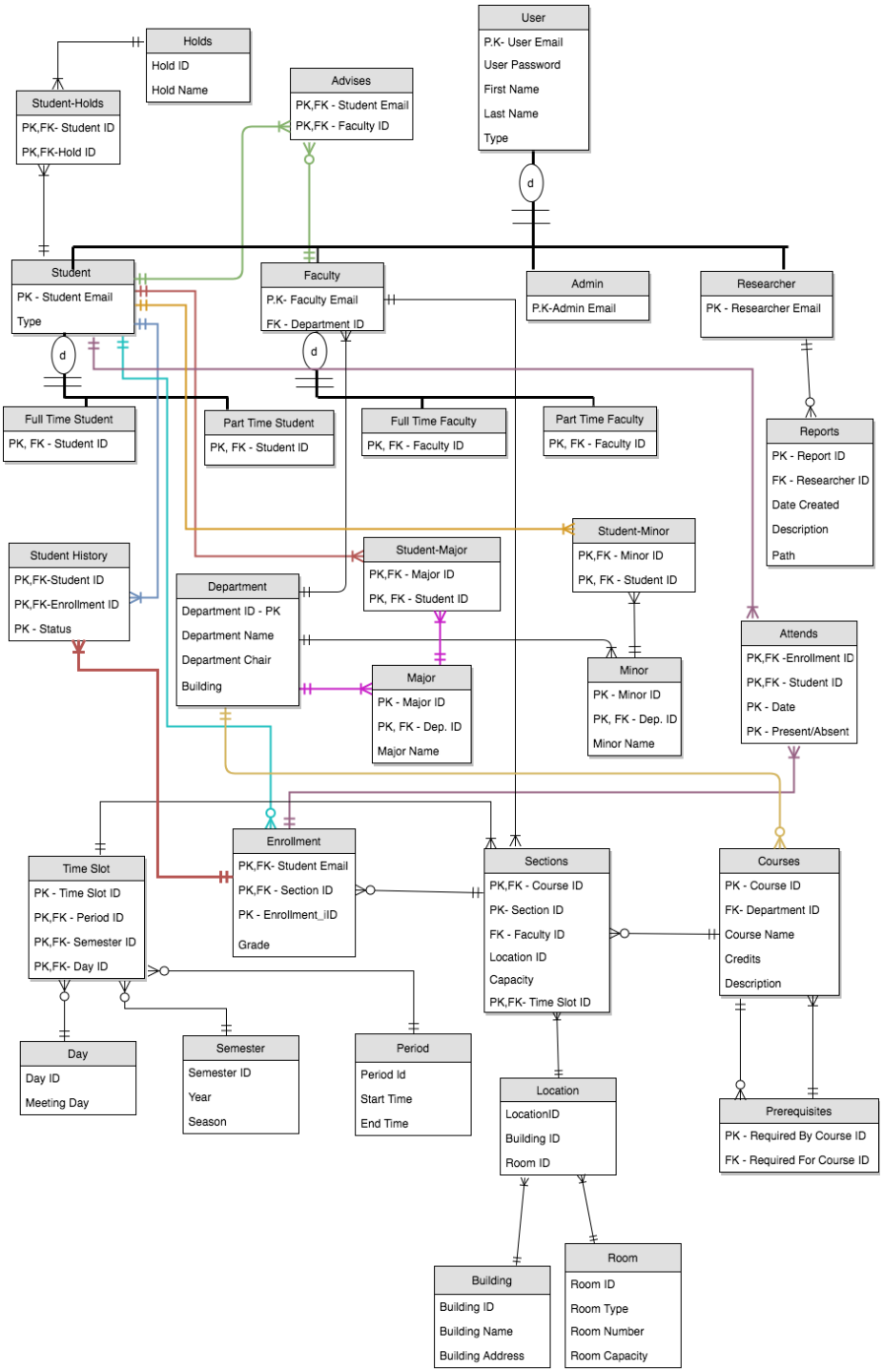
Admin

- Primary Key- Admin Email

Researcher

- Primary Key - Researcher Email

ERD - Diagram


Holds
Hold ID
Hold Name
Student-Holds
PK,FK- Student ID
PK,FK-Hold ID
Advises
PK,FK - Student Email
PK,FK - Faculty ID
User
P.K- User Email
User Password
First Name
Last Name
Type
d
Student
PK - Student Email
Type
Faculty
P.K- Faculty Email
FK - Department ID
Admin
P.K-Admin Email
Researcher
PK - Researcher Email
Full Time Student
PK, FK - Student ID
Part Time Student
PK, FK - Student ID
Full Time Faculty
PK, FK - Faculty ID
Part Time Faculty
PK, FK - Faculty ID
Reports
PK - Report ID
FK - Researcher ID
Date Created
Description
Path
Student History
PK,FK-Student ID
PK,FK-Enrollment ID
PK - Status
Department
Department ID - PK
Department Name
Department Chair
Building
Student-Major
PK,FK - Major ID
PK, FK - Student ID
Student-Minor
PK,FK - Minor ID
PK, FK - Student ID
Major
PK - Major ID
PK, FK - Dep. ID
Major Name
Minor
PK - Minor ID
PK, FK - Dep. ID
Minor Name
Attends
PK,FK -Enrollment ID
PK,FK - Student ID
PK - Date
PK - Present/Absent
Time Slot
PK - Time Slot ID
PK,FK - Period ID
PK,FK- Semester ID
PK,FK- Day ID
Enrollment
PK,FK- Student Email
PK,FK - Section ID
PK - Enrollment_ilD
Grade
Sections
PK,FK - Course ID
PK- Section ID
FK - Faculty ID
Location ID
Capacity
PK,FK- Time Slot ID
Courses
PK - Course ID
FK- Department ID
Course Name
Credits
Description
Day
Day ID
Meeting Day
Semester
Semester ID
Year
Season
Period
Period Id
Start Time
End Time
Location
LocationID
Building ID
Room ID
Prerequisites
PK - Required By Course ID
FK - Required For Course ID
Building
Building ID
Building Name
Building Address
Room
Room ID
Room Type
Room Number
Room Capacity

Templating With Golang

To simplify the HTML, Go has server side templating functionality built into it. This allows us to loop over data structures and use boolean logic in the front end HTML logic. Using this functionality allowed us to reduce the amount of repeated code. Examples can be seen throughout the front-end implementation in the system manual.

Some of the functions used were {{range }} and {{if}}.

Routing With Golang

All requests incoming into the server must be routed to a certain function. Otherwise, the server will not know how to respond to a certain user request. For example, when a request of /student/registered is called upon clicking on a link on the front end, the function 'ViewRegisteredCourses' will be called and processed as a "GET" request.

Examples of student routes

```
/* Student Routes*/
routes.Handle("/student",  checkSessionWrapper(displayStudent)).Methods("GET")
routes.HandleFunc("/student/schedule", ViewSchedule).Methods("GET")
routes.HandleFunc("/student/registered", ViewRegisteredCourses).Methods("GET")
routes.HandleFunc("/student/registered", DropRegisteredCourse).Methods("POST")
routes.HandleFunc("/student/holds", ViewHolds).Methods("GET")
routes.HandleFunc("/student/advisor", ViewAdvisor).Methods("GET")
routes.HandleFunc("/student/transcript", ViewTranscript).Methods("GET")
routes.HandleFunc("/student/search", AddCoursePage).Methods("GET")
//routes.HandleFunc("/student/register", StudentSearchCourseResults).Methods("GET")
routes.HandleFunc("/student/register", RegisterForSection).Methods("POST")
```

Public Features

Search Master Schedule

- **Based on the parameters passed into the front-end form, a unique query is built to find the course sections that match the search. If no parameters at all are passed in, then the user will see all of the courses available.**

Search Master Schedule Front - End Implementation

```
{{define "masterScheduleSearch"}}
{{template "head"}}

{{$isAd := .User.IsAdmin}}

{{$isStudent := .Student.IsStudent}}

{{if .User.IsAdmin}}
    <h1>Admin is Logged in.</h1>
    {{template "admin-navbar"}}
{{end}}

{{if .Student.IsStudent}}
    <h1>Student is Logged in.</h1>
    {{template "student-navbar"}}
{{end}}

<div class="contact-w3-agileits" id="contact" >
    <div class="container">
        <div class="col-md-12 contact-left-w3l" >
            <h3>Class Schedule Search</h3>
            <div class="visit">

                <div class="clearfix"></div>
            </div>


            <div class="col-md-8 contact-left-w3l">
                <form action="/course/" id="requestData" class="form-horizontal" method="GET">
                    <label class="form-group">Department</label>
                    <select name="department" class="form-control">
                        <option value="">Select A department</option>
                        {{ range .Departments }}

                            <option
                                    value="{{ .DepartmentID }}">
                                {{ .DepartmentName }}
                            </option>

                        {{ end }}
                    </select><br/>
```

```
            <br />

            <label class="form-group">Course Name</label>
            <input type="text" class="form-control" name="course-name" placeholder="" >
            <br />


            <label class="form-group">Day</label>
            <select name="day" class="form-control">
                <option value="">Select A Day</option>
                <option value="1">MW</option>
                <option value="2">TR</option>
                <option value="3">MWF</option>
                <option value="4">SA</option>
                <option value="5">SU</option>
            </select>

            <br />


            <label class="form-group">Instructor</label><br />
            <input type="text" class="form-control" name="instructor" placeholder="" >
            <br />


            <input type="submit" class="btn-success"  value="Search">
            <!--<form>-->
                <!---->
            <!--</form>-->
            <!--<input type="button" class="btn-success" value="Reset">-->
        </form>
    </div>

    <div class="col-md-8 contact-left-w3l">
        <h2>Search Results</h2>

        <h3>Department: {{ .Params.Department }}</h3>
        {{ if .Params.CourseName }}
            <h3>Course Name: {{ .Params.CourseName }}</h3>
        {{ end }}
        {{ if .Params.CourseNum }}
            <h3>Course Number: {{ .Params.CourseNum }}</h3>
```

```
<div class="col-md-8 contact-left-w3l">
    <h2>Search Results</h2>

    <h3>Department: {{ .Params.Department }}</h3>
    {{ if .Params.CourseName }}
        <h3>Course Name: {{ .Params.CourseName }}</h3>
    {{ end }}
    {{ if .Params.CourseNum }}
        <h3>Course Number: {{ .Params.CourseNum }}</h3>
    {{ end }}
    {{ if .Params.Professor }}
        <h3>Professor: {{ .Params.Professor }}</h3>
    {{ end }}


    <ul>
        {{ range .Results }}
        <li>
            <div>
                <h3>
                    {{ .CourseName }} Section {{ .CourseSectionNumber }}
                </h3>
                <h4>
                    Professor: {{ .FirstName }} {{ .LastName }}
                </h4>
                <h4>
                    Credits: {{ .CourseCredits }}
                </h4>
                <h4>Number Enrolled: {{.NumEnrolled}}</h4>
                <h4>
                    Section Capacity: {{ .Capacity }}
                </h4>
                <h4>Day: {{ .MeetingDay }} </h4>
                <h4>Time: {{ .Time }} </h4>

                <h4>Location: {{ .RoomNumber }}</h4>
                <h4>Building: {{ .BuildingName }}</h4>

                {{if $isAd}}
                <form action="/section/update" method="GET">
                    <input type="text" hidden name="section" value="{{.SectionID}}">
                    <input type="submit" value="Edit Section" class="btn btn-danger pull-right">
```

```
                        {{if $isStudent}}
                        <form action="/student/register" method="POST">
                            <input type="text" value="{{.SectionID}}" name="section" hidden>
                            <input type="submit" value="Register" class="btn btn-primary pull-right">
                        </form>
                        {{end}}

                        <hr>
                        <h4>Prerequisites:</h4>
                        <ul>
                            {{ range .Prerequisites }}
                                <li>
                                    <div class="col-md-8 contact-left-w3l course-styles">
                                        <p>{{ .CourseName }}</p>
                                    </div>
                                </li>
                            {{ end }}
                        </ul>
                    </div>
                    <br/>
                </li>
                <hr>
                <br/>



                {{  end }}
            </ul>

        </div>

    </div>
</div>
</div>

{{template "end"}}

{{end}}
```

