CS440: Introduction to Software Engineering

Design Document

Group members:
Kamran Lodhi mlodhi3@uic.edu
Nazari Skrupsky nskrou2@uic.edu
Nestor Sotres nsotre2@uic.edu
Michal Szumilo mszumi2@uic.edu

1. Introduction

a) Preface

The overall goal is to provide a centralized access point for students to sign up for classes, review their degree progress, and allow for the staff to easily set up and manage the semesters course assignments and scheduling.

Another goal will be to deploy an easy to use intuitive design for both students and staff while applying modular based architectural design.

b) Scope

The general scope of the software solution will be limited to that of aiding students in registering for courses and allowing them to review their degree audits. The application will not help a student, change, search, or otherwise manage their degree progress.

For Staff, the scope will be in assisting the set up of a term's courses and general scheduling. In a working state the application will be able to accept a list of subjects (Math, Computer Science, etc) and courses within the subjects (eg Linear Algebra, Computer Design, etc). The subject and course catalog can be created, modified, or removed by staff. Alternately, the subject and course lists can be imported from a csv or xml file. Scheduling of course time, building, classroom, Professor, T.A., and other general scheduling related attributes can be added, deleted, and modified by the staff.

Some of the major inputs will come from comma delimited, XML files, or direct input using the application's GUI. Each semester a new set of student records and class schedules will have to be entered into the system in a specified format. To make courses available for student registration, staff can then add terms and add term subjects and term courses from the list of available subjects and courses. Term subjects and term courses are associated with a particular term and contain details about the specific course offering during that term(e.g., professor, location, time, etc).

c) Definition of Acronyms

GUI - Graphical User Interface
GPA - Grade Point Average
TA - Teaching Assistant

2. Class Diagram

Entity Classes

Account Class (Abstract)

Name: String 15 chars
Surname: String 15 chars
Address: String 50 chars
Phone: Integer 10 digits
Email: String (Format: account@host.{com,net,org,etc}) 35 chars
Id: Integer (9 digit randomly generated number)
Role: {Administrator,Adviser,TA,Professor,Student} 16 chars

getters:
name(); //return the name
surname(); //return the surname
address(); //return the address
phone(); //return the phone
email(); //return the email
id(); //return the id

setters:
name(name:String); //update the name
surname(surname:String); //update the surname
address(address:String); //update the address
phone(phone:integer); //update the phone
email (email:string); //update the email

methods:
modifyUser(id: Integer, name: String, surname: String, address: String, phone: integer, email: String, role: String); //update the user account

Administrative Account Class (Concrete) inherits from Account Class

role= “Administrator” String 16 Chars

Academic Account Class (Abstract) inherits from Account Class

Emergency Contact Name: String 30 chars
Emergency Contact Phone: Integer 10 digits

getters:
emergencyContactName(); //return the emergency contact name
emergencyContactPhone(); //return the emergency contact phone

setters:
emergencyContactName(contactName:String); //update the emergency contact name
emergencyContactPhone(contactPhone:Integer); //update the emergency contact name

Advising Account Class (Concrete) inherits from Academic Account Class

role = “Adviser” String 16 chars
adviserID : integer 16 digits
advisees : Set (Integers) 16 digits
removeHold(studentId: Int);
assignAdvisor(studentId: Int);
acceptNewStudent(studentId: Int);

TA Account Class (Concrete) inherits from Academic Account Class

Role = “TA” String 16 chars
TA (Id) : Integer
TA_Courses : List

getTheListOfStudentsForCourse(courseId : String);
contactCourseStudents (studentsList : List / Array);
contactProfessor (professorID : Integer);
coursesAssignedToTA (studentTAId : Integer, semesterId : String) // the ID will be sent to course class and the function will return the list of all the courses that the TA is taking during any semester.

Professor Account Class (Concrete) inherits from Academic Account Class

professorId : integer 16 digits
Role = “Professor” String 16 chars
ListOfCoursesForTeaching : List

getTheListOfStudentsForCourse(courseId : String)

coursesBeingTaughtByProfessor(professorId : Integer, semesterId: String) // the professor Id will be sent to the course class and it will return the list of all the courses that are being taught by the professor during the semester. In this way, he will be able to look at the list of all the courses that he has taught previously or is going to teach now.

contactCourseStudents (studentsList : List / Array, message : String);
contactCourseTA (studentTAID : Integer, message : String);

Student Account Class (Concrete) inherits from Academic Account Class

Role = “Student” String 16 chars
Major: String String 32 chars
GPA: float 4 digits
studentID: integer 16 digits
AdviserID: integer 16 digits
Enrolled Courses: *Set* (Term Courses) // This should be Schedule Class Object
StudentID: String // Helps differentiate between similar Student names

generateSchedule();//Create new schedule using the Schedule Maker
lookUpCourses();   //Call to Schedule Maker Class

Hold Class (Abstract)

Active: Boolean
PlacedBy(ID): Integer 9 digits
DateTimePlaced: DateTime format: hh:mm:ss mm/dd/yyyy

getters:
active(); //return status of hold (true/false)
placedBy(); //return the id of the account that placed the hold
dateTimePlaced(); //return the DateTime when hold was placed

methods:
removeHold(); //set the status to false to disable the hold
addHold();

displayHolds();   // displays holds for current user

Academic Standing Hold Class (Concrete) inherits from Hold Class

Description: String 255 chars

getters:
description(); //return the description

Unpaid Tuition Hold Class (Concrete) inherits from Hold Class

Balance: Float   8 digits

getters:
balance(); //return the balance

Advising Hold Class (Concrete) inherits from Hold Class

Advisor(id): Integer 9 digits

getters:
advisor(); //return the advisor id

Medical Hold Class (Concrete) inherits from Hold Class

Immunization: Boolean
Health Insurance: Boolean

getters:
immunization(); //return the status of immunization flag (t/f)
healthInsurance(); //return the status of health insurance flag (t/f)

Subject Class (Concrete)

Name: String 25 chars
College: String 25 chars
Department: String 25 chars //here just for aesthetic reasons. not used anywhere else? can be removed
Courses(names): list(Courses)

getters:
name(); //return the name of the subject
college(); //return the name of the college
department(); //return department name
courses(); //return the list of courses that belong to the subject

setters:
name(name); //update the subject name
college(college); //update the college affiliation
department (department); //update the department affiliation

methods:
addCourse(courseName);// create a new course with the provided name and add it to the subject
removeCourse(courseName); //delete the course with the provided name and remove it from the subject

Course Class (Concrete)

name: String 25 chars
subject(name): String 25 chars
description: String 255 chars
prerequisite Requirements(names): Courses
creditHours: Integer 1 digit

getters:
name(); //return the course name
subject(); //return the subject name
description(); //return the course description
prerequisites(); //return the prerequisite courses
creditHours(); //return the number of credit hours

setters:
name(name); //update the course name
subject(subject); //update the subject membership
description(description); //update the description
prerequisites(prerequisites); //update the prerequisite course requirements
creditHours (hours); //update the number of credit hours

Term Subject Class (Concrete)

Term(Id): Integer 3 digits
Subject(name): String 25 digits //this is the general subject (not a term subject)
TermCourses(Ids): list(Integer 9 digits)

getters:
termCourses(); //return the list of term courses that belong to the term subject

methods:
addTermCourse(courseName); //create a new term course with the provided name and add it to the term subject
removeTermCourse(courseName); //delete the term course with the provided name and remove it from the subject

Term Course Class (Concrete)

Id: Integer 9 digits
name: String 25 digits
classroom location: String 25 digits
meetingTime: String 25 digits
classCapacity: Integer 3 digits
instructor(Id): Integer 9 digits
tA(Id): Integer 9 digits
students(Ids): list(Integer 9 digit)

getters:
name(); //return the term course name
classroomLocation(); //return the classroom location
meetingTime(); //return the meeting time
capacity(); //return the capacity
instructor(); //return the instructor
ta(); //return the ta
students(); //return the list of enrolled students

Term Class (Concrete)

Id: Integer 3 digits
Year: Integer 4 digits

Semester: {Fall,Sprint,Summer}
Registration Start Date: DateTime format: hh:mm:ss mm:dd:yyyy
Registration End Date: DateTime format: hh:mm:ss mm:dd:yyyy
TermSubjects(Ids): list(Integer 6 digits)

getters:
Id(); //return the term id
year(); //return the term year
semester(); //return the semester year
registrationStart(); //return the start datetime of registration
registrationEnd(); //return the end datetime of registration
termSubjects(); //return the list of term subjects
termSubject(termSubjectName); //return the term subject provided by name

methods:
addTermSubject(subjectName); //create a new term subject with the provided name and add it to the term
removeTermSubject(subjectName); //delete the term subject with the provided name and remove it from the term
isRegistrationOpen(); //determine if registration is open and return the status(t/f)

Schedule Class

Student: User Account Class
CoursesSignedUpFor: Term Course Class List

addCourse();
dropCourse();
printSchedule();
messageProf(course : String); //Using the name of a class the module will contact the professor using the Message Manager
getOnWaitingList(course : String);
viewCourseDetail(course : String);

Calendar Class

assignAnEventOnDate(event: Event, date:DateTime, location:String) // using this class, an event can be assigned on a particular date and time.

isDateValid(date: DateTime) // this function checks whether the date entered is valid or not. Also, it checks whether the event times are within a particular range or not, for example, all events can take place only between 9 am till 9 pm.

checkIfNoClash(date : DateTime, location : String) // this class is a boolean function, it will be passed a date to determine whether any event can be signed on that date.

