Software Engineering
CSE 308.01

Team Shield
Phillip Elliott
Jeffrey Kabot
Evan
Guby
Weize Lin

Design Document

Section 1: Functionality

The overview and functional requirements of the course scheduling system are taken from the Project Description document provided by Professor Stoller. This document is found on Blackboard in the assignments section for this course under the name “project.txt.”

Section 2: Use Case Diagrams

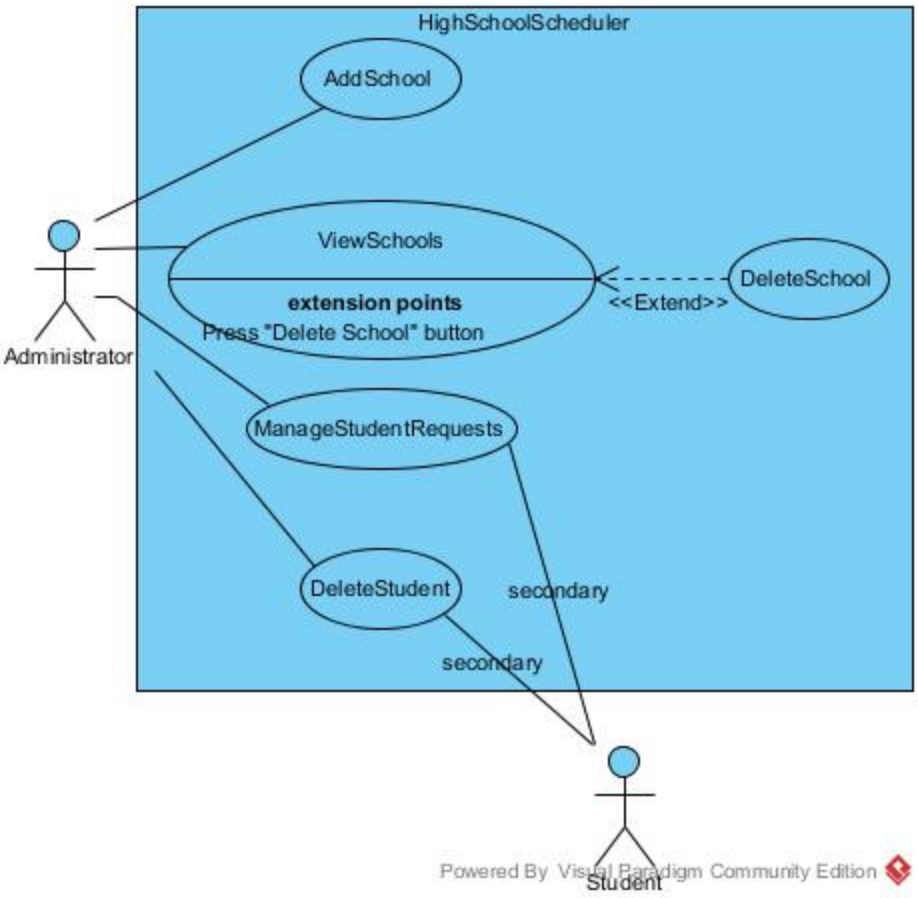

Figure 2.1 -- Administrator Use Case Diagram
The System administrator manages the inclusion and exclusion of schools and student accounts.