Search Master Schedule Back - End Implementation

```
func searchMasterSchedule(w http.ResponseWriter, r *http.Request){

        println("Inside searchMasterSchedule")

        queryVals := r.URL.Query()

        departmentQuery,_ := queryVals["department"]
        courseNameQuery,_ := queryVals["course-name"]
        professorQuery := queryVals["instructor"]
        day := queryVals["day"]

        depID := departmentQuery[0]
        courseName := courseNameQuery[0]
        professor := professorQuery[0]
        dayID := day[0]

        whereMap := make(map[string]interface{})

        whereStuff := "WHERE "

        numQueries := 0

        if depID != "" {
                println("Department query present: " + depID)
```

```
if depID != "" {
        println("Department query present: " + depID)
        //depID, _ := strconv.ParseUint(departmentQuery[0], 10, 64)
        whereMap["department_id"] = depID
        whereStuff += "department_id = " + depID
        numQueries++
}
if courseName != "" {
        whereMap["course_name"] = courseName
        if numQueries == 0 {
                whereStuff += " course_name = '" + courseName + "'"

        }else {
                whereStuff += " AND course_name = '" + courseName + "'"

        }
        numQueries++
}

if dayID != ""{
        if numQueries == 0 {
                whereStuff += " day.day_id = " + dayID
        } else {
                whereStuff += " AND day.day_id = " + dayID
        }
        numQueries++
}

if professor != "" {
        prof := strings.Split(professor, " ")
        if numQueries == 0 {
                whereStuff += " first_name = '" + prof[0] + "'"
                whereStuff += " AND last_name = '" + prof[1] + "'"

        }else {
                whereStuff += " AND first_name = '" + prof[0] + "'"
                whereStuff += " AND last_name = '" + prof[1] + "'"
        }
        numQueries++

}
```

```
//registering for next semester
if numQueries == 0 {
        whereStuff += " semester.year = 2018 AND semester.season = 'Spring'"
}else {
        whereStuff += " AND semester.year = 2018 AND semester.season = 'Spring'"
}

type IsAdmin struct {
        IsAdmin bool
        User model.MainUser
}

        type CourseData struct {
                CourseName string
                CourseCredits int
                CourseDescription string
                DepartmentID uint
                SectionID uint
                CourseSectionNumber int
                CourseID uint
                FacultyID uint
                FirstName string
                LastName string
                TimeSlotID uint
                LocationID uint
                DayID uint
                MeetingDay string
                RoomID uint
                RoomNumber string
                RoomType string
                BuildingID uint
                BuildingName string
                Capacity uint
                Time string
                Prerequisites []model.Course
                User IsAdmin
                NumEnrolled int
        }

//coursesFound := []model.Course{}
//db.Where(model.Course{CourseName: courseName}).Or(model.Course{DepartmentID: uint(depID)}).Or(model.Course{CourseName: cou

queryRes := []CourseData{}
```

```
	//rows, err := db.Joins("JOIN course ON course.course_id = section.course_id").Where(whereMap).Rows()
	sql := `SELECT course.course_name, course.course_credits, course.course_description, course.department_id, section.section_id
	section.course_id, section.faculty_id, section.time_slot_id, section.location_id, section.course_section_number,
	main_user.first_name, main_user.last_name,
	day.meeting_day, day.day_id,
	building.building_name,time,
	room.room_number, room.room_type,capacity

	FROM section
	JOIN course ON course.course_id = section.course_id
	JOIN main_user ON main_user.user_id = section.faculty_id
	JOIN location ON section.location_id = location.location_id
	JOIN building ON building.building_id = location.building_id
	JOIN room ON room.room_id = location.room_id
	JOIN time_slot ON time_slot.time_slot_id = section.time_slot_id
	JOIN semester ON time_slot.semester_id = semester.semester_id
	JOIN day ON time_slot.day_id = day.day_id
	JOIN period ON period.period_id = time_slot.period_id `

	sql += whereStuff

	if strings.Contains(whereStuff,";&"){
		fmt.Println("escape")
	}
	fmt.Println("QUery to be run is", sql)
	db.Raw(sql).Scan(&queryRes)


		//if err == nil{
		//	//rows.Scan(&queryRes)
		//
		//} else {
		//	println(err.Error())
		//}
	/*
	for _, val := range queryRes{
		println(val.CourseName)
	}
	*/
	fmt.Println(queryRes)
	allDepartments := []model.Department{}
```

```
*/
fmt.Println(queryRes)
allDepartments := []model.Department{}

db.Find(&allDepartments)

coursesFoundPrereqs := make(map[string][]model.Course)

// get prereqs for every unique course in queryRes
for qIndex, val := range queryRes{
        if _, present := coursesFoundPrereqs[val.CourseName]; !present {
                course := model.Course{}
                db.First(&course, val.CourseID)
                prereqs := course.FindCoursePrerequisites(db)
                coursesFoundPrereqs[val.CourseName] = prereqs
                queryRes[qIndex].Prerequisites = prereqs
                println("Adding prereqs for: " + course.CourseName)
                for _,c := range val.Prerequisites{
                        println("-prereq::" + c.CourseName)
                }
        } else {
                queryRes[qIndex].Prerequisites = coursesFoundPrereqs[val.CourseName]
        }
}

//chosenDep := model.Department{}
//db.First(&chosenDep, depID)
searchParams := map[string]string{
        //"Department": chosenDep.DepartmentName,
        "Professor": professor,
        "CourseName": courseName,
        //"CourseNum": courseNum,
}

_,user := CheckLoginStatus(w,r)
```

```
//courseEnrollment := make(map[int]int)
for k, _ := range queryRes {
        count := 0
        db.Table("enrollment").Select("*").Where("section_id = ?", queryRes[k].SectionID).Count(&count)
        queryRes[k].NumEnrolled = count
}


admin := IsAdmin{}

if user.UserType == 3 {
        admin.IsAdmin = true
        admin.User = user
}else{
        admin.IsAdmin = false
}

type IsStudent struct {
        IsStudent bool
        User model.MainUser
}

student := IsStudent{}

if user.UserType == 1 {
        student.IsStudent = true
        student.User = user
}else {
        student.IsStudent = false
}


data :=  map[string]interface{}{
        "Results": queryRes,
        "Departments": allDepartments,
        "Params": searchParams,
        "User":admin,
        "Student": student,
}

err := global.Tpl.ExecuteTemplate(w, "masterScheduleSearch", data)
```

System Login and User Authentication

Back-End Implementation

- When a request is made for the login page, the system utilizes a session manager which checks for a cookie stored on the client computer of the user. The cookie is used to retrieve an instance of session data, which are all stored in memory on the server and used to track authentication, is then checked to see if there is a UserID set for the session.
- If one is found, this means the user is already logged in, and they are forwarded to the appropriate page for their role (student, teacher, admin, etc)
- If not, the form for the user to enter email and password is displayed.

```
func loginPage(w http.ResponseWriter, r *http.Request){
    // use session information to determine the user, and if they are already logged in
    logged, u := CheckLoginStatus(w, r)
    // a user already logged in will be sent to the page of their respective role
    if logged{
        checkUserType(u,w,r)
    }else {
        global.Tpl.ExecuteTemplate(w, "login", nil)
    }
}
```

```
func CheckLoginStatus(w http.ResponseWriter, r *http.Request) (bool,model.MainUser){
    sess := globalSessions.SessionStart(w,r)
    sess_uid := sess.Get("UserID")
    u := model.MainUser{}
    if sess_uid == nil {
        //http.Redirect(w,r, "/", http.StatusForbidden)
        //Tpl.ExecuteTemplate(w,"index", "You can't access this page")
        return false, u
    } else {
        uID := sess_uid
        db.First(&u, uID)
        fmt.Println("Logged in User, ", uID)
        //Tpl.ExecuteTemplate(w, "user", nil)
        return true, u
    }
}
```

- When the user submits a login form, the email and password data is sent via a POST http request to the server.
- The email is used to search the database for a unique user (since email is stored as unique in the User table). If no user is found, the login fails, and the form is displayed again with an error message.
- If a user with the form's email is found, the password of that database record is compared with form's password after it has been hashed (using the same hashing algorithm used to encrypt the password when it was inserted)

```
func loginUser(w http.ResponseWriter, r *http.Request) {
    sess := globalSessions.SessionStart(w, r)
    r.ParseForm()

    if r.Method == "GET" {
        t, _ := template.ParseFiles("login.gtpl")
        w.Header().Set("Content-Type", "text/html")
        t.Execute(w, sess.Get("username"))
    } else {
        //is a POST
        formEmail := r.FormValue("email")
        formPassword := r.FormValue("password")
        // Try to find user in DB
        user := model.MainUser{}
        db.Where(&model.MainUser{UserEmail: formEmail}).First(&user)

        if user.UserEmail != "" {
            dbPassword := user.UserPassword

            if user.CheckPasswordMatch(formPassword) {
                fmt.Println("User found in DB with email:", formEmail, " and password: ", dbPassword)
                sess.Set("username", r.Form["username"])
                sess.Set("UserID", user.UserID)
                //http.Redirect(w,r,"/user/" + strconv.Itoa(int(user.UserID)), http.StatusFound)

                //Tpl.ExecuteTemplate(w,"user",user)
                checkUserType(user, w, r)
            } else {
                global.Tpl.ExecuteTemplate(w,"login","Error, username or password does not match.")
            }

        } else {
            fmt.Println()
            global.Tpl.ExecuteTemplate(w,"login","User not found")
        }
    }
}
```

Check User Type

- **This function is called immediately after the user is logged in. It is used to route the user to their current role. For example, if a user logs in and has a user type of 1, the /student route will be requested.**

```
func checkUserType(user model.MainUser, w http.ResponseWriter, r *http.Request){

    switch user.UserType {

    case 1:
        fmt.Println("You're a student")
        fmt.Println("User data is", user.FirstName)
        http.Redirect(w,r,"/student", http.StatusFound)

        // The data is lost after redirect because it's a new request,
        // now I need to get the student data and render the template, which is a different request
        //since http is stateless, you l;ose the data structure after the first request.
    case 2:
        fmt.Println("Youre a faculty")
        http.Redirect(w,r,"/faculty", http.StatusFound)


    case 3:
        fmt.Println("Youre an admin")
        http.Redirect(w,r,"/admin", http.StatusFound)
        //Tpl.ExecuteTemplate(w,"admin", "administrative user!")

    case 4:
        fmt.Println("Youre a researcher")
        http.Redirect(w,r,"/researcher", http.StatusFound)
        //Tpl.ExecuteTemplate(w,"admin", "administrative user!")

    default:
        fmt.Println("Not sure your type")
        http.Redirect(w,r,"/", http.StatusFound)
        global.Tpl.ExecuteTemplate(w,"index",nil)
        //return user,user.UserType
    }

}
```

Login - Front End Implementation

- **Requires the user to enter their email and password.**

```
{{define "login"}}
{{template "head"}}

    <body style="...">
    <section class="container" style="...">
        <div class="col-xs-3">
        </div>
        <div class="col-lg-5" style="...">
            <div class="row">
                <div class="col-xs-12" style="...">
                    <div class="col-xs-3">
                    </div>
                    <div class="col-xs-9">
                        <img style="..." src="../public/pics/5.png" />
                    </div>
                </div>

                <form action="/login" id="requestData" class="form-horizontal" method="POST">
                    <div class="col-xs-12" style="...">
                        <input type="email" class="form-control" name="email" placeholder="Email" style="..." />
                        <input type="password" class="form-control" name="password" placeholder="Password" style="..." />
                        <button type="submit" class="btn-warning" style="...">LOGIN</button>
                    </div>
                </form>

                <div class="login-error">{{.}}</div>
            </div>
        </div>
    </section>
    </body>

{{template "end"}}
{{end}}
```

Administration Functionality Implementation

Admin/Faculty View Student Transcript Front-End Form

View Transcript

Use Case Name	View Transcript
Description	Displays a transcript for a selected student.
Participating Actors	Admin,Faculty
Input	Student ID
Output	A Student Transcript
Flow	1) The user enters the user email in the input field. 2) The user submits the form. 3) The user will be able to view their transcript.
Entry Condition	The user is logged into the system.
Exit Condition	The user will be able to view their up to date transcript.