Event Class

name : String (16 char)
description : String (64 char)
eventTime : Date Class

Date Class

year : int (4 digits)
month: int (2 digits)
day : int (2 digits)
format : String (8 char)

College Class (Concrete)
//new class
//the college class will hold the name of the college and a subject list

Name: String 25 chars
Subjects(names): list(Subjects)

getters:
subjects(); //return the list of subjects that belong to the college
subject(subjectName); //return the subject given by the provided name

setters:
name(name); //update the college name

methods:
addSubject(subjectName); //create a new subject with the provided name and add it to the college

removeSubject(subjectName); //delete the subject with the provided name and remove it from the college

Boundary Classes

Schedule Maker Diagram Class

courseName : String (16 char)
courseTime : String (8 char)
weekday : String (16 char)
generateScheduleDisplay();
printSchedule();

Display Course Offerings Class

courseName : String (16 char)
weekday : String (16 char)
courseID : String (16 char)
printClassResults()
printAvailClasses()

Display Overall Course Schedule Class

role : String (8 char)
printDetailSchedule : boolean
printSchedule();
printClassDetailSchedule();
printListOfStudents();

Display Manage Account Class

role : String (8 char)
displayAccountView(role);

Manage Graduation Display Class

student : String (16 char)

Database Management Screen

generateDisplay();

Print Students List Class

studentsList : Array
printStudentsList(studentsList);

Login and Logout Class

userName : String (16 char)
password : String (16 char)
login()
saveChanges()

Calendar View Class

showMonthlyView();
showWeeklyView();
showDailyView();

<u>Control Classes</u>

Schedule Maker Class

fromScratch : boolean
studentHasCourses: boolean
Schedule : Schedule Class
coursesInterested In : String Array
possibleSchedules : Schedule Class Array

generateWeekSchedules();
getCurrentSchedule();
addClass();
deleteClass();

Course Offerings Class

course: String 16 chars
department : String 16 chars
courses : String Array // Holds Classes returned by query class.
getDepartmentCourses(); //searches classes by department
getCourse(); //searches by course

Calculate GPA Class

GPA : float (4 digits)
calculateGPA();

Database Class

query : String 64 chars
result : String 64 chars
setQuery(String);
search(query);//search database
update(); //update database

Message Class

from(Id): Integer (9 digits)
to(Id): Integer (9 digits)
message : String (64 chars)
sendMessage(to: Integer, message: String);
getMessages();

Admin Tools Class (Concrete)

role: String; //differentiate diff users using the class
methods:
manageUsers();
manageSubject();
manageCourse()
manageTerm();
manageCalendar();
manageMessages();
importApplicationData(fileName: string);

Get Student List For Course Class

courseID : String (16 char)
getStudentsListForCourse(courseID)

Authentication Class

AuthenticateUser();
showErrorMessage();

Assign Event to Calendar

event : Event
eventDate : Date
location : String (16 char)
checkIfNoClash(eventDate , location);
addEvent(Event, eventDate, location);

File Parser Utility Class (Concrete)

methods:
parseCSV(fileName); //parse the file as csv
parseXML(fileName); //parse the file as xml

3. Detailed Design

a) Method Interface Description

b) Local data Structures

c) Algorithm for the method (Pseudocode)

<u>Student</u>

Display Course Offerings Class

courseName : String (16 char)

```
weekday : String (16 char)
courseID : String (16 char)

// Generates initial display for user filled with input boxes, buttons, menus, etc.
void generateDisplay(){
    → Generate GUI
}

//Displays results from query for a class
printClassResults(){
    → From the returned query display all the classes (open and closed)
}

//Displays resutls from query for classes
printAvailClasses(){
    → From the returned query display all classes
}
```

Course Offerings Class

```
course: String 16 chars
department : String 16 chars
courses : String Array        // Holds Classes returned by query class.
//course of department must not be empty.


//sends query to database on department
getDepartmentCourses(){
    if department != empty && isValid{
            → query database for all classes in department
            → return classes in department
    }else{
            → display error message
    }
}
//queries database for a course
getCourseInfo(){
    if course != empty && isValid{
            → query database for an approximate match on the course
            → return class result
    }else{
            → display error message
    }

}
```

Database Class

```
query : String 64 chars
result : String 64 chars
setQuery( String);{
    → Set the value of the query to be used
}
//Search database
search(query){
    → Query the Database using the query
}
//Update database
update(query);
```

Course Class (Concrete)

```
nameame: String 25 chars
subject(name): String 25 chars
description: String 255 chars
prerequisite Requirements(names): Courses
creditHours: Integer 1 digit

getters:

//Return the course name
name(){
    return name
}

//Return the subject name
subject(){
    return subject
}

//Return the course description
description(){
   return description
}

//Return the prerequisite courses
prerequisites(){
   return prerequisites
}

//Return the number of credit hours
creditHours(){
   return creditHours
}

setters:
//Update the course name
name( name ){
   → Set name
}

//Update the subject membership
subject( subject ){
   → Set subject
}

//Update the description
description( description ){
   -->Set description
}

//Update the prerequisite course requirements
prerequisites( prerequisites ){
   → Set prerequisites
}

//Update the number of credit hours
```

```
creditHours ( hours ){
   → Set hours
}
```

Term Course Class (Concrete)

```
id: Integer 9 digits
name: String 25 digits
classroomLocation: String 25 digits
meetingTime: String 25 digits
classCapacity: Integer 3 digits
instructor(Id): Integer 9 digits
tA(Id): Integer 9 digits
students(Ids): list(Integer 9 digit)

getters:
//Return the term course name
name(){
   return name
}

//Return the classroom location
classroomLocation(){
   return classroomLocation
}

//Return the meeting time
meetingTime(){
   return meetingTime
}

//Return the capacity
capacity(){
   return capacity
}

//Return the instructor
instructor(){
   return instructor
}
```

```
//Return the TA
ta(){
   return tA
}

//Return the list of enrolled students
students(){
   return students
}
```

Schedule Maker Diagram Class

```
courseName : String (16 char)     //name of the course
courseTime : String (8 char)      //time of course
weekday : String (16 char)        //weekday (M-F)

//Generate initial display for schedules
generateScheduleDisplay(){
   → Create initial GUI
}

//Display the Students Schedule on the GUI
printSchedule(){
   → Get the students schedule
        → for every course display courseName, courseTime, weekday  on the GUI
}
```

Schedule Maker Class

```
fromScratch : boolean
studentHasCourses: boolean
Schedule : Schedule Class
coursesInterestedIn : String Array
possibleSchedules : Schedule Class Array


//Get students current schedule
getCurrentSchedule(){
   if students schedule != empty
         -->Set the current schedule to include courses student is enrolled for
         -->Set studentHasCourses == true
   }else{
         -->Set studentHasCourses == false
   }
}

//Generate all possible schedules for a week
generateWeekSchedules(){
   if fromScratch == true{
         -->Generate full schedule based on students major
         -->generate a new schedule using a greedy algorithm for scheduling
                 -->get courses that are part of the student's degree
                 -->add the earliest starting class to the schedule
                         -->for every class that has a time slot open after it
                         -->Add the course that starts immediately after and does not conflict
with other classes
                         -->Once 12+ credit hours are reached move on start a new schedule
   }else{
         -->get students current schedule
         -->generate a new schedule using a greedy algorithm for scheduling
                 -->get courses that are part of the student's degree
                 -->for every class that has a time slot open after it
                         -->Add the course that starts immediately after and does not conflict
with other classes
                         -->Once 12+ credit hours are reached move on start a new schedule
   }
}


//Student wants to generate a complete schedule from scratch
setFromScratch( boolean){
   -->set FromScratch  //if true the class will generate complete schedules, ignoring any
classes the student is signed up for
}

//Set current Students schedule
setSchedule(){
    for every course in the Students schedule
            -->Add it to the schedule
}

//Adds a class to the current schedule
addClass(){
    -->add class to schedule
}

//Remove a class from the current schedule
deleteClass(String){
    -->remove class from schedule
}
```

Schedule Class

```
Student: User Account Class
CoursesSignedUpFor: Term Course Class List

//Add a course to the Student's schedule
addCourse( String ){
    -->Add course to students schedule
}

//Remove a course from the Student's schedule
dropClass(String){
    -->Remove the course from the Students schedule
}

//print the students schedule
printSchedule(){
    -->print student's schedule to the GUI
}

//Using the name of a class the module will contact the professor using the Message Manager
messageProf(course : String){
    --> Send message to the course's professor
}
```

```
//Enter on a waiting list for the class
getOnWaitingList(course : String){
   -->get the course
   -->add students ID to the courses waiting list
}

//View the course's detail information
viewClassDetail(course : String){
   --print the courses detail information to GUI
}
```

Professor

```
getTheListOfStudentsForCourse(courseId : String)
{
        →(check if the role is valid to access the information)
        {
                →  display the proper error message
                → return;
        }

        → (check if the course ID exists in the database)
        {
                → if invalid id, display a proper error message
                → return;
        }
        → retrieve the list of all the students from Database.
        → return list;
}

coursesBeingTaughtByProfessor(professorId : Integer, semesterId: String)
{
        → (check if professor ID valid)
        {
                → if not, proper error message is displayed.
                → return;
        }
        → (check if semester ID valid)
        {
                → if not, proper error message is displayed.
                → return;
        }

        → Go to database and retrieve the relevant information from the course table.
        → return list;
}

contactCourseStudents (studentsList : List / Array, message : String)
{
        → The students List is passed to the method and the message to be sent is also
passed.
        → for(i = 0; i < numberOfStudents; i++)
        {
                → studentsList[i].sendMessage(message);
        }
}

contactCourseTA ( studentTAID : Integer, message : String)
{
        → (check if studentTAID valid)
        {
                → if not, proper error message is displayed.
        }
        → message is then sent to the TA.
}
```

Pseudo Codes for Teacher Assistant

```
getTheListOfStudentsForCourse(courseId : String)
{
        →(check if the role valid to access the information)
        {
                → if not, display the proper error message
                → return;
        }
        → (check if the course ID exists in the database)
        {
                → if invalid id, display a proper error message
                → return;
        }
        → retrieve the list of all the students from Database.
        → return list;
}

contactCourseStudents (studentsList : List / Array)
{
        → The students List is passed to the method and the message to be sent is also
passed.
        → for(i = 0; i < numberOfStudents; i++)
        {
                → studentsList[i].sendMessage(message);
        }
}

contactProfessor (professorID : Integer)
{
        → (check if professorID valid)
        {
                → if not, proper error message is displayed.
                → return;
        }
        → message is then sent to the TA.
}

coursesAssignedToTA (studentTAId : Integer, semesterId : String)
{
        → (check if  studentTAId valid)
        {
                → if not, proper error message is displayed.
                → return;
        }

        → (check if semesterId valid)
        {
```

```
        → if not, proper error message is displayed.
        → return;
    }

    → Go to database and retrieve the relevant information from the course table.
    → return list;
}
```