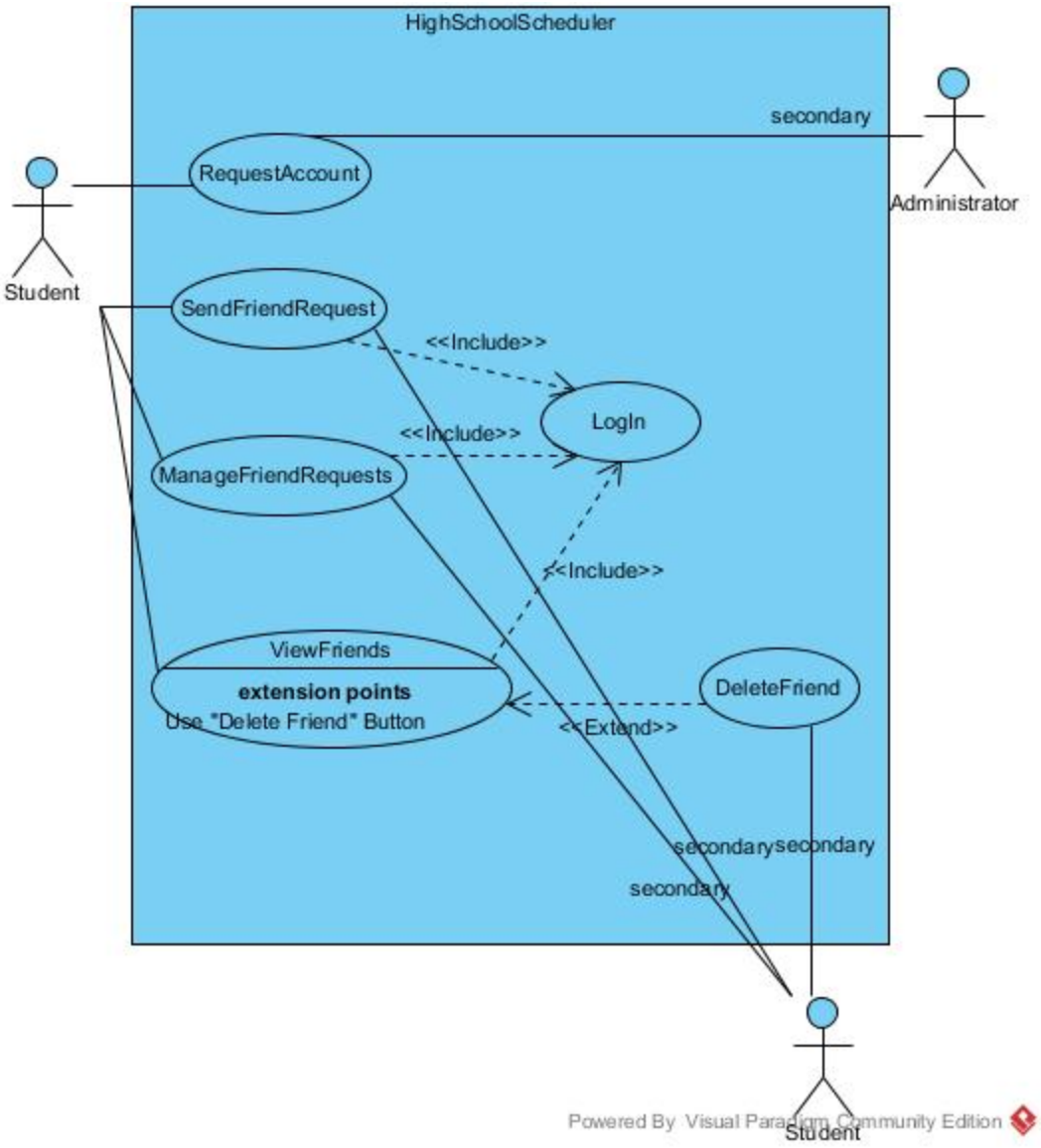

Figure 2.2 -- Student Use Case Diagram 1

Students can maintain a set of friends within the system, so that the schedules they generate may overlap. Students without an account must first request one.