- **If a faculty or admin submits this form, they will first be prompted to enter in the student email in the form field.**

Front End Implementation

```
{{define "viewStudentTranscriptAdmin"}}
{{template "head"}}
<div class="container">

    <h1>Welcome {{ .FirstName}}</h1>

    {{template "admin-navbar"}}

    <h1>View Student Transcript</h1>

    <div class="contact-w3-agileits" id="contact" >
        <div class="col-md-12 contact-left-w3l" >
            <h3>Student Action</h3>
            <div class="visit">

                <div class="clearfix"></div>
            </div>

            <div class="col-md-8 contact-left-w3l">

                <form action="/admin/transcript/{student}" id="requestData" class="form-horizontal" method="GET">

                    <label class="form-group">Email</label>
                    <input type="email" class="form-control" name="email"><br />

                    <input type="submit" value="submit" id="submit" class="btn btn-default pull-right">

                </form>

            </div>

        </div>
    </div>
    {{if .NoUser}}
    <h2>No results for that user</h2>
    {{else}}
    <h2>Try searching for a user</h2>
    {{end}}
    <br/>

</div>
{{template "end"}}
{{end}}
```

View Student Transcript Details Front End

- **If the user that was searched has a transcript, it will be displayed by this page.**

```
{{define "adminViewStudentTranscriptDetails"}}
{{template "head"}}

<div class="container">



    {{template "admin-navbar"}}

    {{if .}}
    <h3>Transcript For {{.User.FirstName}} {{.User.LastName}}</h3>
    <h3>Current GPA is: {{.GPA.StudentGPA}}</h3>
    {{range .Transcript}}
    <div class="course_details">
        <p>Course Name: {{.CourseName}}</p>
        <p>Course Credits: {{.CourseCredits}}</p>
        <p>Course Status: {{.Status}}</p>
        <p>Semester Year: {{.Year}}</p>
        <p>Semester Season: {{.Season}}</p>
        <p>Grade Received: {{.Grade}}</p>
        <hr>
    </div>
    <hr>
    {{end}}
    {{else}}
    <h1>Error loading transcript details</h1>
    {{end}}



</div>

{{template "end"}}
{{end}}
```

View Student Transcript Details Back-End

- **The View Student Transcript function takes in the user email from front-end form and uses it to query the database. This function also takes the grades of each completed course for the student and calculates their current GPA.**

```
func viewStudentTranscript(w http.ResponseWriter, r *http.Request){
        user := model.MainUser{}

        email := r.FormValue("email")
        count := 0

        if err != nil {
                err.Error()
        }


        //first check if the email is not blank
        if email != ""{
                db.Where(&model.MainUser{UserEmail: email}).Find(&user).Count(&count)
        }
        if count > 0 {
                fmt.Println("You have a user", user.FirstName)
        } else {

                m := map[string]interface{}{
                        "NoUser": "Nobody Home",
                }

                fmt.Println("Showing no users found error")
                global.Tpl.ExecuteTemplate(w,"viewStudentTranscriptAdmin", m)
                return
        }
        type Transcript struct {
                StudentID uint
                Grade string
                Status string
                Year int
                Season string
                CourseName string
                CourseCredits int
        }

        st := []Transcript{}
```

```
db.Raw(`
 SELECT enrollment.student_id,grade,status,year,season,course_name,course_credits
 FROM student_history
 JOIN enrollment ON student_history.enrollment_id = enrollment.enrollment_id
 JOIN section ON enrollment.section_id = section.section_id
 JOIN course on course.course_id = section.course_id
 JOIN time_slot ON time_slot.time_slot_id = section.time_slot_id
 JOIN semester ON time_slot.semester_id = semester.semester_id
 WHERE enrollment.student_id = ?`,user.UserID).Scan(&st)
fmt.Println(st)

totalGrade := 0
var total float64 = 0


for i := 0 ; i < len(st); i++{
        g := st[i].Grade

        if strings.Compare(g,"-") != 0 { // if the grade is not 0
                totalGrade++

                if strings.Compare(g,"A") == 0{
                        total += 4

                }else if strings.Compare(g,"B+") == 0{
                        total += 3.3

                }else if strings.Compare(g,"B") == 0{
                        total += 3.0

                }else if strings.Compare(g,"B-") == 0{
                        total += 2.7

                }else if strings.Compare(g,"C") == 0{
                        total += 2.0

                }else if strings.Compare(g,"C-") == 0{
                        total += 1.7
```

```
                        }else if strings.Compare(g,"D+") == 0{
                                total += 1.3

                        }else if strings.Compare(g,"D") == 0{
                                total += 1.0

                        }else if strings.Compare(g,"F") == 0{
                                total += 0.0
                        }

                }

        }

        gpa := total/float64(totalGrade)

        fmt.Println("GPA is ", Round(gpa, .5,2))
        RoundedGpa := Round(gpa, .5, 2)

        type GPA struct {
                StudentGPA float64
        }

        studentGpa := GPA{StudentGPA:RoundedGpa}

        m := map[string]interface{}{
                "User":user,
                "Transcript":st,
                "GPA":studentGpa,
        }



        errTemp := global.Tpl.ExecuteTemplate(w, "adminViewStudentTranscriptDetails", m)
        if errTemp != nil {
                fmt.Println(errTemp.Error())
        }

}
```

View Student Schedule Admin/Faculty

View Student Schedule Front-End Form

- **If a faculty or admin submits this form, they will first be prompted to enter in the student email in the form field.**

View Schedule

Use Case Name	View Schedule
Description	Displays a list of all the students courses for a selected student.
Participating Actors	Admin, Faculty
Input	Semester
Output	A list of Courses
Flow	1) The user enters the student email. 2) The user submits the form. 3) A list of students courses for the selected system are displayed
Entry Condition	The user is logged into the system.
Exit Condition	The user will be able to view the a list of courses they are taking for the current semester.

- **The form requires the administrator to enter in the students email for the transcript they wish to view.**
- **If the email is in the system as a student, the the Admin will be redirected to view the students transcript, otherwise they will get a message saying that no user is available.**

```
{{define "viewStudentScheduleAdmin"}}
{{template "head"}}
<div class="container">

        <h1>Welcome {{ .FirstName}}</h1>
    {{template "admin-navbar"}}

    <h1>View Student Schedule</h1>

    <div class="contact-w3-agileits" id="contact" >
            <div class="col-md-12 contact-left-w3l" >
                <h3>Student Action</h3>
                <div class="visit">

                    <div class="clearfix"></div>
                </div>

                <div class="col-md-8 contact-left-w3l">

                    <form action="/admin/student/{student}" id="requestData" class="form-horizontal" method="GET">

                        <label class="form-group">Email</label>
                        <input type="email" class="form-control" name="email"><br />


                        <input type="submit" value="submit" id="submit" class="btn btn-default pull-right">

                    </form>
                </div>

            </div>
        </div>
    <br>
        {{if .NoUser}}
            <h2>No results for that user</h2>
        {{else}}
            <h2>Try searching for a user</h2>
        {{end}}
        <br/>
</div>
{{template "end"}}
{{end}}
```

View Student Schedule Details Front- End

- The view of the student schedule details

```
{{define "adminViewStudentScheduleDetails"}}
{{template "head"}}

<div class="container">


    {{template "admin-navbar"}}

    {{if .}}
        {{range .}}
            <div class="course_details">
                <p>Course Name: {{.CourseName}}</p>
                <p>Course Credits: {{.CourseCredits}}</p>
                <p>Room Number: {{.RoomNumber}}</p>
                <p>Building Name: {{.BuildingName}}</p>
                <p>Time: {{.Time}}</p>
                <p>Meeting Days: {{.MeetingDay}}</p>
                <p>Professor: {{.FirstName}} - {{.LastName}}</p>
            </div>
            <hr>
        {{end}}
        {{else}}
            <h1>Error loading course details</h1>
        {{end}}



</div>

{{template "end"}}
{{end}}
```

View Student Schedule Backend Implementation

- **The View Student Transcript function takes in the user email from front-end form and uses it to query the database and passes the results to the front-end.**

```
func ViewStudentSchedule(w http.ResponseWriter, r *http.Request){

        user := model.MainUser{}

        email := r.FormValue("email")

        count := 0

        if email != ""{
                db.Where(&model.MainUser{UserEmail: email}).Find(&user).Count(&count)
        }
        //nu := NoUser{"Nobody found"}
        if count > 0 {
                fmt.Println("You have a user", user.FirstName)
        } else {

                m := map[string]interface{}{
                        "NoUser": "Nobody Home",
                }

                fmt.Println("Showing no users found error")
                global.Tpl.ExecuteTemplate(w,"viewStudentScheduleAdmin", m)
                return
        }

        type StudentSchedule struct {
                CourseName string
                CourseCredits string
                RoomNumber string
                BuildingName string
                Time string
                MeetingDay string
                FirstName string
                LastName string
        }
```

```
        ss := []StudentSchedule{}
        db.Raw(`SELECT student_history.student_id,course_name,course_credits,building_name,
        room_number,meeting_day, first_name, last_name, time,student_history.status
        FROM student_history
        JOIN enrollment ON student_history.enrollment_id = enrollment.enrollment_id
        JOIN section ON enrollment.section_id = section.section_id
        JOIN course ON course.course_id = section.course_id
        JOIN time_slot ON time_slot.time_slot_id = section.time_slot_id
        JOIN semester ON time_slot.semester_id = semester.semester_id
        JOIN period ON time_slot.period_id = period.period_id
        JOIN day ON time_slot.day_id = day.day_id
        JOIN location ON section.location_id = location.location_id
        JOIN building ON location.building_id = building.building_id
        JOIN room ON location.room_id = room.room_id
        JOIN faculty ON section.faculty_id = faculty.faculty_id
        JOIN main_user ON faculty.faculty_id = main_user.user_id
        WHERE enrollment.student_id = ? AND student_history.status = 'In progress'`,user.UserID).Scan(&ss)
        fmt.Println(ss)

        global.Tpl.ExecuteTemplate(w, "adminViewStudentScheduleDetails", ss)

}
```

View Student Holds

View Student Holds Front-End Form

- The form requires the administrator to enter in the students email for the transcript they wish to view.
- If the email is in the system as a student, the the Admin will be redirected to view the students transcript, otherwise they will get a message saying that no user is available.

```
{{define "viewStudentHoldsAdmin"}}
{{template "head"}}
<div class="container">

    <h1>Welcome {{.FirstName}}</h1>

    {{template "admin-navbar"}}



    <form action="/admin/holds/{user}" id="requestData" class="form-horizontal" method="GET">

        <div class="form-group">
            <label class="control-label col-sm-1" for="email">Student Email</label>
            <div class="col-sm-10">
                <input type="email" class="form-control" placeholder="Email" required id="email" name="email"
            </div>
        </div>

        <input type="submit" value="submit" id="submit" class="btn btn-default pull-right">

    </form>


</div>


<script>

</script>

{{template "end"}}
{{end}}
```

View/Delete Student Hold Front-End Details

Remove Student Hold

Use Case Name	Remove Student Hold
Description	Removes a hold for a student
Participating Actors	Admin
Input	StudentID, Semester, Hold
Output	Boolean-Successful
Flow	1) The user selects “View Student Holds” 2) The user searches for the user by user email. 3) A List of that user’s holds will appear if they have any. 4) The user selects the button next to the hold that says remove. 5) A response is given to indicate if the hold was successfully lifted
Entry Condition	The user is logged into the system.
Exit Condition	The user has lifted a hold placed on a student transcript.

- **The View Student Transcript function takes in the user email from front-end form and uses it to query the database and passes the results to the front-end. From this page, the admin also has an option to remove the hold.**

```
{{define "adminStudentHold"}}
{{template "head"}}

<div class="container">

    {{template "admin-navbar"}}

    {{if .User.FirstName}}
    <h1>Holds for {{.User.FirstName}} {{.User.LastName}}</h1>
    {{$user := .User.UserID}}
        <hr>
    {{if .Holds}}
        {{range .Holds}}
            <h2>Hold name: {{.HoldName}}</h2>
    <form action="/admin/holds/{{$user}}/{{.HoldID}}" method="POST" id="{{.HoldID}}">
        <input type="submit" value="Remove Hold"
               class="btn btn-default pull-right">
    </form>
            <hr>
        {{end}}

        {{else}}
            <h1>No holds for this user</h1>
        {{end}}
    {{end}}



</div>

{{template "end"}}
{{end}}
```

Delete Student Hold Back-End Implementation

- **The Delete student hold takes in the user ID and the hold ID and deletes the record from the student_holds table.**

```
func AdminDeleteHold(w http.ResponseWriter, r *http.Request){
        vars := mux.Vars(r)
        holdId := vars["id"]
        user := vars["user"]

        holdInt , err := strconv.Atoi(holdId)
        if err != nil {
                fmt.Println(err.Error())
        }
        userInt , err := strconv.Atoi(user)
        if err != nil {
                fmt.Println(err.Error())
        }

                fmt.Println("StudentID =", userInt, "HoldID =", holdInt, "User =", user)
        studentHold := model.StudentHolds{}

        db.Raw("SELECT * FROM student_holds WHERE student_id = ? AND hold_id = ?", userInt,holdInt).Scan(&st
        fmt.Println("Hold found", studentHold)
        db.Delete(&studentHold)
        global.Tpl.ExecuteTemplate(w, "adminSuccess", "Hold removed.")


}
```

Admin Add Course Front-End Form

- **The form requires the administrator to enter the course information to be added to the database.**

```
{{define "addCourseAdmin"}}
{{template "head"}}

{{template "admin-navbar"}}

<div class="contact-w3-agileits" id="contact" >
    <div class="container">
        <div class="col-md-12 contact-left-w3l" >
            <h3>Add Course</h3>
            <div class="visit">

                <div class="clearfix"></div>
            </div>

            <form action="/admin/course/{course}" id="requestData" class="form-horizontal" method="POST">


            <div class="col-md-8 contact-left-w3l">


                <label class="form-group">Course Name</label>
                <input type="text" class="form-control" name="name" placeholder="" required><p id="demo"></p

                <label class="form-group">Course Credits</label>
                <input type="number" class="form-control" name="credits" placeholder="" required id="major"

                <label class="form-group">Course Description</label>
                <textarea type="text"  class="form-control" name="description" placeholder="" required id="s

                <label class="form-group">Department Name</label>

                <select name="department" id="department-select" required class="form-control">
                    {{range .}}
                        <option value="{{.DepartmentID}}">{{.DepartmentName}}</option>
                    {{end}}
                </select>
                <br />

                <input type="submit" value="submit" id="submit" class="btn btn-default pull-right">


            </div>
```