Pseudo Code for Adviser

```
removeHold(studentId: Int)
{
    → check the permission for modifying holds
    → if not successful generate error message else
        → check if studentId is valid and belongs to student else generate error
        → modify student account and remove hold
        → save id who removed hold
        → update database row

}
assignAdvisor(studentId: Int)
{
    → check the permission for modifying adviser on students account
    → validate studentId
    → update database record for adviser

}
acceptNewStudent(studentId: Int)
{
    → check the permission for modifying student account
    → validate if studentId is already active
    → update database info about new student

}
```

Events Class

```
assignAnEventOnDate(event: Event, date:DateTime, location:String)
{
    → (isDateValid(date))
    {
        → (checkIfNoClash(date,location))
        {
            → assign the event at the date or location
        }
        else
        {
            → proper error message is displayed
        }
    }
    else
    {
        → proper error message is displayed
    }

}

isDateValid(date: DateTime)
{
    → check time. Event time must be between particular time range.
    → if time falls out of the range, return false
    → check day. e.g. If weekend, return false
    → else return true.
}

checkIfNoClash(date : DateTime, location : String)
{
    → send the date and location to the database to determine whether any event exists at
    that particular slot
    → return true or false accordingly
}
```

Account Controller Class

```
modifyUser( id: Integer, name: String, surname: String, address: String, phone: integer,
email: String, role: String ) //modify account information of existing user
{
    → If the id doesn't match the caller's id and the caller is not an admin
        → return an error.
        → else if id is invalid
                → return an error.
        → else
                → update the user id with the provided name, surname, address, phone, and
        email
                → if the caller is an admin
                        → update the user id with the provided role
}
```

```
createUser( name: String, surname: String, address: String, phone: integer, email: String,
role: String ) //create a new account
{
        → if the caller is not an admin
                → return an error
        → else
                → create a new user account with the provided name, surname, phone,
email, role, and random id.
}
```

```
deleteUser( id: Integer ) //delete existing user
{
        → if the caller is not an admin
                → return an error
        → else if the provided is invalid
                → return an error
        → else
                → delete the account associated with the provided id
}
```

Subject Controller Class

```
modifySubject( name: String, college: String, department: String, courses: list(Courses) )
//modify an existing subject
{
        → if the caller is not an admin
                → return an error
        → if the subject name doesn’t match an existing subject
                → return an error
        → else
        → update the named subject with the provided college, department, and
course list
}

createSubject( name: String, college: String, department: String, courses: list(Courses) )
//create a new subject
{
    → if the caller is not an admin
        → return an error
    → else
        → create a new subject with the provided name, college, department, and
course list
}
deleteSubject (name: String) //delete an existing subject
{
    → if the caller is not an admin
        → return an error
    → if the subject name doesn't match an existing subject
        → return an error
    → else
        → delete the subject that matches the provided name
}
```

Course Controller Class

```
modifyCourse( name: String, subject: String, description: String, prerequisite requirements:
list(Courses), credit hours ) //modify an existing course
{
    → if the caller is not an admin
        → return an error
    → if the course name doesn't match an existing course
        → return an error
    → else
        → update the named course with the subject, description, prerequisite
requirements, course list, and credit hours
}

createCourse( name: String, subject: String, description: String, prerequisite requirements:
list(Courses), credit hours ) //modify an existing course
```

```
{
    → if the caller is not an admin
        → return an error
    → else
        → create a new course with the provided name, description, prerequisite
requirements, course list, and credit hours under the specified subject
}
```

```
deleteCourse( name: String ) //delete an existing course
{
    → if the caller is not an admin
        → return an error
    → else
        → delete the course provided by the name parameter
}
```

File Parser Utility Class

```
parseCSV( fileName: String ) //parse the csv file and insert the data into the database
{
    → fileHandle := open( fileName )
    → if fileHandle null
        → return an error
    → else if fileName extension is .csv
        → line := null
        → data := null
        → while( line := fileHandle.nextLine() != null )
            → data := line.split(‘,’)
            → update database with the data
    → else if fileName extension is .xml
        → invoke the Qt XML module to extract attribute values and update the
database
}
```

Administrator Controller Class

```
manageUsers( actionEvent ) //display the manage users window
{
    → if the user is an admin.
```

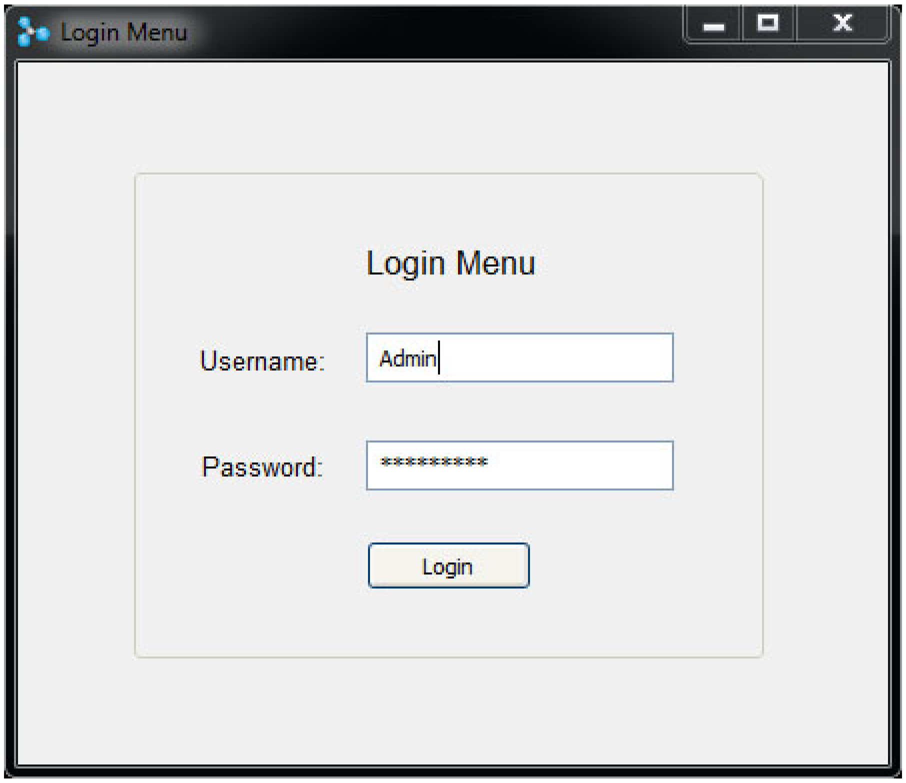
```
        → query the database for all user accounts and display them in a list to allow
account management (modification, deletion, and creation)
    → else return error.
}

manageSubjects() //display the manage subjects window
{
    → if the user is an admin.
        → query the database for all subjects and display them in a list to allow
subject management (modification, deletion, and creation).
    → else return error.
}

manageCourses() //display the manage courses window
{
    → if the user is an admin
        → query the database for all courses and display them in a list to allow
course management( modification, deletion, and creation).
    → else return error.
}

manageTerms() //display the manage terms window
{
    → if the user is an admin
        → query the database for all terms and display them in a list to allow term
management( modification, deletion, and creation).
    → else return error.
}

manageCalendar() //display the manage calendar window
{
    → if the user is an admin
        → query the database for all calendar events and mark them on the calendar.
Allow calendar management( modification, deletion, and creation).
    → else return error.
}

manageMessages() //display the manage messages window
{
        → if the user is an admin
                → query the database for messages and display them in a list to allow
message management( deletion and creation).
        → else return error.
}

importApplicationData(fileName: String)
{
        → if the user is an admin
                → prompt the user for an import file
                → if the extension of provided file is .csv
                        → call method parseCSV( fileName ) of class fileParserUtility
                → else if file extension is .xml
                        → call method parseXML( fileName ) of class fileParserUtility
                → else return error
}
```

4. User Interface Design

a) Description

A rapid prototype was created to show the GUI display look and feel. We show the GUI of only the administrator role, which has the greatest level of functionality. Although screens of the remaining roles are not shown, their functionality is a subset of the administrative role and, therefore, their UI's will be similar except with reduced information and available actions. The general rule of each screen is to allow the user to return back to the previous window, which is allowed by the top action button on most screens except for on the login and main menu.

b) Screen images



login menu

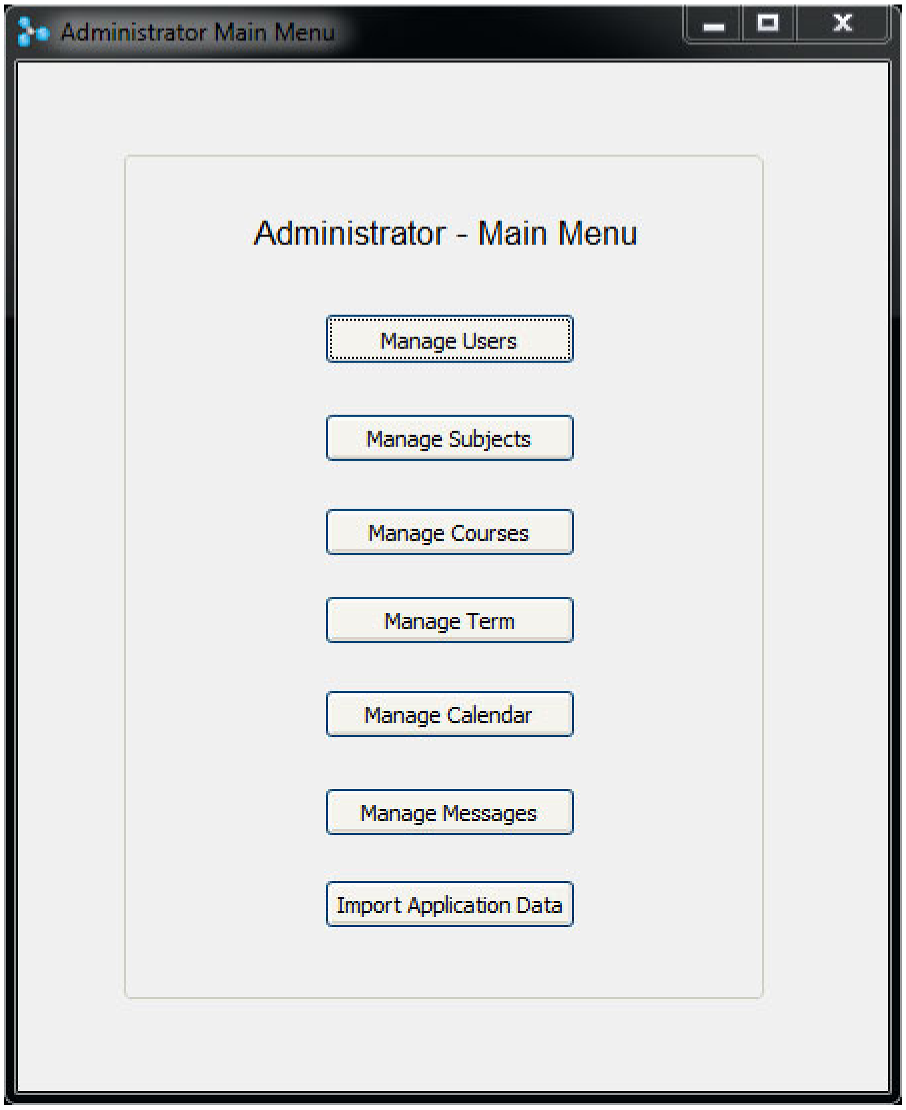
1. Description

The *Login Menu* solicits the username and password used for authenticating the user to her account.