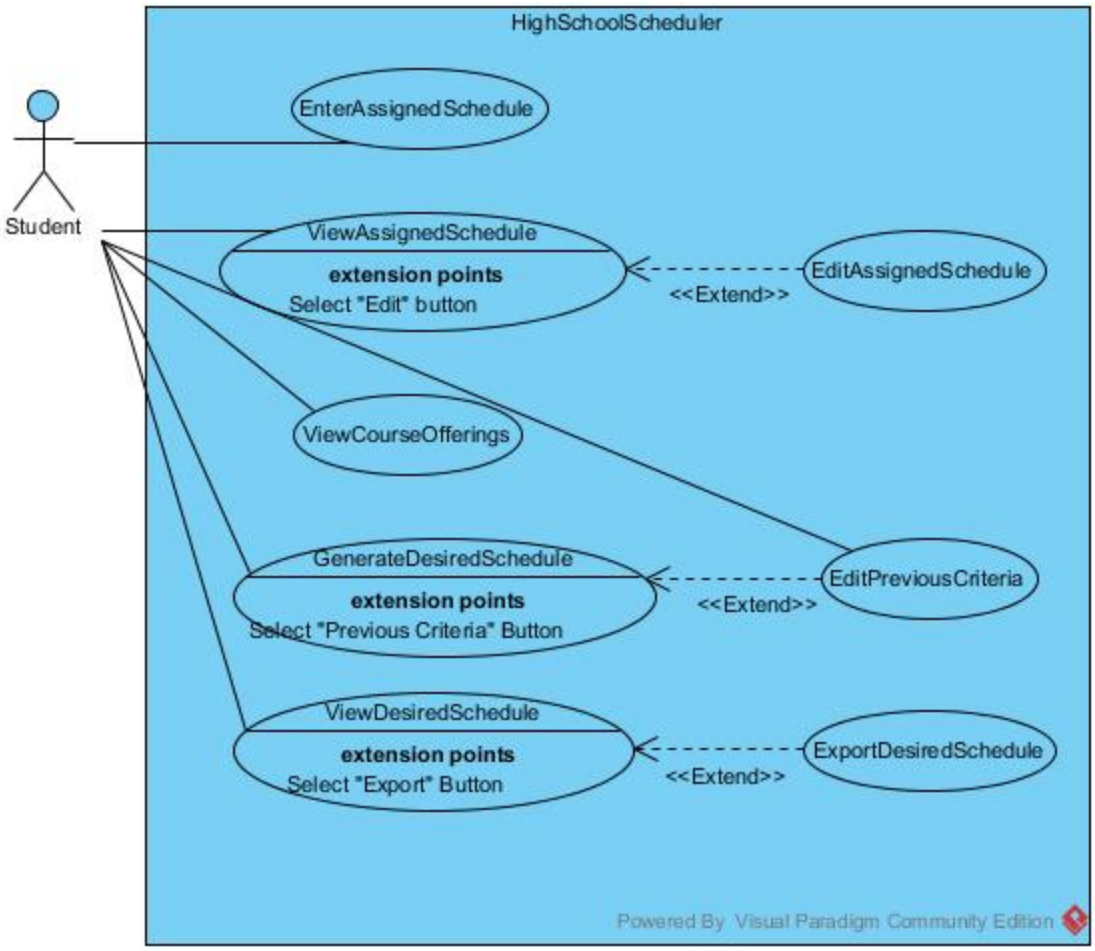

Figure 2.3 -- Student Use Case Diagram 2

Students can enter their assigned schedule so the system learns what courses are available and their friends can make schedules that overlap with the students'. Students can generate potential schedules according to their desired criteria.

Section 3: Use Cases

Use Case: LogIn
Description: User supplies some log-in credentials to the system to authenticate their identity. Upon authentication the user gains their account-appropriate privileges within the system.
Primary Actor: Administrator, Student
Trigger: Logged-off user accesses the schedule system homepage.
Post-conditions: User gains access to their account's abilities within the system.
Primary Flow:

1. User enters the account name and password.
2. System checks the credentials.
1. User is logged in to the system.

Alternate Flow:

1. If the wrong password is supplied for an account, go back to the log-in screen.
2. If the wrong account name is supplied, ask the user if they would like to create a student account.

Note: For all use cases with the exception of "RequestAccount", we assume that the user has sucessfully logged into the system with a valid account.