Admin Add Course Back-End

- **Creates the course in the database based on the form values. After the course is created, the user will be required to enter prerequisites.**

```
func AdminAddCourse(w http.ResponseWriter, r *http.Request){
        isLogged, user := CheckLoginStatus(w,r)
        if !isLogged || user.UserType != 3{
                http.Redirect(w, r, "/", http.StatusForbidden)
        }

        courseName := r.FormValue("name")
        courseCredits := r.FormValue("credits")
        courseDescription := r.FormValue("description")
        courseDepartment := r.FormValue("department")


        intCredits,err := strconv.Atoi(courseCredits)
        if err != nil {
                fmt.Println(err.Error())
        }

        intDepartment, err := strconv.Atoi(courseDepartment)
        if err != nil {
                fmt.Println(err.Error())
        }

        course := model.Course{CourseName:courseName, CourseCredits:intCredits,
        CourseDescription:courseDescription, DepartmentID:uint(intDepartment)}

        fmt.Println("Course info is", course)
        errors := db.Create(&course)
        if errors.Error != nil {
                fmt.Println(errors.Error)
                return
        }

        courses := []model.Course{}
        db.Table("course").Select("course_name, course_id").Scan(&courses)


        m := map[string]interface{}{
                "Courses": courses,
                "CurrentCourse":course,
        }

        global.Tpl.ExecuteTemplate(w, "coursePreReq", m)
```

Add Course Prerequisites Front-End

- **Allows the user to add the prerequisites to the course by populating a table full of the courses selected as prerequisites. All of the courses in the database are loaded into the form and drop down options. When a value is selected, it will be added to a table full of courses which will be entered as prerequisites.**

```
{{define "coursePreReq"}}
{{template "head"}}

{{template "admin-navbar"}}

<div class="contact-w3-agileits" id="contact" >
    <div class="container">
        <div class="col-md-12 contact-left-w3l" >
            <h3>Add Course</h3>
            <div class="visit">

                <div class="clearfix"></div>
            </div>
            <!--
            <form action="/admin/course/prereq/{course}" id="requestData" class="form-horizontal" method="P
            -->

                <div class="col-md-8 contact-left-w3l" id="pre-req-form">


                    <label class="form-group">Course Name</label>
                    <input type="text" class="form-control" name="course-name" readonly value="{{.CurrentCou

                    <input type="text" class="form-control" id="course" name="course-id" value="{{.CurrentCo

                    <label class="form-group">Course Pre-Reqs</label>
                    <select name="course" id="courses" required class="form-control">
                        {{range .Courses}}
                            <option value="{{.CourseID}}" >{{.CourseName}}</option>
                        {{end}}
                    </select>

                    <input type="button" id="select-course" value="Add Course" class="btn btn-default pull-r
                    <input type="button" id="reset-courses" value="Reset" class="btn btn-default pull-right"


                    <table class="table table-striped">
                        <thead>
                            <tr>
                                <td>Course ID</td>
                                <td>Course Name</td>
                            </tr>
```

Add Course Prerequisites Back-End

- **The prerequisites are taken from the html table by a javascript functions and sends an ajax request of a JSON object. The code is processed on the backend inserted into the database.**

```
func AddCoursePreRequisit(w http.ResponseWriter, r *http.Request){
        fmt.Println("inside admin add course pre-req")

        jsonVal := r.FormValue("prereqs")
        courseID := r.FormValue("course")

        courseIDint,_ := strconv.Atoi(courseID)

        fmt.Println("Course prereqs are",jsonVal)
        fmt.Println("Course id is",courseID)

        courses := []Course{}

        bytes := []byte(jsonVal)
        err := json.Unmarshal(bytes,&courses)
        if err != nil{
                fmt.Print(err.Error())
        }

        fmt.Println("JSON data is",courses)


        for _, course := range courses {
                courseRequirementIDint, _ := strconv.Atoi(course.Id)
                prereq := model.Prerequisite{CourseRequiredBy:uint(courseIDint), CourseRequirement:uint(cou
                db.Create(&prereq)
                fmt.Println("Course added", prereq)
        }

        global.Tpl.ExecuteTemplate(w, "adminSuccess", nil)

}
```

Add Course Prerequisites Back-End

- **JavaScript function that gets all the rows in the table and constructs a JSON object to submit the prerequisites to the server.**

```
function getCourses(){
    var table = document.getElementById('course-results');
    var arrObjects = [];
    var courseIDs = [];

    for (var r = 0; r < table.rows.length; r++) {
        var course = new Object();
        course.id = table.rows[r].cells[0].textContent;
        course.name = table.rows[r].cells[1].textContent;
        arrObjects.push(course); //push each course object to an array
        console.log("Course info is" + course.id+ course.name);
        courseIDs.push(course.id)
    }

    var jsonString = JSON.stringify(arrObjects); //stringify those objects
    console.log(jsonString);
    console.log("Course Preq ID's are", courseIDs)
    var courseID = document.getElementById('course').value; //value of current course


    $.ajax({
        //url:"/admin/course/"+ courseID +"/"+courseIDs,
        type:"POST",
        url:"/admin/courses/prereq",

         data: {
             prereqs: jsonString,
             course: courseID
         },

        success:function () {
            console.log("Ajax sent")
            $('#pre-req-form').hide();
            $('#success-message').append("Course was added correctly");

        }
    });
}
```

Admin Search Course Back-End

- **The search course function takes the course that was clicked in the front end and displays the course information. The request is submitted by an Ajax request shown below.**

```
func AdminSearchCourse(w http.ResponseWriter, r *http.Request){

        fmt.Println("Inside admin search course")
        vars := mux.Vars(r)
        id := vars["course"]
        idInt, _ := strconv.Atoi(id)
        fmt.Println("the id is", idInt)
        course := model.Course{}
        //db.Table("course").Select("*").Where("course_id")
        db.Where(model.Course{CourseID:uint(idInt)}).Find(&course)

        prereqs := course.FindCoursePrerequisites(db)

        fmt.Println("Course is", course)
        fmt.Println("Pre-Reqs are", prereqs)


        courseDetail := model.Course{}
        db.Raw(`SELECT course_id, course_name,course_credits,course_description
        FROM course
        WHERE course.course_id =?`,id).Scan(&courseDetail)
        fmt.Println(courseDetail)



        m := map[string]interface{}{
                "Course": courseDetail,
                "PreReqs": prereqs,
        }

        err := global.Tpl.ExecuteTemplate(w, "adminViewCourseSection", m)
        if err != nil {
                fmt.Println(err.Error())
        }
        //return
}
```

Admin View Course AJAX Request

- **On the click of a row in the front end, this request will be sent to the server and update the view.**

```
$('#course-results').on('click', function(event) {
        var id = $(event.target).closest("tr").attr('id');
        //var id = this.id;
        console.log("Found id " + id);

    $.ajax({
        type:"GET",
        url:"/admin/course/search/"+id,
        success:function(){
            $(location).attr('href','/admin/course/search/'+id);
        }
    });

});
```

Admin Update Course

- **Displays the course information and allows the user to update the text fields and submit the new course information to the database.**

```
func UpdateCourse(w http.ResponseWriter, r *http.Request){
        fmt.Println("insude update course")

         name := r.FormValue("new-course-name")
         description := r.FormValue("new-course-description")
         credits := r.FormValue("new-course-credits")
         course := r.FormValue("course")

        oldCourse := r.FormValue("old-course-name")
        oldDescription := r.FormValue("old-course-description")
        oldCredits := r.FormValue("old-course-credits")

        fmt.Println("Course name", name)
        fmt.Println("Course desc", description)
        fmt.Println("Course cred", credits)
        fmt.Println("Course id", course)

        fmt.Println("Old Course desc", oldDescription)
        fmt.Println("Old Course cred", oldCredits)
        fmt.Println("Old Course name", oldCourse)


        if strings.Compare(name,"") == 0{
                name = oldCourse
        }

        if strings.Compare(description,"") == 0{
                description = oldDescription
        }

        if strings.Compare(credits,"") == 0 {
                credits = oldCredits
        }


        fmt.Println("Course is", course)

        type QueryRes struct {
                QueryRes string
        }

        qr := QueryRes{}
```

Populate Rooms For Buildings

- **Populates the rooms in the input fields of Admin Add Section, Update Section. The implementation is done by selecting all the information about the rooms for the current building, converting the query results into a json objects, and writing that value to the response body. This allows the JavaScript on the client side to fill in the form information with data from the server.**

```
func GetRoomsForBuilding(w http.ResponseWriter, r *http.Request){
        vars := mux.Vars(r)
        buildingId := vars["id"]
        rooms := []model.Room{}

        //if they select all then change the query

        buildingInt, _ := strconv.Atoi(buildingId)
        db.Raw("select room_id,room_type,room_number,room_capacity from location natural join " +
                        "room natural join building where building_id = ?",buildingInt).Scan(&rooms)


        //encode the rooms to a slice of bytes in json form
        data , err := json.Marshal(rooms)
        if err != nil {
                fmt.Println(err.Error())
        }

        //write those bytes to the response
        w.Write(data)
        fmt.Println(data)
}
```

Populate Rooms For Buildings AJAX Request

- **When a building is selected, this sends an ajax request when the input field changes, allowing the rooms to dynamically update based on the building selected.**

```
$('#building-select').on('change',function(){
    $.ajax({
        url:"/admin/section/room/"+this.value,
        type:"GET",
        success:function(data){
            var jsonData = JSON.parse(data);
            var results = $('#room-results');
            //empty results
            results.empty();

            jsonData.forEach(function(value){
                var option = $("<option value="+value.RoomID +"> Number: " +value.RoomNumber + "-- Capacity:"

                // "---- <b>Type: </b>"+ value.RoomType +
                // option.add('value',value.RoomID);
                results.append(option);
            });
            console.log(jsonData)
        }
    })
});
```

Admin Add Section

- **Allows the admin to add a section based on the input fields. A new time-slot is created to represent the section. There are certain database that must be checked before adding a section. The room can't already be occupied and the Professor teaching must not be concurrently teaching any clases.**

```
func AdminAddSection(w http.ResponseWriter, r *http.Request){

        //TODO things to check for
        //TODO Teacher can't teach concurrently
        //TODO room can't be occupied

        sectionNum := r.FormValue("section-number")
        courseSubject := r.FormValue("course-subject")
        courseName := r.FormValue("course")
        faculty := r.FormValue("faculty-name")
        time := r.FormValue("time")
        buildingNum := r.FormValue("building")
        roomNum := r.FormValue("room")
        semester := r.FormValue("semester")
        day := r.FormValue("day")
        capacity := r.FormValue("capacity")

        fmt.Println("Capacity", capacity)


        capacityInt,_ := strconv.Atoi(capacity)


        fmt.Println("Section Num:",sectionNum)
        fmt.Println("Course Name:",courseName)
        fmt.Println("Course Subject:",courseSubject)
        fmt.Println("Faculty",faculty)
        fmt.Println("Time",time)
        fmt.Println("Building",buildingNum)
        fmt.Println("Room Num:",roomNum)
        fmt.Println("Semester:",semester)
        fmt.Println("Day",day)


        semesterInt, _ := strconv.Atoi(semester)
        dayInt , _ := strconv.Atoi(day)
        timeInt , _ := strconv.Atoi(time)
```

```
timeSlot := model.TimeSlot{PeriodID:uint(timeInt),SemesterID:uint(semesterInt),DayID:uint(dayInt)}

buildingInt , _ := strconv.Atoi(buildingNum)
roomInt, _ := strconv.Atoi(roomNum)
facultyInt, _ := strconv.Atoi(faculty)

fac := model.Faculty{}
//db.Where(model.Faculty{FacultyID:uint(facultyInt)}).Scan(&fac)
db.Table("faculty").Select("*").Where("faculty_id = ?",facultyInt).Scan(&fac)

db.Create(&timeSlot)

location := model.Location{}
db.Where(model.Location{BuildingID:uint(buildingInt),RoomID:uint(roomInt)}).First(&location)

timeSlotID := timeSlot.TimeSlotID

sectionInt, _ := strconv.Atoi(sectionNum)
courseInt, _ := strconv.Atoi(courseName)
//facultyID, _ := strconv.Atoi(faculty)

newCourseSection := model.Section{CourseSectionNumber:sectionInt,CourseID:uint(courseInt), Faculty

//TODO: Complete, 1st series of test passed
type RoomCheck struct{
        Section_id int
        Location_id int
        Building_id int
        Building_name string
        Room_id int
        Room_number string
}
rc := []RoomCheck{}
```

```
	db.Raw(`
		SELECT section.section_id, location.location_id,season,year, building.building_id,room.ro
		FROM section
		JOIN location ON section.location_id = location.location_id
		JOIN building ON building.building_id = location.building_id
		JOIN room ON room.room_id = location.room_id
		JOIN time_slot ON time_slot.time_slot_id = section.time_slot_id
		JOIN day ON time_slot.day_id = day.day_id
		JOIN semester ON time_slot.semester_id = semester.semester_id
		JOIN period ON period.period_id = time_slot.period_id
		WHERE location.room_id = ? AND building.building_id = ? AND period.period_id = ?
		AND day.day_id = ? AND season = ? AND year = ?`,location.RoomID, location.BuildingID,time

	if len(rc) > 0 {
		global.Tpl.ExecuteTemplate(w, "adminSuccess", "Cant add section, because the room is alread
		fmt.Println("Cant add room, because is already ocupied at this time")
		return
	}


	type ProfessorCheck struct {
		CourseName string
		Period_id int
		Time string
		Day_id int
		Meeting_day string
		Year string
		Season string
	}

	cc := []ProfessorCheck{}
```

```
	db.Raw(`
		SELECT course_name,meeting_day,time,year,season,period.period_id,day.day_id
		FROM section
		JOIN course ON course.course_id = section.course_id
		JOIN time_slot ON time_slot.time_slot_id = section.time_slot_id
		JOIN semester ON time_slot.semester_id = semester.semester_id
		JOIN period ON time_slot.period_id = period.period_id
		JOIN day ON time_slot.day_id = day.day_id
		JOIN location ON section.location_id = location.location_id
		JOIN building ON location.building_id = building.building_id
		JOIN room ON location.room_id = room.room_id
		WHERE section.faculty_id = ? AND semester.year = ? AND semester.season = ?
		AND period.period_id = ? AND day.day_id= ?;
	`,facultyInt,2018,"Spring",time,day).Scan(&cc)
	fmt.Println(cc)


	fmt.Println("QUery parameters are time,day,faculty ", time, day, facultyInt)

	if len(cc) > 0{
		global.Tpl.ExecuteTemplate(w, "adminSuccess", "Cant add section,teacher is already teaching
		fmt.Println("Cant add section,teacher is already teachinga  course at this time slot exit f
		return
	}

	pd := model.Period{}

	//db.First(&pd,time)
	db.Table("period").Select("*").Where("period_id = ?", time).Scan(&pd)

	fmt.Println("Course section", newCourseSection)
	db.Create(&newCourseSection)
	global.Tpl.ExecuteTemplate(w, "adminSuccess", "Section successfully added")
}
```