2. Objects and Actions

- Main input boxes are displayed in center
    - Username Input box - Allows user to input their user name
    - Password Input box - Allows the user to input their password
- Action button is presented at the bottom
    - Login button - Application will authenticate the user and allow them to use the application

---



administrator main menu

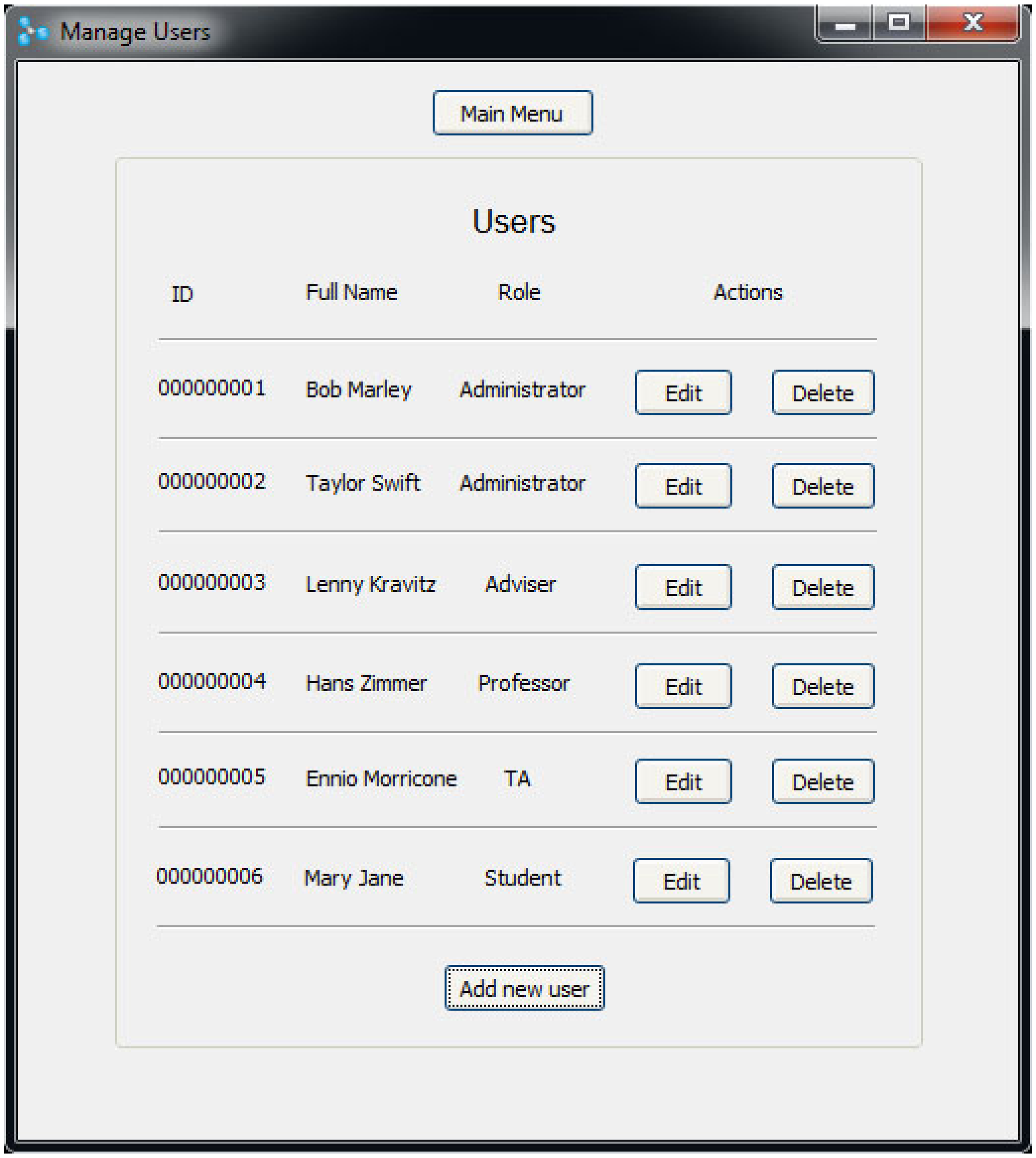
1. Description

The main menu (shown here of an administrator) displays all the top level options available to the logged in user. The menu consists of action buttons, which bring up the appropriate screen when pressed. When at a further nested screen, the user is allowed to navigate backwards to this menu through a sequence of back buttons.

2. Objects and Actions

- A very straightforward menu is displayed down the center of the screen. The actions performed by the buttons are intuitively labeled.
  - Manage Users button
  - Manage Subjects button

- Manage Courses button
- Manage Term button
- Manage Calendar button
- Manage Messages button
- Import Application Data button



Users

ID	Full Name	Role	Actions	
000000001	Bob Marley	Administrator	Edit	Delete
000000002	Taylor Swift	Administrator	Edit	Delete
000000003	Lenny Kravitz	Adviser	Edit	Delete
000000004	Hans Zimmer	Professor	Edit	Delete
000000005	Ennio Morricone	TA	Edit	Delete
000000006	Mary Jane	Student	Edit	Delete

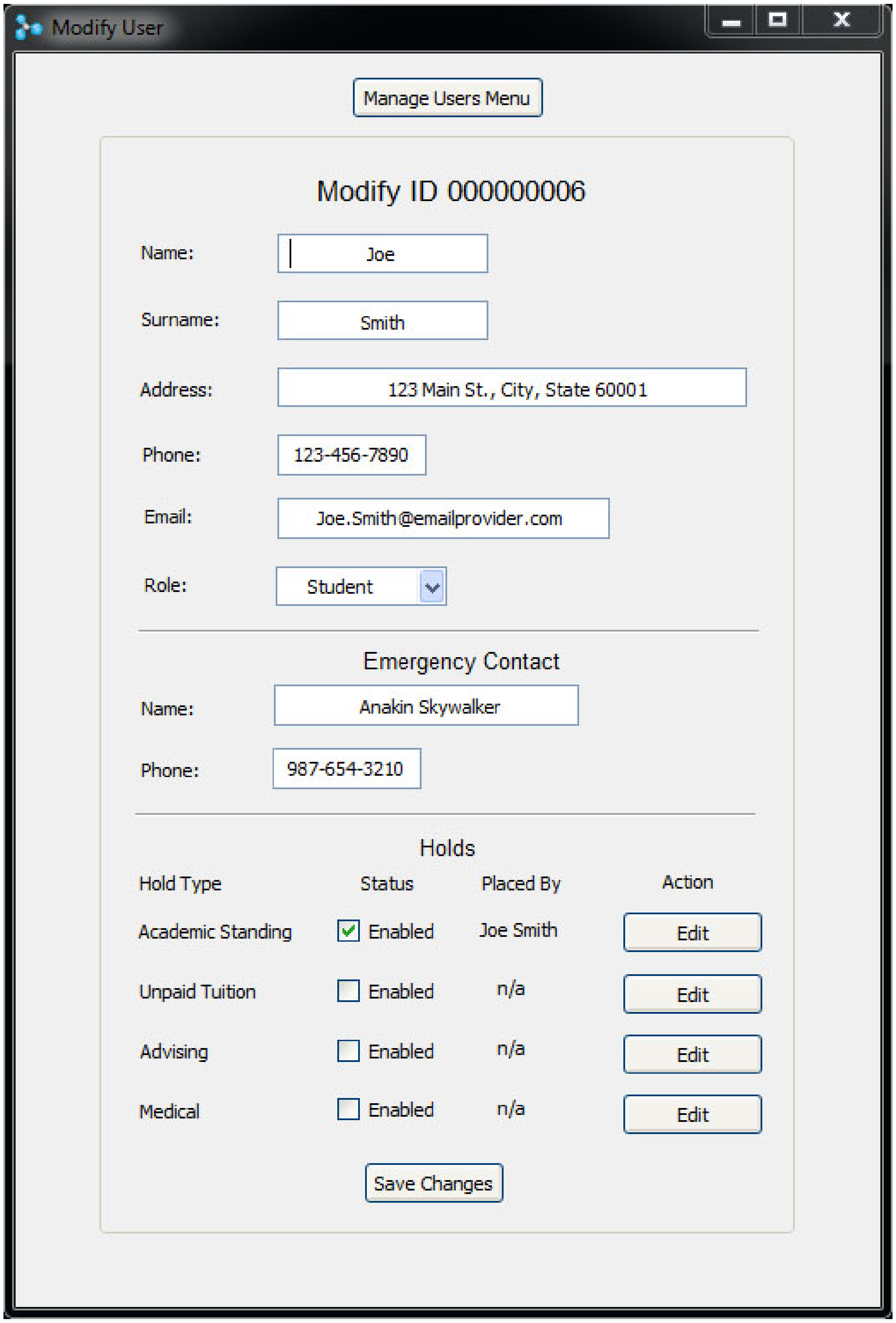
manage users

1. Description

The *manage users* screen is visible only by the administrator. It displays all of the user accounts in a row orderly fashion with the column attributes showing the id, full name, and role. Each row also contains action buttons to allow the user's information to be modified or for the user to be deleted from the system.