Change Semester Status

- **Allows the Administrator to view the current semester status and update the semester to registration or grading periods.**

```
func changeSemesterStatus(w http.ResponseWriter, r *http.Request) {
        status := r.FormValue("status")
        semester := r.FormValue("semester")

        //sem := model.Semester{}

        type err struct {
                errormsg string
        }

        e := err{}

        //db.Model(&sem).Select("semester").Updates(map[string]interface{}{"semester_status":status}).W


        db.Raw(`
                UPDATE SEMESTER
                SET semester_status = ?
                WHERE semester_id = ?;
        `, status, semester).Scan(&e)

        global.Tpl.ExecuteTemplate(w, "adminSuccess", "Semester Status Changed Successfully.")

}
```

Admin Update Section

- **Allows the administrator to update the sections room, in the case that the section requires a need for a greater capacity. The admin can update the room or the section capacity.**

```
func AdminUpdateSection(w http.ResponseWriter, r *http.Request){
        newBuilding := r.FormValue("new-building")
        currentBuilding := r.FormValue("old-building")
        newRoom := r.FormValue("new-room")
        currentRoom := r.FormValue("current-room")
        section := r.FormValue("section-info")
        timeID := r.FormValue("time-id")
        day := r.FormValue("day-id")


        fmt.Println("New building", newBuilding)
        fmt.Println("Current BUilding", currentBuilding)
        fmt.Println("New Room", newRoom)
        fmt.Println("Current Room", currentRoom)
        fmt.Println("Section", section)
        fmt.Println("Time", timeID)
        fmt.Println("Day", day)



        currentLocation := model.Location{}
        db.Select("*").Table("location").Where("room_id = ?  AND building_id = ? ",currentRoom,currentBuil

        newLocation := model.Location{}
        db.Select("*").Table("location").Where("room_id = ?  AND building_id = ? ",newRoom,newBuilding).Sca

        fmt.Println("new Location ", newLocation)
        type Err struct {
                error string
        }

        e := Err{}

        type RoomCheck struct{
                Section_id int
                Location_id int
                Building_id int
                Building_name string
                Room_id int
                Room_number string
```

```
db.Raw(`
        SELECT section.section_id, location.location_id,season,year, building.building_id,room.roo
        FROM section
        JOIN location ON section.location_id = location.location_id
        JOIN building ON building.building_id = location.building_id
        JOIN room ON room.room_id = location.room_id
        JOIN time_slot ON time_slot.time_slot_id = section.time_slot_id
        JOIN day ON time_slot.day_id = day.day_id
        JOIN semester ON time_slot.semester_id = semester.semester_id
        JOIN period ON period.period_id = time_slot.period_id
        WHERE location.room_id = ? AND building.building_id = ? AND period.period_id = ?
        AND day.day_id = ? AND season = ? AND year = ?`,newLocation.RoomID, newLocation.BuildingID

if len(rc) > 0 {
        global.Tpl.ExecuteTemplate(w, "adminSuccess", "Can't update to this room because the room i
        fmt.Println("Cant add room, because is already ocupied at this time, please try a different
        return
}

db.Raw(`
        UPDATE section SET location_id = ? WHERE section_id = ?
`,newLocation.LocationID, section).Scan(&e)

fmt.Println(e)

global.Tpl.ExecuteTemplate(w, "adminSuccess", "Room updated successfully")
```

Admin Add User

Back-End implementation

- First check the database for an existing user with the provided email
- If no entry exists, we create a new user with the given email and password
- Based on the type of user, additional entries are made by calling additional functions - if they are a student, a new student is created, and a new full/part time student is also created; if they are a faculty, a new faculty member is created, and so on for each user type

```
func createUser (w http.ResponseWriter, r *http.Request) {
    r.ParseForm()
    formEmail := r.FormValue("email")

    userDB := model.MainUser{}
    userDB.UserEmail = formEmail

    count := 1
    db.Model(&model.MainUser{}).Where("user_email = ?", formEmail).Count(&count)
    if count == 0 {
        userDB.FirstName = r.FormValue("first-name")
        userDB.LastName = r.FormValue("last-name")
        userDB.UserPassword = r.FormValue("password")
        userType, _ := strconv.Atoi(r.FormValue("user-type"))
        userDB.UserType = userType
        valid, err := userDB.ValidateData()
        if valid {
            db.Create(&userDB)

            switch userDB.UserType {

            case 1:
                fmt.Println("You're a student")
                student := model.Student{}
                m := map[string]interface{}{
                    "User":    userDB,
                    "student": student,
                }
                res := global.Tpl.ExecuteTemplate(w, "viewNewUserAdmin", m)
                if res != nil{
                    println("newUserForm: ", res.Error())
                }
                //return
```

```
        case 2:
            fmt.Println("Youre a faculty")
            faculty := model.Faculty{}
            m := map[string]interface{}{
                "User":    userDB,
                "faculty": faculty,
            }
            global.Tpl.ExecuteTemplate(w, "viewNewUserAdmin", m)

        case 3:
            fmt.Println("Youre an admin")
            admin := model.Admin{AdminID: userDB.UserID}
            db.Create(&admin)
            http.Redirect(w, r, "/admin", http.StatusFound)
            displayAdmin(w, r)

        case 4:
            fmt.Println("Youre a researcher")
            researcher := model.Researcher{ResearcherID: userDB.UserID}
            db.Create(&researcher)
            //global.Tpl.ExecuteTemplate(w, "admin-new-user-generic", userDB)
            http.Redirect(w, r, "/admin", http.StatusFound)
            displayAdmin(w, r)

        default:
            fmt.Println("Not sure your type")
            global.Tpl.ExecuteTemplate(w, "viewNewUserAdmin", userDB)
        }

    } else {
        // validation failed
```

```
        // validation failed
        m := map[string]interface{}{
            "error": err,
        }
        global.Tpl.ExecuteTemplate(w, "viewNewUserAdmin", m)
    }


} else {
    // add to the err - email already taken
    m := map[string]interface{}{
        "error": "Email Already Taken",
    }
    global.Tpl.ExecuteTemplate(w, "viewNewUserAdmin", m)
    //return
}
}
```

```go
func createStudent(w http.ResponseWriter, r *http.Request) {
    r.ParseForm()
    formEmail := r.FormValue("email")
    credits,_ := strconv.Atoi(r.FormValue("credits"))
    studentType, _ := strconv.Atoi(r.FormValue("student-type"))
    mu := model.MainUser{}
    db.Where(&model.MainUser{UserEmail: formEmail}).First(&mu)
    stu := model.Student{StudentID:mu.UserID, StudentType:studentType}
    db.Create(&stu)

    if studentType == 1{
        stuFT := model.FullTimeStudent{FullTimeStudentID: stu.StudentID, NumCredits:credits}
        db.Create(&stuFT)
    } else {
        stuPT := model.PartTimeStudent{PartTimeStudentID: stu.StudentID, NumCredits:credits}
        db.Create(&stuPT)
    }
    http.Redirect(w,r,"/admin", http.StatusFound)
    displayAdmin(w,r)
}
```

```go
func createFaculty(w http.ResponseWriter, r *http.Request) {
    r.ParseForm()
    formEmail := r.FormValue("email")
    mu := model.MainUser{}
    db.Where(&model.MainUser{UserEmail: formEmail}).First(&mu)
    facultyType, _ := strconv.Atoi(r.FormValue("faculty-type"))
    println("Faculty type num: ", facultyType)
    department, _ := strconv.ParseUint(r.FormValue("department"), 10, 64)
    faculty := model.Faculty{FacultyID: mu.UserID, FacultyType: facultyType, DepartmentID: uint(department)}
    db.Create(&faculty)
    if (facultyType == 1) {
        facultyFT := model.FullTimeFaculty{FullTimeFacultyID: faculty.FacultyID}
        db.Create(&facultyFT)
    } else if (facultyType == 2) {
        facultyPT := model.PartTimeStudent{PartTimeStudentID: faculty.FacultyID}
        db.Create(&facultyPT)
    }
    http.Redirect(w, r, "/admin", http.StatusFound)
    displayAdmin(w, r)

}
```

Create User Front-End

```
{{define "adminUserDetail"}}
            <div class="col-md-8 contact-left-w3l">

                <form action="/admin/user" id="requestData" class="form-horizontal" method="POST">

                    <label class="form-group">Email</label>
                    <input type="email" class="form-control" name="email" placeholder="" required=""><br />

                    <label class="form-group">Password</label>
                    <input type="password" class="form-control" name="password" placeholder="" required=""><br />

                    <label class="form-group">First Name</label>
                    <input type="text" class="form-control" name="first-name" placeholder=""><br />

                    <label class="form-group">Last Name</label>
                    <input type="text" class="form-control" name="last-name" placeholder=""><br />

                    <label class="form-group">User Type</label>
                    <!--<input type="text" class="form-control" name="major" placeholder=""><br />-->
                    <select name="user-type" required="true">
                        <option value="1">Student</option>
                        <option value="2">Faculty</option>
                        <option value="3">Admin</option>
                        <option value="4">Researcher</option>
                    </select>

                    <div class="dropdown-divider"></div>

                    <input type="submit" value="submit" id="submit" class="btn btn-default pull-right">

                </form>
            </div>
{{end}}
```

```
{{define "viewNewUserAdmin"}}
{{template "head"}}

<div class="container">
    <h1>Welcome {{.FirstName}}</h1>
    {{template "admin-navbar"}}
    <h1>Create New User</h1>
    <div class="contact-w3-agileits" id="contact" >
        <div class="col-md-12 contact-left-w3l" >
            <h3>User Information</h3>
            <div class="visit">
                <div class="clearfix"></div>
            </div>
            <!--universal user info here-->
            {{ if .error }}
                {{ .error }}
            {{ end }}

            {{ if not .User }}
                {{ template "adminUserDetail" }}
            {{ else }}
            <!--put detail for user types here..-->
                {{ if .student }}
                    {{ template "adminStudentDetail" . }}
                {{ else if .faculty }}
                    {{ template "adminFacultyDetail" . }}
                {{ end }}
            {{ end }}
        </div>
    </div>
    <br/>
    <div class="clearfix"></div>
</div>

{{template "end"}}
{{end}}
```

```
{{ define "adminStudentDetail" }}
<div class="col-md-8 contact-left-w3l">

    <form action="/admin/user/student" id="requestData" class="form-horizontal" method="POST">

        <label class="form-group">Email</label>
        <input  contenteditable="false" type="email" class="form-control" name="email"
                placeholder="{{.User.UserEmail}}" value="{{.User.UserEmail}}" readonly ><br />

        <label class="form-group">First Name</label>
        <input contenteditable="false" type="text" class="form-control" name="first-name"
               placeholder="{{.User.FirstName }}" value="{{.User.FirstName }}" readonly><br />

        <label class="form-group">Last Name</label>
        <input contenteditable="false" type="text" class="form-control" name="last-name"
               placeholder="{{.User.LastName }}" value="{{.User.LastName }}" readonly><br />

        <div class="dropdown-divider"></div>
        <label class="form-group">Credits</label>
        <input  contenteditable="false" type="number" class="form-control" name="credits" placeholder="" required=""><br />

        <label class="form-group">Full / Part Time</label>
        <!--<input type="text" class="form-control" name="major" placeholder=""><br />-->
        <select name="student-type" required="true">
            <option value="1">Full Time</option>
            <option value="2">Part Time</option>
        </select>

        <div class="dropdown-divider"></div>

        <input type="submit" value="submit" id="submit" class="btn btn-default pull-right">

    </form>
</div>

{{ end }}
```

```
{{ define "adminFacultyDetail" }}
<div class="col-md-8 contact-left-w3l">
    <form action="/admin/user/faculty" id="requestData" class="form-horizontal" method="POST">

        <label class="form-group">Email</label>
        <input  contenteditable="false" type="email" class="form-control" name="email"
                placeholder="{{.User.UserEmail }}" required="" value="{{.User.UserEmail }}"><br />

        <label class="form-group">First Name</label>
        <input contenteditable="false" type="text" class="form-control" name="first-name"
               placeholder="{{.User.FirstName }}" value="{{.User.FirstName }}"><br />

        <label class="form-group">Last Name</label>
        <input contenteditable="false" type="text" class="form-control" name="last-name"
               placeholder="{{.User.LastName }}" value="{{.User.LastName }}"><br />

        <div class="dropdown-divider"></div>

        <label class="form-group">Department</label>
        <select contenteditable="false" name="department" required="true">
            <option value="1">Math</option>
            <option value="2">Computer Science</option>
        </select>

        <label class="form-group">Full / Part Time</label>
        <!--<input type="text" class="form-control" name="major" placeholder=""><br />-->
        <select name="faculty-type" required="true">
            <option value="1">Full Time</option>
            <option value="2">Part Time</option>
        </select>

        <div class="dropdown-divider"></div>
        <input type="submit" value="submit" id="submit" class="btn btn-default pull-right">
    </form>
</div>
{{ end }}
```

Delete User Back-End Implementation

- A form is used to search for users by name and email
- If the user exists, the subtype must be deleted first; for a student, first a full or part time student is deleted, then a student, and finally a user

```
func deleteUser(w http.ResponseWriter, r *http.Request) {
    //r.ParseForm()
    params := mux.Vars(r)
    formEmail,_ := strconv.Atoi(params["userID"])
    //formEmail := r.URL.Query().Get("userID")
    println("UserID coming in is--" + params["userID"])
    mu := model.MainUser{}
    db.Where(&model.MainUser{UserID: uint(formEmail)}).First(&mu)
    println("Fropm DB found user email: " + mu.UserEmail)
    if mu.UserID != 0 {
        userType := mu.UserType
        if userType == 1 {
            student := model.Student{}
            db.First(&student, mu.UserID)
            if student.StudentType == 1 && student.StudentID != 0 {
                studentFT := model.FullTimeStudent{}
                db.First(&studentFT, student.StudentID)
                if studentFT.FullTimeStudentID != 0 {
                    println("Deleting Full Time Student")
                    db.Delete(&studentFT)
                } else {
                    println("FT student not found")
                }
                println("Deleting Student")
                db.Delete(&student)
            } else if student.StudentType == 2 && student.StudentID != 0 {
                studentPT := model.PartTimeStudent{}
                db.First(&studentPT, student.StudentID)
                if studentPT.PartTimeStudentID != 0 {
                    println("Deleting Part Time Student")
                    db.Delete(&studentPT)
                } else {
```

```
            } else {
                println("Part time student not found")
            }
            println("Deleting Student")
            db.Delete(&student)
        } else {
            println("Student not found")
        }

    } else if userType == 2 {
        faculty := model.Faculty{}
        db.First(&faculty, mu.UserID)
        if faculty.FacultyType == 1 && faculty.FacultyID != 0 {
            studentFT := model.FullTimeStudent{}
            db.First(&studentFT, faculty.FacultyID)
            if studentFT.FullTimeStudentID != 0 {
                println("Deleting Full Time Faculty")
                db.Delete(&studentFT)
            } else {
                println("FT faculty not found")
            }
            println("Deleting faculty")
            db.Delete(&faculty)
        } else if faculty.FacultyType == 2 && faculty.FacultyID != 0 {
            facultyPT := model.PartTimeFaculty{}
            db.First(&facultyPT, faculty.FacultyID)
            if facultyPT.PartTimeFacultyID != 0 {
                println("Deleting Part Time Student")
                db.Delete(&facultyPT)
            } else {
                println("Part time faculty not found")
            }
```

```
            println("Deleting faculty")
            db.Delete(&faculty)
        } else {
            println("Faculty not found")
        }

    } else if userType == 3 {
        admin := model.Admin{}
        db.First(&admin, mu.UserID)

        if admin.AdminID != 0 {
            db.Delete(&admin)
        } else {
            println("Admin not found")
        }
        println("Deleting msin ser")
    } else if userType == 4 {
        researcher := model.Researcher{}
        db.First(&researcher, mu.UserID)

        if researcher.ResearcherID != 0 {
            db.Delete(&researcher)
        } else {
            println("Researcher not found")
        }
    }
```

```
    }

    println("Deleting main user")
    db.Delete(&mu)
} else {
    println("Main user not found")
}
data :=  map[string]interface{}{
    "deleted": mu,
}
global.Tpl.ExecuteTemplate(w, "searchUsersAdmin", data)
}
```

Delete User Front-End

```
{{ define "searchUsersAdmin" }}
{{template "head"}}
<div class="container">

    {{template "admin-navbar"}}

    <h1>Select User</h1>

    <div class="contact-w3-agileits" id="contact" >
        <div class="col-md-12 contact-left-w3l" >

            <form action="/admin/user/search" id="requestData" class="form-horizontal" method="GET">

                <label class="form-group">Email</label>
                <input  class="form-control" name="email" placeholder="" ><br />

                <label class="form-group">First Name</label>
                <input type="text" class="form-control" name="first-name" placeholder=""><br />

                <label class="form-group">Last Name</label>
                <input type="text" class="form-control" name="last-name" placeholder=""><br />


                <div class="dropdown-divider"></div>

                <input type="submit" value="submit" id="submit" class="btn btn-default pull-right">

            </form>
        </div>
    </div>
```

```
<div class="col-md-12 contact-left-w3l" >
    <ul>
        {{ if .Users }}
            {{ range .Users }}
                <li>
                    <div>
                        <h4>Email: {{ .UserEmail }} </h4>
                        <span>First Name: {{ .FirstName }},  Last Name: {{ .LastName }}</span>

                        <button id="delete-{{ .UserID }}" UserEmail="{{ .UserEmail }}"
                                UserID="{{ .UserID }}" btnAction="delete"> Delete User
                            <form action="/admin/user/{{ .UserID }}/delete" method="POST" id="{{.UserID}}">
                                <input type="submit" value="Remove User" class="btn btn-default pull-right">
                            </form>

                        </button>

                        <br/>
                    </div>
                </li>
                <br/>
            {{ end }}
        {{ else }}
            {{ if .deleted }}
                {{ .deleted.UserEmail }} Has been deleted.
            {{ else }}
                <h3> No Results Found </h3>
            {{ end }}
        {{ end }}
    </ul>
</div>
```

Researcher

Get csv file of all students who have earned within a grade range of a specific course

Back-End

```
func genReportStudentsByGrade(w http.ResponseWriter, r *http.Request){
    r.ParseForm()
    formGradeLow := r.FormValue("letter-grade-boundlow")
    formGradeHigh := r.FormValue("letter-grade-boundhigh")
    courseNum := r.FormValue("department")

    grades := []string{"F","C-","C","C+","B-","B", "B+", "A-", "A"}

    getIndex := func(vs []string, t string) int {
        for i, v := range vs {
            if v == t {
                return i
            }
        }
        return -1
    }

    indexLow := getIndex(grades, formGradeLow)
    indexHigh := getIndex(grades, formGradeHigh)
    println("gradeLow:" + formGradeLow + ", gradeHigh: " + formGradeHigh)
    println("IndexLow:" + strconv.Itoa(indexLow) + ", indexHigh: " + strconv.Itoa(indexHigh))
    gradeSlice := grades[indexLow:indexHigh+1]
```

```
contains := func(slice []string, item string) bool {
    set := make(map[string]struct{}, len(slice))
    for _, s := range slice {
        set[s] = struct{}{}
    }
    _, ok := set[item]
    return ok;
}

productsSelected := r.Form["filter-options"]

sql := `SELECT student_history.grade, main_user.first_name, main_user.last_name
FROM student_history
JOIN enrollment ON student_history.enrollment_id = enrollment.enrollment_id
JOIN section ON enrollment.section_id = section.section_id
JOIN student ON student_history.student_id = student.student_id
JOIN main_user ON main_user.user_id = student.student_id
WHERE
student_history.grade IN (?)
AND student_history.status = ?
AND section.course_id = ? `

//if contains(productsSelected, "major"){
//  major := r.FormValue("major")
//  sqlDepartmentFilter := ` AND student.student_id IN
//  (SELECT student_id FROM student_major
//  JOIN major ON student_major.major_id = major.major_id
//  WHERE major.department_id = `+ major + " )"
//
//  sql += sqlDepartmentFilter
//}
```

```
if contains(productsSelected, "student-type"){
    stuType := r.FormValue("full-or-part")
    println("Filtering for student type: " + stuType)
    sqlDepartmentFilter := ` AND student.student_type = `+ stuType
    sql += sqlDepartmentFilter
}

type StudentData struct {
    FirstName string
    LastName string
    Grade string
}
records := []StudentData{}
db.Raw(sql, gradeSlice, "Complete", courseNum).Scan(&records)

modtime := time.Now()
filepath := "Record" + strconv.Itoa(modtime.Nanosecond()) + ".csv"
outfile, err := os.Create("./"+filepath)

if err != nil {
    log.Fatal("Unable to open output")
}
defer outfile.Close()
writer := struct2csv.NewWriter(outfile)

for _, record := range records {
    if err := writer.WriteStruct(record); err != nil {
        log.Fatalln("error writing record to csv:", err)
    } else {
        println("FNAME: " + record.FirstName + ", LNAME: " + record.LastName + ", GRADE: " + record.Grade
    }
}
```

```
    writer.Flush()
    // tell the browser the returned content should be downloaded
    w.Header().Set("Content-Type", "text/csv")
    path := "./"+filepath
    w.Header().Set("Content-Disposition", "attachment; filename="+filepath)

    http.ServeFile(w,r,path)
    global.Tpl.ExecuteTemplate(w, "researchStudentsByGrade", nil)
    os.Remove(path)
}
```

Researcher finds students by grades in course Front End

```
{{ define "researchStudentsByGrade" }}
{{ template "head" }}

<div class="contact-w3-agileits" id="contact" >
    <div class="container">
        <div class="col-md-12 contact-left-w3l" >
            <h3>Student Report by Grade</h3>
            <div class="visit">

                <div class="clearfix"></div>
            </div>

            {{ template "researcher-navbar" }}

            <div class="col-md-8 contact-left-w3l">
                <h3>Grade Criteria</h3>
                <form action="/research/students/grades" id="requestData" class="form-horizontal" method="POST">
                    <label class="form-group">Find all students with a grade at least</label>
                    <select name="letter-grade-boundlow" class="form-control">
                        <option value="A">A</option>
                        <option value="A-">A-</option>
                        <option value="B+">B+</option>
                        <option value="B">B</option>
                        <option value="B-">B-</option>
                        <option value="C+">C+</option>
                        <option value="C">C</option>
                        <option value="C-">C-</option>
                        <option value="F">F</option>
                    </select>
```

```
                    <label class="form-group">but not greater than</label>
                    <select name="letter-grade-boundhigh" class="form-control">
                        <option value="A">A</option>
                        <option value="A-">A-</option>
                        <option value="B+">B+</option>
                        <option value="B">B</option>
                        <option value="B-">B-</option>
                        <option value="C+">C+</option>
                        <option value="C">C</option>
                        <option value="C-">C-</option>
                        <option value="F">F</option>
                    </select>
                    <br/>



                    <label class="form-group">Course</label>
                    <select name="department" class="form-control">

                        {{ range .Courses }}

                        <option
                                value="{{ .CourseID }}">
                            {{ .CourseName }}
                        </option>

                        {{ end }}
                    </select><br/>
```

```
                    <br />

                    <h3>Additional Filters:</h3>

                    <label class="form-group">Full / Part Time</label>
                    <input type="checkbox" name="filter-options" value="student-type">
                    <select name="full-or-part" required="true">
                        <option value="1">Full Time</option>
                        <option value="2">Part Time</option>
                    </select>
                    <br />


                    <input type="submit" class="btn-success"  value="Generate Report">
                    <!--<form>-->
                    <!---->
                    <!--</form>-->
                    <!--<input type="button" class="btn-success" value="Reset">-->
                </form>
            </div>
        </div>
    </div>
</div>

{{template "end"}}

{{end}}
```