2. Objects and Actions

- Navigation controls are given on the top of the screen
    - Main Menu button - returns the user to the Administrator main menu.
- Main content is displayed on the center of the screen
    - First column shows the User's ID
    - Second column shows User's Full Name
    - Third column displays Users Role
    - Fourth column holds the action buttons
        - Edit button - allows the Admin to edit a user
        - Delete button - deletes a user from the system
- The bottom holds additional function buttons
    - Add New User button- allows new users to be added to the system.



Modify User

Manage Users Menu

Modify ID 000000006

Name:	Joe
Surname:	Smith
Address:	123 Main St., City, State 60001
Phone:	123-456-7890
Email:	Joe.Smith@emailprovider.com
Role:	Student

Emergency Contact

Name:	Anakin Skywalker
Phone:	987-654-3210

Holds

Hold Type	Status	Placed By	Action
Academic Standing	✓ Enabled	Joe Smith	Edit
Unpaid Tuition	☐ Enabled	n/a	Edit
Advising	☐ Enabled	n/a	Edit
Medical	☐ Enabled	n/a	Edit

Save Changes

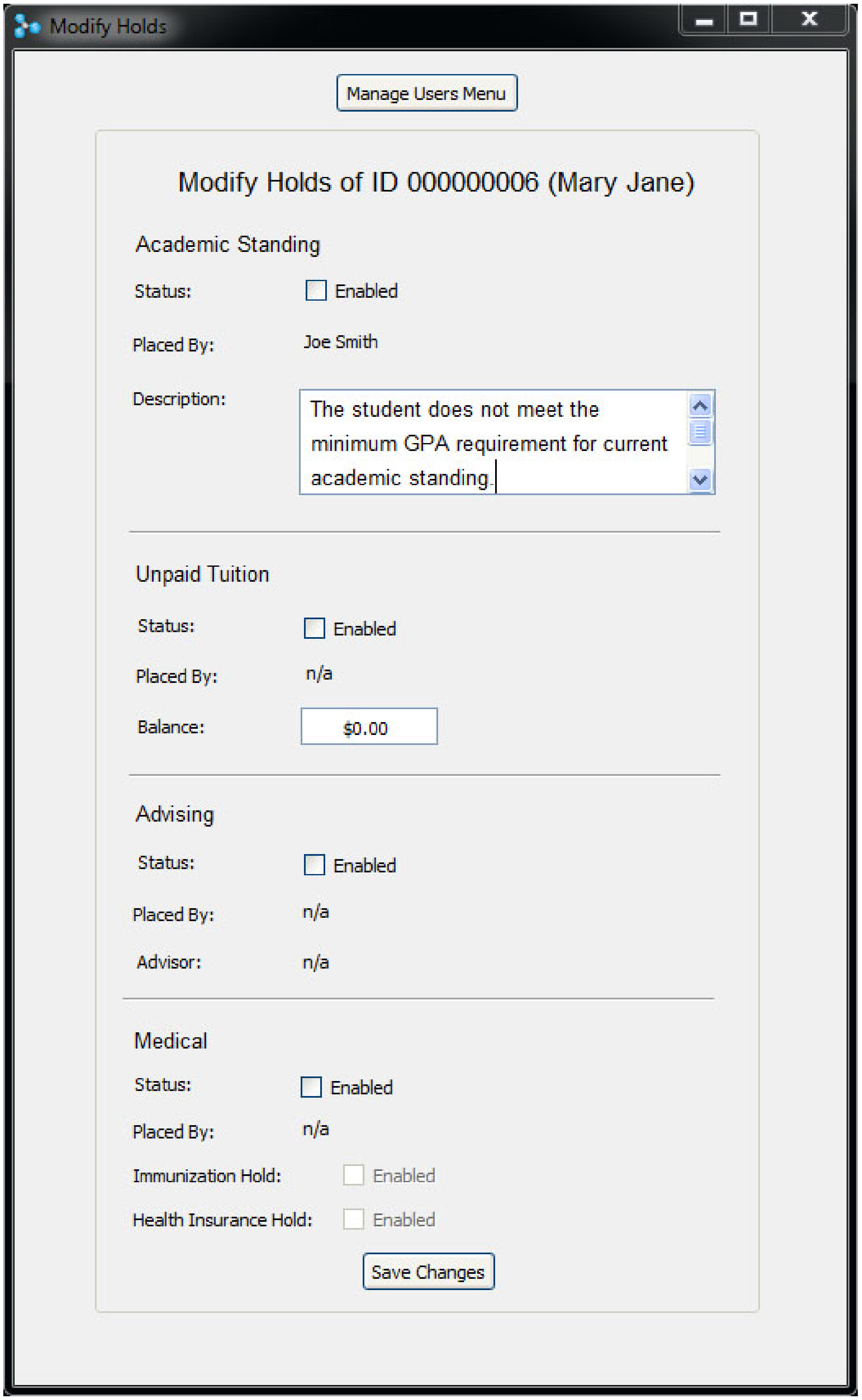
modify user

1. Description

The *Modify User* screen shows the user account information that is modifiable. The administrator is allowed to modify all information of all users in the system and can also elevate or demote the user's role by selecting an option from the select box. In addition, only the administrator is allowed to add or remove holds on any student account, whereas the adviser can add or remove holds only from her student advisees. The modify user screen is similar for the non privileged users, who are limited to modifying only their own information and are prevented from changing their roles and hold status.

2. Objects and Actions

- Top section of the screen provides navigation controls
    - Manage Users Menu button- returns administrator back to the user management screen. All other non privileged users will instead see a button to return back to the main menu.
- First section displays the user's basic information
    - Several input boxes will display basic information. Certain boxes will not be editable given the User's permissions. Input boxes are fairly self explanatory.
        - Name
        - Surname
        - Address
        - Phone
        - Email
        - Role
- Second section shows the Emergency contact information for the User
    - Input boxes are active for editing depending on the User permissions. Information is self explanatory.
        - Name
        - Phone
- Third Section shows the holds on a user in table format
    - First column shows the type of hold
    - Second column allows the admin to activate/deactivate the hold via a checkbox
    - Third column shows who activated the hold on the account
    - Fourth column holds the control button
        - Edit button - Allows the Admin to edit the hold



Modify Holds of ID 000000006 (Mary Jane)

Academic Standing

Status:	☐ Enabled
Placed By:	Joe Smith
Description:	The student does not meet the minimum GPA requirement for current academic standing.

Unpaid Tuition

Status:	☐ Enabled
Placed By:	n/a
Balance:	$0.00

Advising

Status:	☐ Enabled
Placed By:	n/a
Advisor:	n/a

Medical

Status:	☐ Enabled
Placed By:	n/a
Immunization Hold:	☐ Enabled
Health Insurance Hold:	☐ Enabled

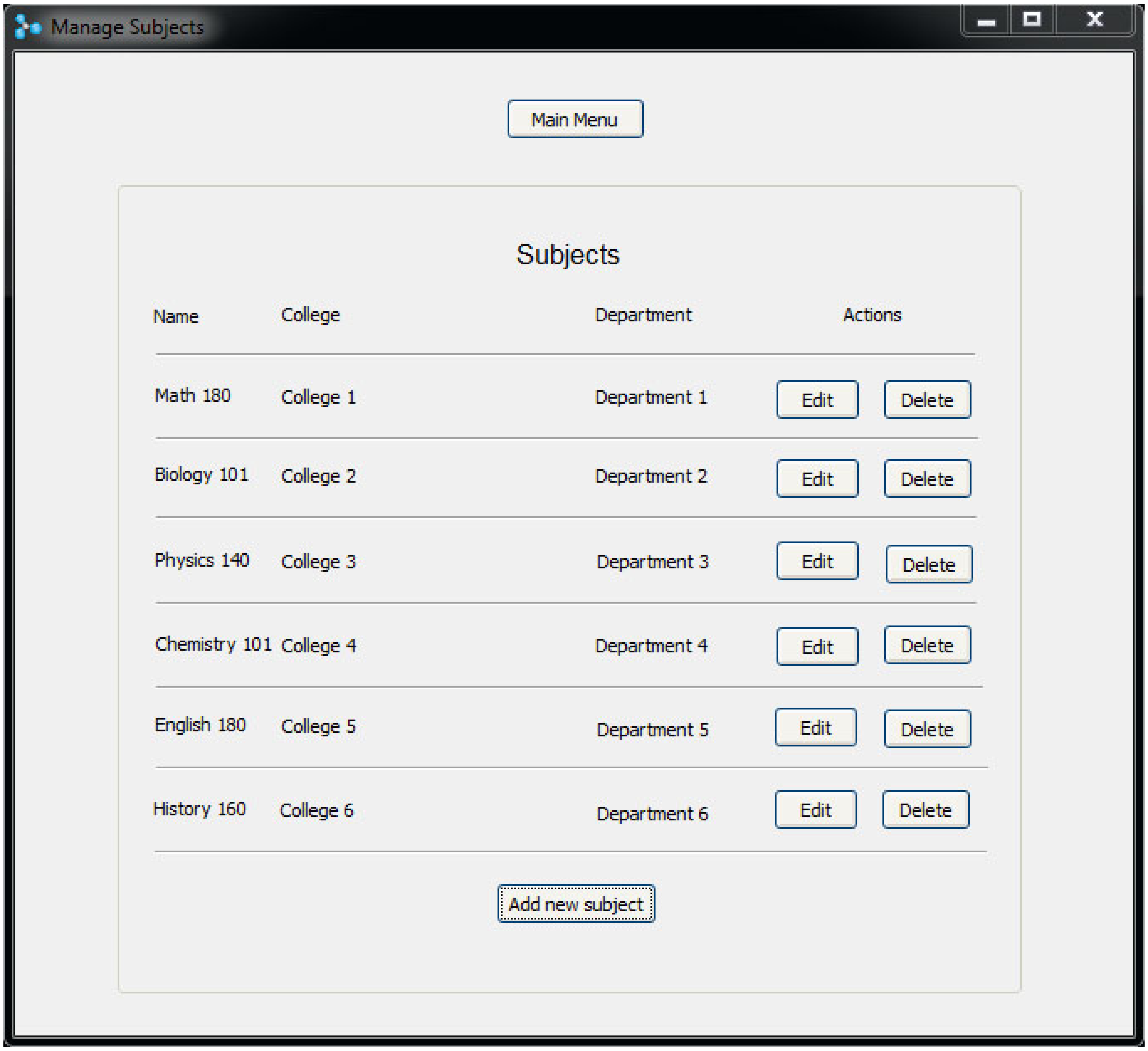
modify holds

1. Description

The modify holds screen allows administrators and advisers to make changes to existing holds on student accounts. The screen displays the user id and name for whom the hold is being modified. There are four group boxes that group the attributes of the four different kinds of holds together. Each hold shows the status of the hold (whether enabled or disabled) and the name of the user who placed it as well as additional input fields for the appropriate attributes.

2. Objects and Actions

The “Save Changes” button commits the changes to the system, and the “Manage Users Menu” button returns the admin back to a list of all users. An adviser instead would see a “Main menu” button for navigating back to the main menu.



Subjects

Name	College	Department	Actions	
Math 180	College 1	Department 1	Edit	Delete
Biology 101	College 2	Department 2	Edit	Delete
Physics 140	College 3	Department 3	Edit	Delete
Chemistry 101	College 4	Department 4	Edit	Delete
English 180	College 5	Department 5	Edit	Delete
History 160	College 6	Department 6	Edit	Delete

Manage Subjects

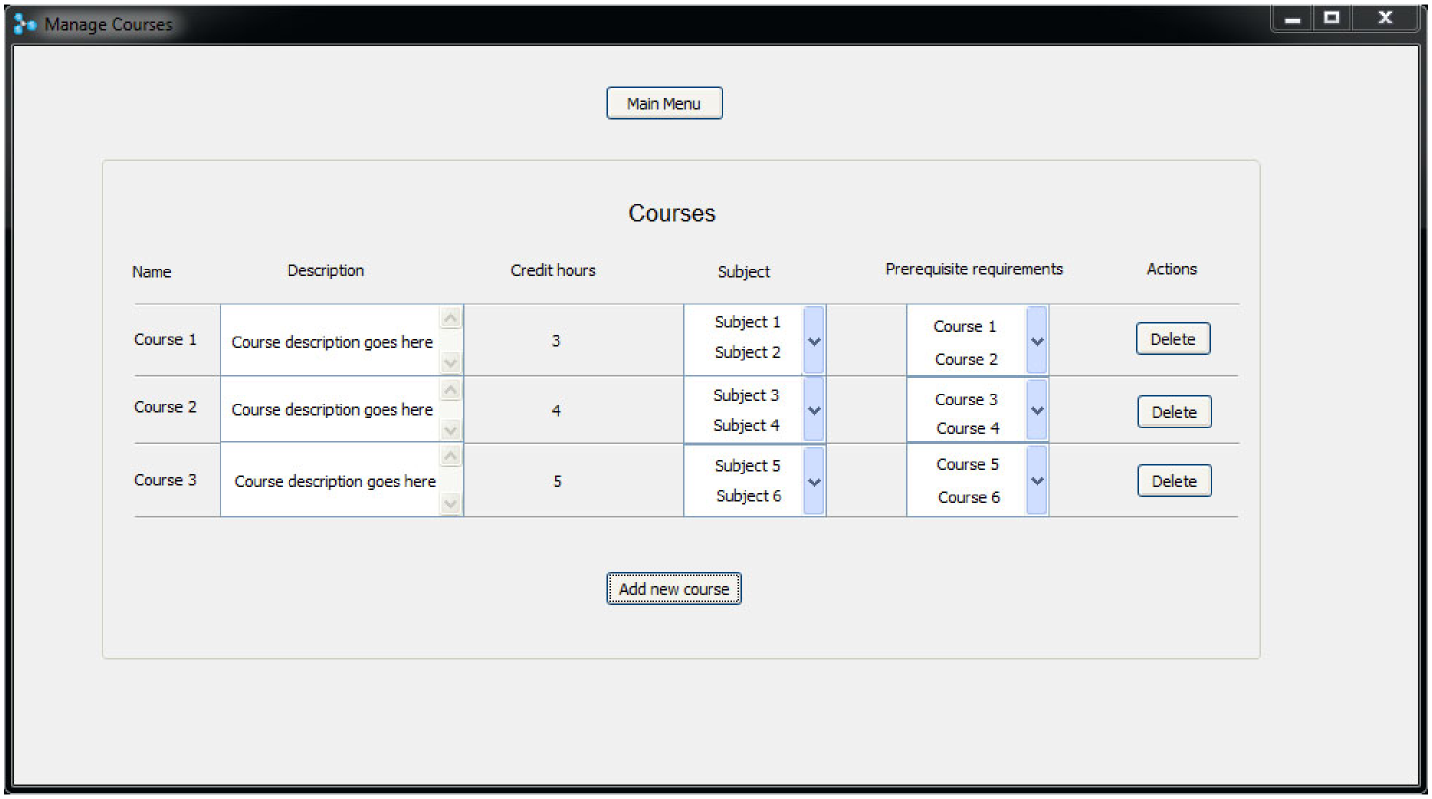
1. Description

The Manage Subjects screen shows a list of the subjects and the controls for the Admin to be able to edit them.

This screen shows the same general design rules as most screens. The top contains navigation buttons, the center displays the main content, and the bottom displays additional control buttons.

2. Objects and Actions

- Navigation is presented at the top of the screen
    - Main Menu button - takes user back to the Main Menu screen
- Table for main content is present on center of the screen
    - First column holds the course’s name
    - Second column holds the college the course belongs to
    - Third column shows the department the course belongs to
    - Fourth column holds action buttons
        - Edit button - allows the user to edit the subject’s information
        - Delete button - deletes a course from the list
- At the bottom of the screen additional functionality buttons and menus are displayed
    - Add New Subject button - Adds a new subject to the table



Courses

Name	Description	Credit hours	Subject	Prerequisite requirements	Actions
Course 1	Course description goes here	3	Subject 1 Subject 2	Course 1 Course 2	Delete
Course 2	Course description goes here	4	Subject 3 Subject 4	Course 3 Course 4	Delete
Course 3	Course description goes here	5	Subject 5 Subject 6	Course 5 Course 6	Delete

Manage Courses

1. Description

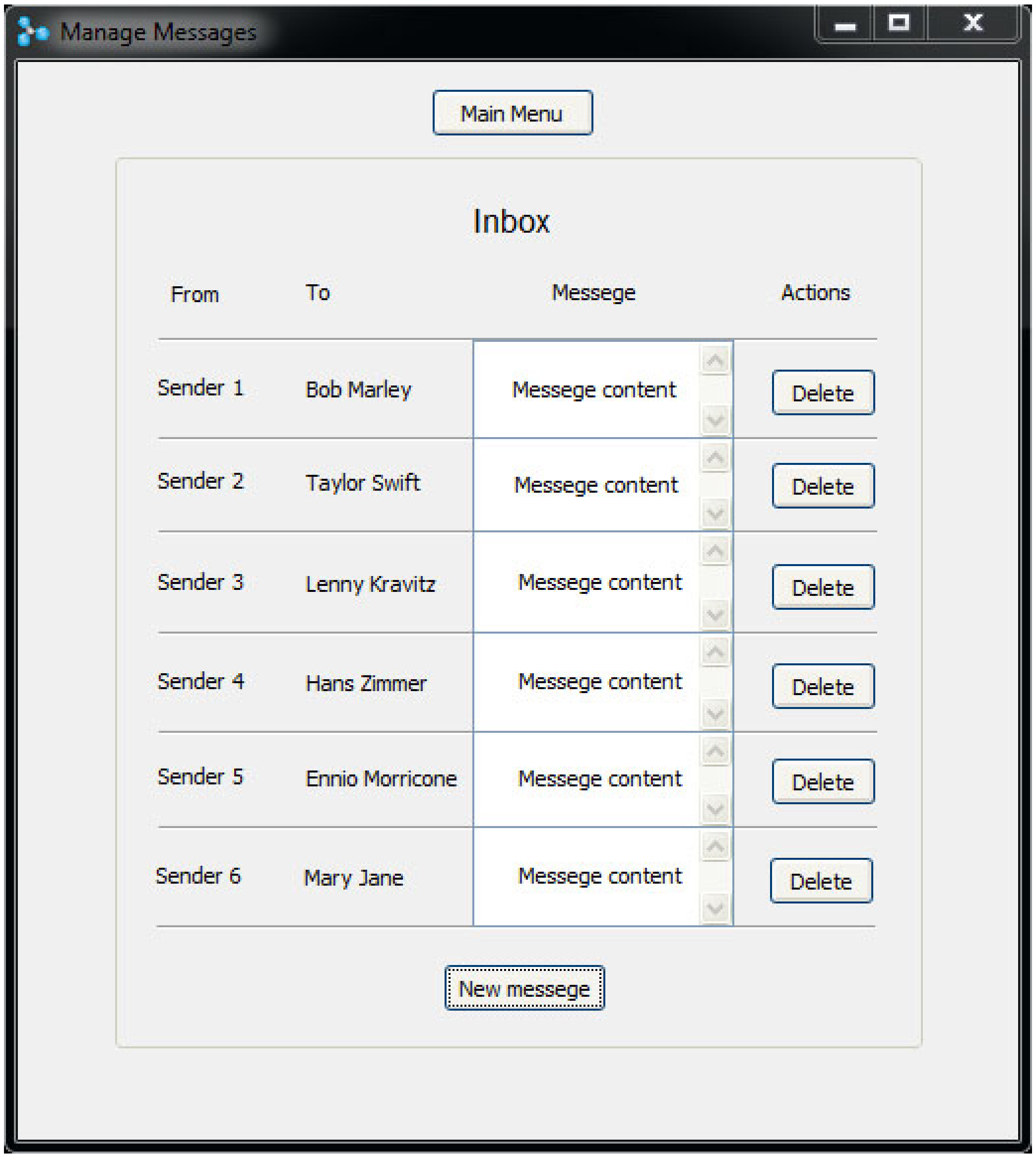
The Manage Courses screen displays the various attributes a course has as well as offers a couple of management tools. It allows the addition of a course, its basic information, and prerequisites.

This structure will be very similar to the one used to display/edit courses, schedules, and degree progress. The main information will be presented in a clear and easy to read table structure. Main information will be on the left hand side while most of the action buttons/menus will be to the right of the information. The table will expand according to the amount of information on screen.

Again, at the top a simple navigation menu will be given to the user. The main focus will be to give the user a way back to the main menu. At the bottom of the screen any additional buttons providing further functionality will be presented to the user.