Faculty

Faculty View Schedule

- Allows the faculty to view the courses he is currently teaching for the fall 2017 semester

```go
package main

import ...

func facultyViewSchedule(w http.ResponseWriter,r *http.Request){

    _, user := CheckLoginStatus(w,r)
    type FacultySchedule struct{
        CourseName string
        BuildingName string
        RoomNumber string
        MeetingDay string
        Time string
        Year int
        Season string
    }

    facultySchedule := []FacultySchedule{}
    db.Raw( sql: `
    SELECT course_name, building_name,room_number,meeting_day,time,year,season
    FROM section
    JOIN course ON course.course_id = section.course_id
    JOIN time_slot ON time_slot.time_slot_id = section.time_slot_id
    JOIN semester ON time_slot.semester_id = semester.semester_id
    JOIN period ON time_slot.period_id = period.period_id
    JOIN day ON time_slot.day_id = day.day_id
    JOIN location ON section.location_id = location.location_id
    JOIN building ON location.building_id = building.building_id
    JOIN room ON location.room_id = room.room_id
    WHERE section.faculty_id = ? AND semester.year = ? AND semester.season = ?`, user.UserID,2017,"Fall").Scan(&facultySchedule)
    //TODO: these are hardcoded for current semester, fix these at some point
```

Faculty Give Grades Form

- **Allows the faculty to view the courses he is currently teaching for the fall 2017 semester with the option to submit grades for the current course. If the faculty member chooses to submit the grades, he will be be shown an additional form with the names of students enrolled in that course and an option to submit grades for them.**

```
    fmt.Println(facultySchedule)

    err := global.Tpl.ExecuteTemplate(w, "ViewFacultyScheduleDetails", facultySchedule)
    if err != nil {
        fmt.Println(err.Error())
    }
}


func giveStudentGradesPage(w http.ResponseWriter, r *http.Request){

    _, user := CheckLoginStatus(w,r)
    type FacultyScheduleGrade struct{
        CourseName string
        SectionID uint
        BuildingName string
        RoomNumber string
        MeetingDay string
        Time string
        Year int
        Season string
    }

    facultySchedule := []FacultyScheduleGrade{}
    db.Raw( sql: `
    SELECT course_name,section.section_id,building_name,room_number,meeting_day,time,year,season
    FROM section
    JOIN course ON course.course_id = section.course_id
    JOIN time_slot ON time_slot.time_slot_id = section.time_slot_id
    JOIN semester ON time_slot.semester_id = semester.semester_id
    JOIN period ON time_slot.period_id = period.period_id
    JOIN day ON time_slot.day_id = day.day_id
    JOIN location ON section.location_id = location.location_id
```

```
    facultySchedule := []FacultyScheduleGrade{}
    db.Raw( sql: `
    SELECT course_name,section.section_id,building_name,room_number,meeting_day,time,year,season
    FROM section
    JOIN course ON course.course_id = section.course_id
    JOIN time_slot ON time_slot.time_slot_id = section.time_slot_id
    JOIN semester ON time_slot.semester_id = semester.semester_id
    JOIN period ON time_slot.period_id = period.period_id
    JOIN day ON time_slot.day_id = day.day_id
    JOIN location ON section.location_id = location.location_id
    JOIN building ON location.building_id = building.building_id
    JOIN room ON location.room_id = room.room_id
    WHERE section.faculty_id = ? AND semester.year = ? AND semester.season = ?`, user.UserID,2017,"Fall").Scan(&facultySchedule)
    fmt.Println(facultySchedule)
    //TODO: these are hardcoded for current semester, fix these at some point
    err := global.Tpl.ExecuteTemplate(w, "FacultyGiveGrades", facultySchedule)
    if err != nil {
        fmt.Println(err.Error())
    }
}

func giveStudentGradesForm(w http.ResponseWriter, r *http.Request) {

    vars := mux.Vars(r)
    sectionID := vars["sectionID"]
    sectionIDint,_ := strconv.Atoi(sectionID)

    type GradingPeriod struct {
        SectionID uint
        SemesterID uint
        SemesterStatus string
    }
```

```
gp := GradingPeriod{}

db.Raw( sql: `
    SELECT section.section_id, semester.semester_id, semester_status
    FROM section
    JOIN time_slot ON section.time_slot_id = time_slot.time_slot_id
    JOIN semester ON time_slot.semester_id = semester.semester_id
    WHERE section.section_id = ?
`, sectionIDint).Scan(&gp)

if strings.Compare(gp.SemesterStatus, b: "Grading") != 0 || strings.Compare(gp.SemesterStatus, b: "Open") != 0{
    fmt.Println( a: "Error it is not currently a grading period")
    global.Tpl.ExecuteTemplate(w, "facultySuccess", "Error it is not currently a grading period")
    return
}

type CourseInfo struct{
    CourseName string
    SectionID uint
}

courseDetail := CourseInfo{}

db.Raw( sql: `
    SELECT course_name,section_id
    FROM section
    JOIN course ON course.course_id = section.course_id
    WHERE section.section_id = ?`,sectionIDint).Scan(&courseDetail)

fmt.Println( a: "Course Detail is", courseDetail)

type GradingForm struct{
    StudentID uint
```

```
    type GradingForm struct{
        StudentID uint
        FirstName string
        LastName string
        Status string
    }

    gradeForm := []GradingForm{}

    db.Raw( sql: `
    SELECT student_history.student_id,first_name,last_name,student_history.status
    FROM student_history
    JOIN enrollment ON student_history.enrollment_id = enrollment.enrollment_id
    JOIN section ON enrollment.section_id = section.section_id
    JOIN student ON enrollment.student_id = student.student_id
    JOIN main_user ON main_user.user_id = student.student_id
    WHERE section.section_id = ? AND student_history.status = 'In progress'`,sectionIDint).Scan(&gradeForm)

    fmt.Println(gradeForm)

    m := map[string]interface{}{
        "Course":courseDetail,
        "Roster":gradeForm,
    }

    err := global.Tpl.ExecuteTemplate(w, "FacultyGiveGradesForm", m)
    if err != nil {
        fmt.Println(err.Error())
    }

}

func submitGrades(w http.ResponseWriter, r *http.Request){
```

Faculty Submit Grades

Use Case Name	Add Grades
Description	Adds the final grades for students to the system.
Participating Actors	Faculty
Input	Semester
Output	A list of Grades
Flow	1) The user selects a section 2) A list of students are displayed. 3) The user submits a form that has the student grades. 4) The user receives a success message.
Entry Condition	The user is logged into the system.
Exit Condition	The user will be able to view students grades of a selected course for the current semester.

- **Allows the faculty to view to submit grades for his courses as long as it's a grading period. This method retrieves all the students id's from the section and the section info. For all the students, it loops through updates their grades in the database on the Student History Table.**

```
func submitGrades(w http.ResponseWriter, r *http.Request){

    vars := mux.Vars(r)
    sectionID := vars["sectionID"]
    sectionIDint,_ := strconv.Atoi(sectionID)
    //grades := r.FormValue("grade")
    r.ParseForm()
    id := r.Form["studentId"]
    grades := r.Form["grade"]


    type Err struct {
        error string
    }

    e := Err{}

    for k,_ := range grades{
        db.Raw( sql: `
            UPDATE student_history
            SET grade = ?,
            status = 'Complete'
            FROM enrollment
            WHERE enrollment.enrollment_id = student_history.enrollment_id
            AND enrollment.section_id = ?
            AND student_history.student_id = ?
        `,grades[k],sectionIDint,id[k]).Scan(&e)
        fmt.Println( a: "Updating grade with info sec, grade, stud_id", sectionIDint,grades[k], id[k])
    }

    fmt.Println(e)

    global.Tpl.ExecuteTemplate(w, "facultySuccess", "Grades sucessfully submitted")
```

Student

Student View Schedule

Use Case Name	View Schedule
Description	Displays a list of all the students courses for a selected semester, for a selected student.
Participating Actors	Student
Input	Semester
Output	A list of Courses
Flow	1) The user submits the form. 2) A list of students courses for the selected system are displayed
Entry Condition	The user is logged into the system.
Exit Condition	The user will be able to view the a list of courses they are taking for the current semester.

- **Works similar to Admin view student schedule, except in this function, the student's own id is passed in as a query parameter.**

```go
package main

import ...

func ViewSchedule(w http.ResponseWriter, r *http.Request){


    type StudentSchedule struct {
        CourseName string
        CourseCredits string
        RoomNumber string
        BuildingName string
        Time string
        MeetingDay string
        FirstName string
        LastName string
    }


    _, user := CheckLoginStatus(w,r)
    ss := []StudentSchedule{}
    db.Raw( sql: `SELECT student_history.student_id,course_name,course_credits,building_name,room.room_number,meeting_day,
                first_name, last_name, time,student_history.status
    FROM student_history
    JOIN enrollment ON student_history.enrollment_id = enrollment.enrollment_id
    JOIN section ON enrollment.section_id = section.section_id
    JOIN course ON course.course_id = section.course_id
    JOIN time_slot ON time_slot.time_slot_id = section.time_slot_id
    JOIN semester ON time_slot.semester_id = semester.semester_id
    JOIN period ON time_slot.period_id = period.period_id
    JOIN day ON time_slot.day_id = day.day_id
    JOIN location ON section.location_id = location.location_id
    JOIN building ON location.building_id = building.building_id
    JOIN room ON location.room_id = room.room_id
    JOIN faculty ON section.faculty_id = faculty.faculty_id
    JOIN main_user ON faculty.faculty_id = main_user.user_id
```

Student View Registered Courses

- **Allows the faculty to view the sections that he is currently registered for. These can be distinguished from other courses because they are located in the student_history table with a status of ‘registered’.**

```
    JOIN main_user ON faculty.faculty_id = main_user.user_id
    WHERE enrollment.student_id = ? AND student_history.status = 'In progress'`, user.UserID).Scan(&ss)
    fmt.Println(ss)

    err := global.Tpl.ExecuteTemplate(w, "ViewStudentScheduleDetails", ss)
    if err != nil {
        fmt.Println(err.Error())
    }
}

func ViewRegisteredCourses(w http.ResponseWriter, r *http.Request){

    type StudentSchedule struct {
        CourseName string
        CourseCredits string
        RoomNumber string
        BuildingName string
        Time string
        MeetingDay string
        FirstName string
        LastName string
        SectionID uint
        StudentID uint
    }


    _, user := CheckLoginStatus(w,r)
    ss := []StudentSchedule{}
    db.Raw( sql: `SELECT student_history.student_id,course_name,section.section_id,course_credits,building_name,
              room.room_number,meeting_day, first_name, last_name, time,student_history.status
    FROM student_history
    JOIN enrollment ON student_history.enrollment_id = enrollment.enrollment_id
    JOIN section ON enrollment.section_id = section.section_id
    JOIN course ON course.course_id = section.course_id
    JOIN time_slot ON time_slot.time_slot_id = section.time_slot_id
    JOIN semester ON time_slot.semester_id = semester.semester_id
```

Student Drop Course

Student Remove Course

Use Case Name	Remove Course
Description	Removes a course from a student's schedule for the following semester.
Participating Actors	Student
Input	Course ID
Output	Boolean
Flow	1) The user selects the course they would like to delete. 2) The user submits the form. 3) The user gets a success or failure message.
Entry Condition	The user is logged into the system. It must be during the within the time period to remove a course.
Exit Condition	The user will be able to view the a list of courses they are taking for the current semester.

- **Allows the student to drop a section he is currently registered for. As long as it's a registration period, they are aloud to drop the course.**

```
func DropRegisteredCourse(w http.ResponseWriter, r *http.Request){
        sec := r.FormValue("section")
        stud := r.FormValue("student")


        fmt.Println("sec id", sec)
        fmt.Println("stud id", stud)

        enrollment := model.Enrollment{}
        db.Table("enrollment").Select("*").Where("student_id = ? AND section_id = ?",stud,sec).Scan(&enrollm

        hist := model.StudentHistory{}
        db.Table("student_history").Select("*").Where("enrollment_id = ? AND student_id = ?", enrollment.Enr

        db.Delete(&hist) //delete the hist
        db.Delete(&enrollment) //delete the enrollment

        global.Tpl.ExecuteTemplate(w, "studentSuccess", "The course has been dropped")
}
```

Student Register Section

Student Add Section

Use Case Name	Add Course
Description	Adds a course to a student's schedule for the following semester.
Participating Actors	Student
Input	Course ID
Output	Boolean
Flow	1) The user enters the search criteria for the master schedule 2) The user submits the form 3) The user clicks on the register button next to the section they want. 4) The user gets a success or failure method.
Entry Condition	The user is logged into the system. It must be during the course registration time period.
Exit Condition	The user will get a success or failure message to let them know if the course was added to their schedule.