2. Objects and Actions

- Navigation is presented at the top of the screen
    - Main Menu button - takes user back to the Main Menu screen
- Table for main content is present on center of the screen
    - First column holds the course's name
    - Second column holds the course description
    - Third column shows the credit hours the course is worth
    - Fourth column displays the Subject the course belongs to
    - Fifth column shows the prerequisite requirements needed to take the course
        - A drop down list is shown to indicate that there are more courses than are displayed. This is a design choice to keep the table concise and easy to read.
    - Sixth column holds action buttons
        - Delete button - deletes a course from the list
- At the bottom of the screen additional functionality buttons and menus are displayed
    - Add New Course button - Adds a new course to the table



From	To	Messege	Actions
Sender 1	Bob Marley	Messege content	Delete
Sender 2	Taylor Swift	Messege content	Delete
Sender 3	Lenny Kravitz	Messege content	Delete
Sender 4	Hans Zimmer	Messege content	Delete
Sender 5	Ennio Morricone	Messege content	Delete
Sender 6	Mary Jane	Messege content	Delete

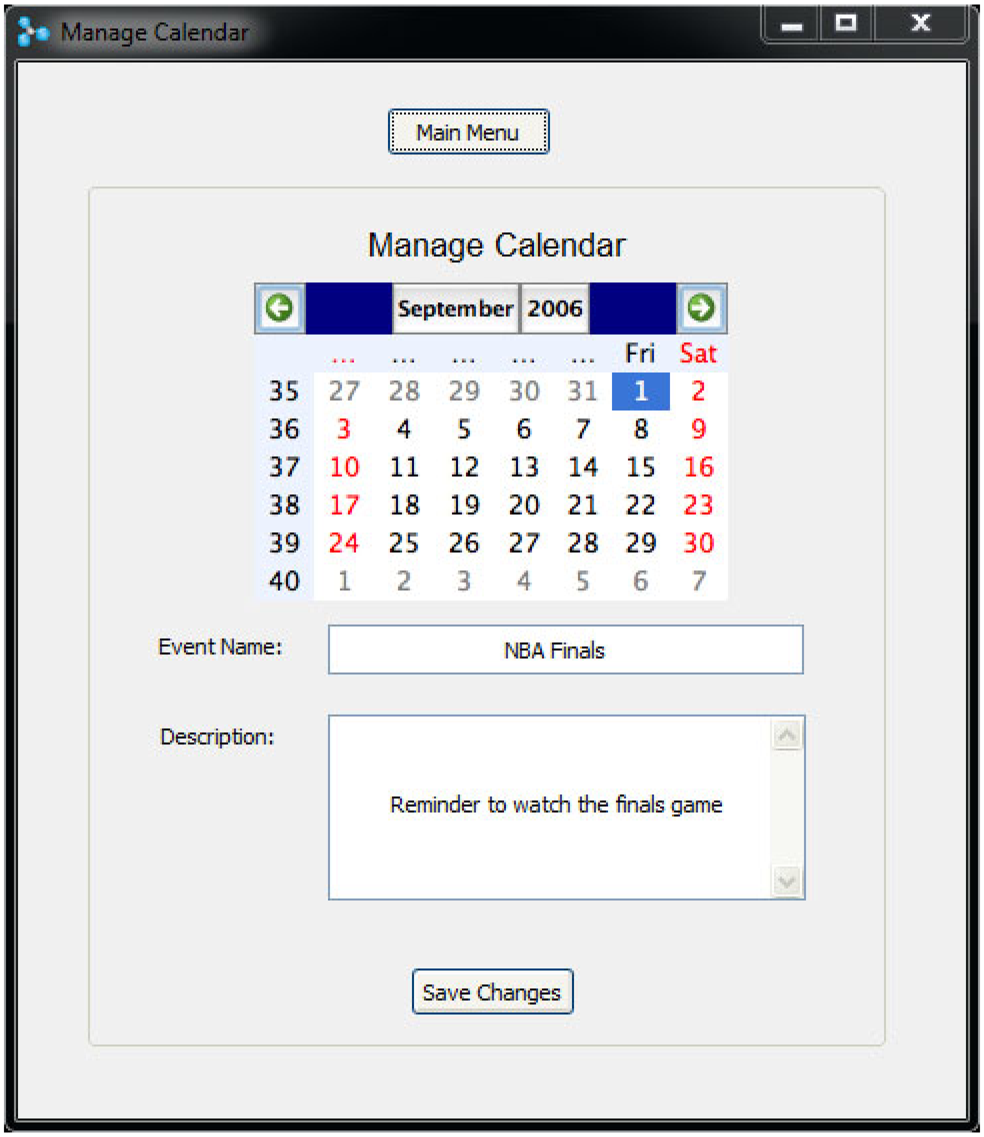
Manage Messages

1. Description

The Manage Messages screen displays a very straightforward list of the messages received by the user. The messaging system is set to be a very basic method of communication for sending brief messages. The standard for displaying lists will be having a table like structure with the main content being in the first couple columns and action buttons being to the right of the displayed data. The screen depicts all messages present in the system to the administrator (as shown in the figure), however, an unprivileged user will only see messages sent or received by him or her.

2. Objects and Actions

- Navigation controls are at the top of the screen
  - Main Menu Button - returns the user to the Main Menu screen
- Table for main content is present on center of the screen
  - The first column holds the sender of the message
  - The second column holds the receiver of the message
  - The third column holds the message itself
  - The fourth column holds the action button
    - Delete button - deletes current message.
- At the bottom of the screen additional functionality buttons and menus are displayed
  - New Message Button - creates a new message to be sent out.



	...	...	...	...	...	Fri	Sat
35	27	28	29	30	31	1	2
36	3	4	5	6	7	8	9
37	10	11	12	13	14	15	16
38	17	18	19	20	21	22	23
39	24	25	26	27	28	29	30
40	1	2	3	4	5	6	7

Event Name: NBA Finals

Description: Reminder to watch the finals game

1. Description

The Manage Calendar screen allows the Admin to add/edit event on the Calendar. GUIs that use the Calendar will be very similar in layout with the calendar being top center.

2. Objects and Actions

- Navigation controls are at the top of the screen
    - Main Menu Button - returns the user to the Main Menu screen
- Calendar is present on top center of the screen
- Event Name input box - Allows the Admin to add an event
- Description input box - Allows the admin to add a description to the event
- At the bottom of the screen additional functionality buttons
- Save button - Saves changes to the calendar

5. Appendices

a) Requirements Traceability Matrix

In order to test every item in the design back to the requirements specification, we have made a traceability matrix in the shape of tables split up along the five actors (user roles) in the system. For each user role, the columns represent the original set of requirement for that particular role and the rows are the classes that were extracted during requirements phase and completed in this design document. When a requirement of a particular actor matches a class, we say there is traceability and mark this reference with an X in the intersecting cell. Thus, every part of the design is linked to some part of the requirements specification, i.e., each column is cross linked to at least one row.

Student

	Modify Personal Info.	Look up Classes	Register for a Class	Drop a Class	View Class Sched.	Generate Sched.	Contact Prof.	View Holds
Schedule Maker Class			X	X		X		
Course Offerings Class		X						

Calculate GPA Class								
Database Class	X	X	X					
Message Manager Class							X	
Admin Tools Class	X							
Get Students List for Courses Class								
Assign Event To Calendar								
File Parser Class								
Schedule Maker Diagram Class			X	X				
Display Course Offerings Class		X						
Display Overall Course Schedule Class					X		X	X
Display Manage Account Class	X							X
Manage Graduation Display Class								
Database Manage								

ment Screen								
Print Students List Class								
Calendar View Class							x	

	See Classes That are Full	Get on Waiting List for Class	View Class Detail	View Graduati on Audit	Apply for Graduati on	Receive Message s	Print Schedule of Classes	View Status
Schedule Maker Class	X	X						
Course Offerings Class	X							
Calculate GPA Class				X				
Database Class		X		X				
Message Manager Class						X		
Admin Tools Class								
Get Students List for Courses Class								
Assign Event To Calendar								
File Parser Class								

Schedule Maker Diagram Class		X						
Display Course Offerings Class	X	X						
Display Overall Course Schedule Class			X				X	
Display Manage Account Class						X		X
Manage Graduation Display Class				X				
Database Management Screen								
Print Students List Class								
Calendar View Class								

Professor

	Modify Personal Info.	See Courses Teaching Current Term	See Courses Taught Prev.	See List of Students Taking Course	Send/ Receive Messages	Contact TA
Schedule						

Maker Class						
Course Offerings Class		x	x			
Calculate GPA Class						
Database Class	x		x	x		
Message Manager Class					x	x
Admin Tools Class	X					
Get Students List for Courses Class				x		
Assign Event To Calendar						
File Parser Class						
Schedule Maker Diagram Class						
Display Course Offerings Class			x			
Display Overall Course Schedule Class						
Display Manage Account						

Class						
Manage Graduation Display Class						
Database Management Screen						
Print Students List Class						
Calendar View Class						

Teacher Assistant

	Modify Personal Info	See Courses he is Assisting This Term	See Previous Assignments as TA	See List of Students taking His Assigned Course	Send /Receive Messages
Schedule Maker Class					
Course Offerings Class		x	x		
Calculate GPA Class					
Database Class	x				
Message Manager Class					x
Admin Tools Class	x				
Get Students List for Courses Class			x	x	
Assign Event To					

Calendar					
File Parser Class					
Schedule Maker Diagram Class					
Display Course Offerings Class					
Display Overall Course Schedule Class		x			
Display Manage Account Class					
Manage Graduation Display Class					
Database Management Screen					
Print Students List Class			x	x	
Calendar View Class					

Adviser/ Counselor

	Pick Student To Advise	Change Students Name	View Students Class Sched.	View Students Progres s	Override Class Capacity	Create Appt.	Modify Appt.	Check Grad. Reqs.	Clear Adv. Hold s	Approve Self Registr ation
Schedule Maker Class										
Course Offerings Class										

Calculate GPA Class				x						
Database Class	x		x	x	x	x	x	x	x	
Message Manager Class						x				
Admin Tools Class		x								x
Get Students List for Courses Class										
Assign Event To Calendar						x	x			
File Parser Class										
Schedule Maker Diagram Class										
Display Course Offerings Class										
Display Overall Course Schedule Class										
Display Manage Account Class	x									