- **Allows the student to register for a course. Numerous checks must happen before the student is allowed to register for a course. They must have the necessary prerequisites, their must be no concurrent courses, they can't have holds, they section can't be full, they can't exceed their max credit limit.**

```
func RegisterForSection(w http.ResponseWriter, r *http.Request){
    section := r.FormValue( key: "section")
    _,user := CheckLoginStatus(w,r)
    fmt.Println( a: "Section is" ,section)

    //TODO check if its a registration period

    //TODO check if room is full

    holds := 0

    db.Table( name: "student_holds").Select( query: "*").Where( query: "student_id = ?",user.UserID).Count(&holds)
    //check holds
    if holds > 0 {
        fmt.Println( a: "Error can't regester because you have holds")
        global.Tpl.ExecuteTemplate(w, "studentSuccess", "Error can't regester because you have holds" )
        return
    }else {
        fmt.Println( a: "No holds, continue on") //holds check works , add view
    }

    student := model.Student{}
    maxCredits := 0

    //db.Where(model.Student{StudentID:user.UserID}).Scan(&student)
    db.First(&student,user.UserID)
    fmt.Println( a: "Found student", student)

    if student.StudentType == 1{ //if full time student limit is 16
        maxCredits = 16
    }else { //part time limit is 8
        maxCredits = 8
    }

    type CourseRegistering struct {
        SectionID uint
```

```go
type CourseRegistering struct {
    SectionID uint
    CourseID uint
    CourseCredits int
    MeetingDay string
    Time string
    Capacity int
}



//course attempting to register
courseRegistering := CourseRegistering{}
totalCredits := 0

db.Raw( sql: `
    SELECT section.section_id, course.course_id, course_credits,meeting_day,time,capacity
    FROM section
    JOIN course on course.course_id = section.course_id
    JOIN time_slot ON time_slot.time_slot_id = section.time_slot_id
    JOIN semester ON time_slot.semester_id = semester.semester_id
    JOIN period ON time_slot.period_id = period.period_id
    JOIN day ON time_slot.day_id = day.day_id
    WHERE section.section_id = ?

`,section).Scan(&courseRegistering)

fmt.Println( a: "Course registering is", courseRegistering)

//splitSlot := strings.Split(courseRegistering.Time,"-")
//stripWhite := strings.Split(splitSlot[1]," ")
//fullSecondHalfofRegisteringCourse := stripWhite[1]+ " "+stripWhite[2]



//TODO course capacity check , do this by checking the count of enrollments for that section then compare it against the section capacity
```

```
count := 0

db.Table( name: "enrollment").Select( query: "*").Where( query: "section_id = ?",section).Count(&count)

if count == courseRegistering.Capacity {
    fmt.Println( a: "Course enrollment has reached capacity")
    global.Tpl.ExecuteTemplate(w, "studentSuccess", "The section has reached the maximum capacity")
    return
}

type RegistrationCheck struct {
    SectionID uint
    CourseID uint
    CourseCredits int
    MeetingDay string
    Time string
    Grade string
    Status string
}

registrationCheck := []RegistrationCheck{}

fmt.Println( a: "Attempting to query users already enrolled courses")
//already enrolled courses
db.Raw( sql: `
        SELECT section.section_id,section.course_id,course.course_credits, meeting_day,time, grade, status
        FROM section
        JOIN course ON course.course_id = section.course_id
        JOIN  enrollment ON enrollment.section_id = section.section_id
        JOIN student_history ON enrollment.enrollment_id = student_history.enrollment_id
        JOIN time_slot ON time_slot.time_slot_id = section.time_slot_id
        JOIN semester ON time_slot.semester_id = semester.semester_id
        JOIN period ON time_slot.period_id = period.period_id
        JOIN day ON time_slot.day_id = day.day_id
        WHERE enrollment.student_id = ? AND status = ?
```

```
	fmt.Println("Registered courses are" , registrationCheck)

	//check course attempting to register vs courses already registered
	for i := 0; i < len(registrationCheck); i++{
		//overlapTime := registrationCheck[i].Time
		//overlapTimeFirstHalf := strings.Split(overlapTime,"-")

		totalCredits += registrationCheck[i].CourseCredits
		if totalCredits >= maxCredits { //check max credits
			fmt.Println("Error you are over the credit limit for your student type")
			global.Tpl.ExecuteTemplate(w, "studentSuccess", "Error you are over the credit limit for your student type" )
			return
		}

		//check if student is already registered for that course this semester
		if courseRegistering.CourseID == registrationCheck[i].CourseID {
			fmt.Println("Error, you are already registered for that same course")
			global.Tpl.ExecuteTemplate(w, "studentSuccess" , "Error, you are already registered for that same course")
			return
		}

		//check if time and day are equal /
		if strings.Compare(courseRegistering.Time,registrationCheck[i].Time) == 0 &&
		strings.Compare(courseRegistering.MeetingDay,registrationCheck[i].MeetingDay) == 0 {
			fmt.Println("Error you are already registrered for that time slot at that day")
			global.Tpl.ExecuteTemplate(w, "studentSuccess", "Error you are already registrered for that time slot at that d
			return
		}


	}

	course := model.Course{}
```

```
}

course := model.Course{}

db.Where(model.Course{CourseID:courseRegistering.CourseID}).Find(&course)

//* prereq works
prereqs := course.FindCoursePrerequisites(db)
fmt.Println( a: "Prereqs for this course are ", prereqs)
//if the course has prereqs check to make sure the student has all the prereqs
if len(prereqs) > 0 {
    numPreReqs := len(prereqs)
    coursesTaken := []model.Course{}
    //var hasTakenList [numPreReqs]bool
    hasTakenList := make([]bool, numPreReqs)
    //set courses taken to a boolean
    db.Raw( sql: `
        SELECT course.course_id
        FROM section
        JOIN course ON course.course_id = section.course_id
        JOIN  enrollment ON enrollment.section_id = section.section_id
        JOIN student_history ON enrollment.enrollment_id = student_history.enrollment_id
        WHERE enrollment.student_id = ? AND status != ?;
    `,user.UserID,"Registered").Scan(&coursesTaken)


    fmt.Println( a: "Before looping")
    fmt.Println( a: "Courses Taken", coursesTaken)
    fmt.Println( a: "Pre Reqs needed", prereqs)
    fmt.Println( a: "Boolean array", hasTakenList)
    //first check for each of their preques, they have taken the course
    for i := 0; i < len(prereqs); i++{
        for j := 0; j < len(coursesTaken); j++{
            if prereqs[i].CourseID == coursesTaken[j].CourseID {
                hasTakenList[i] = true
                fmt.Println( a: "Change is has taken array", hasTakenList)
```

```
            }
        }

    }

    fmt.Println( a: "After looping through courses taken and checking with prereqs")
    fmt.Println( a: "Boolean array", hasTakenList)


    for i := 0 ; i < len(hasTakenList); i++{
        if hasTakenList[i] == false {
            fmt.Println( a: "Error, missing pre-req") //Todo test this more
            global.Tpl.ExecuteTemplate(w, "studentSuccess", "Error, you are missing pre-reqs" )
            return
        }
    }
}
fmt.Println( a: "Made it to bottom of function, attempting to insert enrollment & history")
//IF we make it here everything is checked for and we can insert the enrollment
enrollment := model.Enrollment{StudentID:user.UserID, SectionID:courseRegistering.SectionID}

db.Create(&enrollment) //enroll
fmt.Println( a: "created enrollment, ", enrollment)


history := model.StudentHistory{StudentID:user.UserID, EnrollmentID:enrollment.EnrollmentID,Status:"Registered", Grade:"-"}
db.Create(&history)
fmt.Println( a: "Created history,", history)

global.Tpl.ExecuteTemplate(w, "studentSuccess", "Registration successful." )


}
```

Student View Advisor

Use Case Name	View Advisor
Description	Displays Advisor(s) for a selected student.
Participating Actors	Student
Input	Student ID
Output	Advisor
Flow	1) The user selects the view administrator button in the sub navigation bar. 2) The user submits the form. 3) The user will be able to view their advisor and all their information.
Entry Condition	The user is logged into the system.
Exit Condition	The user will be able to view their advisor

View Advisor Front End Implementation

- **After clicking on view advisor in the student sub-navigation bar, the detailed information about the students advisor will appear.**

```
{{define "ViewAdvisor"}}
{{template "head"}}


<div class="container">


    {{template "student-navbar"}}

    <h1>Advising Data</h1>

    {{if .}}
    <div class="course_details">
        <p>Advisor Name {{.FirstName}} {{.LastName}}</p>
        <p>Room Number: {{.RoomNumber}}</p>
        <p>Department: {{.DepartmentName}}</p>
        <p>Building Name: {{.DepartmentBuilding}}</p>
        <p>Department Number: {{.DepartmentPhoneNumber}}</p>
    </div>
    <hr>
    {{else}}
    <h1>Error loading advisor.</h1>
    {{end}}



</div>

{{template "end"}}
{{end}}
```

View Advisor Back End Implementation

- **The user's ID is submitted as a database request, along with all the necessary informat to display to the user.**

```
func ViewAdvisor(w http.ResponseWriter, r *http.Request){

        _ ,user := CheckLoginStatus(w,r)

        type AdvisingData struct {
                FacultyID uint
                FirstName string
                LastName string
                RoomNumber string
                DepartmentName string
                DepartmentBuilding string
                DepartmentPhoneNumber string
        }

        ad := AdvisingData{}

        db.Raw(`
                SELECT faculty.faculty_id, first_name, last_name, room_number,department_building,
                FROM advises
                JOIN faculty ON advises.faculty_id = faculty.faculty_id
                JOIN main_user ON faculty.faculty_id = main_user.user_id
                JOIN department ON department.department_id = faculty.department_id
                WHERE advises.student_id = ?
                `,user.UserID).Scan(&ad)


        global.Tpl.ExecuteTemplate(w, "ViewAdvisor", ad)

}
```

Record of Group Meeting

Date	Start Time	End Time	Notes	Ben Xerri	Philip Witkin	Kwame Mensah
9/7/17	12:30	1:30	Decided what belonged in the project	✓	✓	NA
9/11/17	5:30	6:30	Discussed project proposal and system requirements. Collaborated on google docs	✓	✓	✓
9/12/17	1:00	2:00	Worked on project proposal Collaborated on google docs	✓	✓	✓
9/19/17	1:00	3:00	Discussed requirements. Collaborated on google docs	✓	✓	✓
9/26/17	1:00	3:00	Worked on requirements. Collaborated on google docs.	✓	✓	✓
10/3/17	11:20	12:50	Finished requirements, Started Specifications	✓	✓	✓
10/5/17	1:00	3:00	Continued work on specifications, Collaborated on google docs.	✓	✓	✓
10/10/17	1:00	3:00	Finished specifications	✓	✓	✓
10/12/17	1:00	3:00	Revised Requirements and Specifications	✓	✓	✓
10/17/17	1:00	3:00	Started ERD	✓	✓	✓
10/25/17	1:00	3:00	Distributed Workload	✓	✓	✓

11/2/17	1:00	3:00	Started Implementation	✓	✓	✓
11/5/17	1:00	3:00	Working on Implementation	✓	✓	✓
11/9/17	1:00	3:00	Working on Implementation	✓	✓	✓
11/15/17	1:00	3:00	Working on Implementation	✓	✓	✓
11/17/17	1:00	3:00	Working on Implementation	✓	✓	✓
11/21/17	1:00	3:00	Working on Implementation	✓	✓	✓
11/24/17	1:00	3:00	Working on Implementation	✓	✓	✓
11/27/17	1:00	3:00	Working on Implementation	✓	✓	✓
11/30/17	1:00	3:00	Working on Implementation	✓	✓	✓
12/7/17	1:00	3:00	Working on Implementation	✓	✓	✓
12/9/17	1:00	3:00	Working on Implementation	✓	✓	✓

Final Notes

Starfleet Academy was a group project created for System Design and Implementation during our senior year. It is an open source project. The code is available at

https://github.com/PicardsFlute/Starfleet

Current Contributors:

Back-End Work
Benjamin Xerri - Group Leader
Philip Witkin

Front-End Work
Kwame Mensah
Benjamin Xerri
Philip Witkin

Starfleet academy is currently hosted on Heroku at

https://shrouded-scrubland-50647.herokuapp.com/

Using Heroku's generous free plan, we were able to host our entire project with no cost to us as students.