Manage Graduation Display Class								x		
Database Management Screen										
Print Students List Class										
Calendar View Class							x			

Administrator

	Manage User Account	Add/ Remove Holds	Manage Subject	Manage Course	Manage Term	Import Application Data	Manage Calendar	Send Receive Messages
Schedule Maker Class				x				
Course Offerings Class					X			
Calculate GPA Class								
Database Class	X	X	X	X	X	X	X	X
Message Manager Class								X
Admin Tools Class	X	X	X	X	X	X	X	X
Get Students List for Courses Class								
Assign Event To							X	
Calendar							X	

File Parser Class						X		
Schedule Maker Diagram Class								
Display Course Offerings Class								
Display Overall Course Schedule Class								
Display Manage Account Class	X							
Manage Graduation Display Class								
Database Management Screen								
Print Students List Class								
Calendar View Class							X	

b) Supplementary information (as required)

Database schemas

```
--
-- Table structure for table `building`
--

CREATE TABLE IF NOT EXISTS `building` (
  `id_building` int(11) NOT NULL AUTO_INCREMENT,
  `name` text COLLATE utf8_unicode_ci NOT NULL,
  `location` int(11) NOT NULL,
  PRIMARY KEY (`id_building`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci
```

```
AUTO_INCREMENT=1 ;
```

```
--
-- Table structure for table `class`
--

CREATE TABLE IF NOT EXISTS `class` (
  `id_class` int(11) NOT NULL AUTO_INCREMENT,
  `type` int(11) NOT NULL,
  `number` int(11) NOT NULL,
  `credit` int(1) NOT NULL,
  PRIMARY KEY (`id_class`)
) ENGINE=MyISAM  DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci
AUTO_INCREMENT=3 ;

--
-- Dumping data for table `class`
--

INSERT INTO `class` (`id_class`, `type`, `number`, `credit`) VALUES
(1, 1, 440, 3);
```

```
--
-- Table structure for table `classroom`
--

CREATE TABLE IF NOT EXISTS `classroom` (
  `id_classroom` int(11) NOT NULL,
  `room` text COLLATE utf8_unicode_ci NOT NULL,
  `building` int(11) NOT NULL,
  `capacity` int(3) NOT NULL,
  PRIMARY KEY (`id_classroom`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci;
```

```
--
-- Table structure for table `class_type`
--
```

```
CREATE TABLE IF NOT EXISTS `class_type` (
  `id_class_type` int(2) NOT NULL AUTO_INCREMENT,
  `name` varchar(4) COLLATE utf8_unicode_ci NOT NULL,
  `fullname` text COLLATE utf8_unicode_ci NOT NULL,
  PRIMARY KEY (`id_class_type`)
) ENGINE=MyISAM  DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci
AUTO_INCREMENT=5 ;

--
-- Dumping data for table `class_type`
--

INSERT INTO `class_type` (`id_class_type`, `name`, `fullname`) VALUES
(1, 'CS', 'Computer Science'),
(2, 'HIST', 'History');
```

```
--
-- Table structure for table `faculty`
--

CREATE TABLE IF NOT EXISTS `faculty` (
  `id_faculty` int(11) NOT NULL AUTO_INCREMENT,
  `type` int(11) NOT NULL,
  `firstname` varchar(40) COLLATE utf8_unicode_ci NOT NULL,
  `lastname` varchar(40) COLLATE utf8_unicode_ci NOT NULL,
  `address1` text COLLATE utf8_unicode_ci NOT NULL,
  `address2` text COLLATE utf8_unicode_ci NOT NULL,
  `city` text COLLATE utf8_unicode_ci NOT NULL,
  `zip` int(5) NOT NULL,
  `state` int(11) NOT NULL,
  `dob` date NOT NULL,
  `ss` int(9) NOT NULL,
  PRIMARY KEY (`id_faculty`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci
AUTO_INCREMENT=1 ;
```

```
--
-- Table structure for table `faculty_type`
```

```
--

CREATE TABLE IF NOT EXISTS `faculty_type` (
  `id_faculty_role` int(2) NOT NULL AUTO_INCREMENT,
  `name` varchar(80) COLLATE utf8_unicode_ci NOT NULL,
  `description` text COLLATE utf8_unicode_ci NOT NULL,
  PRIMARY KEY (`id_faculty_role`)
) ENGINE=MyISAM  DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci
AUTO_INCREMENT=5 ;

--
-- Dumping data for table `faculty_type`
--

INSERT INTO `faculty_type` (`id_faculty_role`, `name`, `description`) VALUES
(1, 'Teacher', ''),
(2, 'Adviser', ''),
(3, 'Teacher Assistant', ''),
(4, 'Administrator', '');
```

```
--
-- Table structure for table `major`
--

CREATE TABLE IF NOT EXISTS `major` (
  `id_major` int(11) NOT NULL AUTO_INCREMENT,
  `name` text COLLATE utf8_unicode_ci NOT NULL,
  PRIMARY KEY (`id_major`)
) ENGINE=MyISAM  DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci
AUTO_INCREMENT=4 ;

--
-- Dumping data for table `major`
--

INSERT INTO `major` (`id_major`, `name`) VALUES
(1, 'Computer Science'),
(2, 'Anthropology '),
(3, 'Chemistry ');
```

```
--
-- Table structure for table `message`
--

CREATE TABLE IF NOT EXISTS `message` (
  `id_message` int(11) NOT NULL,
  `from` int(11) NOT NULL,
  `to` int(11) NOT NULL,
  `topic` varchar(50) COLLATE utf8_unicode_ci NOT NULL,
  `text` text COLLATE utf8_unicode_ci NOT NULL,
  `senddate` date NOT NULL,
  `readdate` int(11) NOT NULL
) ENGINE=MyISAM DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci;
```

```
--
-- Table structure for table `semester`
--

CREATE TABLE IF NOT EXISTS `semester` (
  `id_semester` int(11) NOT NULL AUTO_INCREMENT,
  `name` text COLLATE utf8_unicode_ci NOT NULL,
  `year` int(4) NOT NULL,
  `class` int(11) NOT NULL,
  `classroom` int(11) NOT NULL,
  `teacher` int(11) NOT NULL,
  `ta` int(11) NOT NULL,
  PRIMARY KEY (`id_semester`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci
AUTO_INCREMENT=1 ;
```

```
--
-- Table structure for table `state`
--

CREATE TABLE IF NOT EXISTS `state` (
  `id_state` int(11) NOT NULL AUTO_INCREMENT,
  `name` text COLLATE utf8_unicode_ci NOT NULL,
  PRIMARY KEY (`id_state`)
) ENGINE=MyISAM  DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci
AUTO_INCREMENT=55 ;

--
-- Dumping data for table `state`
--

INSERT INTO `state` (`id_state`, `name`) VALUES
(1, 'Alabama'),
(2, 'Alaska'),
(3, 'Arizona'),
(4, 'Arkansas'),
(5, 'California'),
(6, 'Colorado'),
(7, 'Connecticut'),
(8, 'Delaware'),
(9, 'District of Columbia'),
(10, 'Florida'),
(11, 'Georgia'),
(12, 'Guam'),
(13, 'Hawaii'),
(14, 'Idaho'),
(15, 'Illinois'),
(16, 'Indiana'),
(17, 'Iowa'),
(18, 'Kansas'),
(19, 'Kentucky'),
(20, 'Louisiana'),
(21, 'Maine'),
(22, 'Maryland'),
(23, 'Massachusetts'),
(24, 'Michigan'),
(25, 'Minnesota'),
(26, 'Mississippi'),
(27, 'Missouri'),
(28, 'Montana'),
(29, 'Nebraska'),
(30, 'Nevada'),
(31, 'New Hampshire'),
(32, 'New Jersey'),
(33, 'New Mexico'),
(34, 'New York'),
(35, 'North Carolina'),
(36, 'North Dakota'),
(37, 'Ohio'),
(38, 'Oklahoma'),
(39, 'Oregon'),
(40, 'Pennsylvania'),
(41, 'Puerto Rico'),
(42, 'Rhode Island'),
(43, 'South Carolina'),
(44, 'South Dakota'),
(45, 'Tennessee'),
(46, 'Texas'),
(47, 'Utah'),
(48, 'Vermont'),
(49, 'Virginia'),
(50, 'Virgin Islands'),
(51, 'Washington'),
(52, 'West Virginia'),
(53, 'Wisconsin'),
(54, 'Wyoming');
```

```
--
-- Table structure for table `student`
--

CREATE TABLE IF NOT EXISTS `student` (
  `id_student` int(11) NOT NULL AUTO_INCREMENT,
  `firstname` varchar(40) COLLATE utf8_unicode_ci NOT NULL,
  `lastname` varchar(40) COLLATE utf8_unicode_ci NOT NULL,
  `telephone` int(11) NOT NULL,
  `email` text COLLATE utf8_unicode_ci NOT NULL,
  `dob` date NOT NULL,
  `ss` int(9) NOT NULL,
  `address1` text COLLATE utf8_unicode_ci NOT NULL,
  `address2` text COLLATE utf8_unicode_ci NOT NULL,
  `city` text COLLATE utf8_unicode_ci NOT NULL,
  `zip` int(5) NOT NULL,
  `state` int(11) NOT NULL,
  `major` int(11) NOT NULL,
  UNIQUE KEY `id_student` (`id_student`)
) ENGINE=MyISAM  DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci
AUTO_INCREMENT=2 ;
```

```
--
-- Dumping data for table `student`
--

INSERT INTO `student` (`id_student`, `firstname`, `lastname`, `telephone`, `email`, `dob`, `ss`,
`address1`, `address2`, `city`, `zip`, `state`, `major`) VALUES
(1, 'Michal', 'Szumilo', 0, '', '2000-01-01', 111223333, '100 Home St.', 'Unit 100', 'Chicago', 60000,
1, 0);
```

```
--
-- Table structure for table `student_progress`
--

CREATE TABLE IF NOT EXISTS `student_progress` (
  `id_student_progress` int(11) NOT NULL AUTO_INCREMENT,
  `student` int(11) NOT NULL,
  `semester` int(11) NOT NULL,
  `finalgrade` int(11) NOT NULL,
  PRIMARY KEY (`id_student_progress`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci
AUTO_INCREMENT=1 ;
```