Administrator Use Cases

Use Case: AddSchool
Description: A system administrator can add a new school to the schools tracked by the course scheduler system. The administrator supplies parameters that determine how the school's schedule is set up.
Primary Actor: Administrator
Post Condition: School will be added to system.
Primary Flow:

1. Administrator enters number of semesters.
2. Administrator enters number of schedule days.
3. Administrator enters number of periods in each day.
4. Administrator enters range of lunch periods.
5. Administrator enters set of all legal schedule blocks.
6. School is saved to the system.

Use Case: ViewSchools
Description: System administrator can retrieve a list of schools that have been added to the course scheduler system.
Primary Actor: Administrator

Primary Flow:

1. Administrator selects the view school option.
2. System displays the list of schools.

Alternative Flow: No schools in system.
Precondition: No schools have been added to the system yet.
2.1. System displays message informing the administrator no schools were found in the system.

Use Case: DeleteSchool
Description: Administrator can delete a school that had previously added to the system.
Primary Actor: Administrator
Post Condition: A school will be deleted from the system
Primary Flow:

1. Administrator selects a school from the view schools list.
2. Administrator selects the delete school option.
3. System prompts administrator to confirm deletion of the school.
4. The school is removed from the system.

Use Case: ManageStudentRequests
Description: Administrator can view an act upon the incoming student account requests. For each request the administrator can approve or deny the student account. The administrator can optionally approve all the incoming requests at once.
Primary Actor: Administrator
Secondary Actor: Student
Primary Flow:

1. Administrator selects Manage Student Accounts option.
2. System displays the list of pending student account requests.
3. Administrator accepts, rejects, or ignores requests.
4. If a student account is approved, the system pushes a notification to the student.

Use Case: DeleteStudent
Description: Administrator can remove a student from the course scheduler system.
Primary Actor: Administrator
Post-conditions: Student will be deleted from the system.
Primary Flow:

1. Administrator selects Manage Student Accounts option.
2. Administrator enters student to be deleted from system.
3. System prompts Administrator to confirm deletion of student.
4. The student account is removed from the system.

Student Use Cases

Use Case: RequestAccount
Description: A student without an account can request access to the course scheduler system for their school. Upon approval by a system administrator the student can use the system.
Primary Actor: Student
Secondary Actor: Administrator
Post-conditions: Administrator receives an account request for that student for the school they specify.
Primary Flow:

1. Student fills out a form with his/her account name, password and school
2. System displays “request sent.”

Alternate Flow:

1. If the school does not exist, system displays “Invalid school, try again.”

Use Case: SendFriendRequest
Description: A student can add another student as a friend within the system, so that they may generate coordinating schedules. Upon approval the users are both friends of each other.
Primary Actor: Student
Secondary Actor: Student
Primary Flow:

1. Student enter a first name and last name, send it to system
2. If an account with that name exists, a friend request is sent to that account

Use Case: Manage Friend Request
Description: Student can observe and act upon their incoming friend requests. The student can choose to approve or deny each request.
Primary Actor: Student
Secondary Actor: Student
Primary Flow:

1. Users accept or reject the friend request from system
2. System indicates that their choice was fulfilled.
3. System pushes a notification to the sender of the request of their approval or rejection.

Use Case: DeleteFriend
Description: Student can remove a user that they had previously accepted as a friend.
Primary Actor: Student
Secondary Actor: Student
Primary Flow:

1. Student enter the name of friend

2. System displays the friend
3. Student chooses delete the friend
4. System replies successful or fail message

Use Case: ViewFriends
Description: A student user can view their friends within the system. For each friend they may views their assigned and desired schedules.
Primary Actor: Student
Primary Flow:

1. Users enter the friend's name
2. System displays the friend
3. Users enter a specified school and academic year
4. System displays the friend's assigned and desired course schedules

Use Case: EnterAssignedSchedule
Description: A student enters the schedule they've been assigned by their school. The system uses this information to learn about course offerings as well as permit the student's friends to generate a schedule that overlaps with the student's.
Primary Actor: Student
Post-conditions: Student will have created their Assigned Schedule
Primary Flow:

1. Student selects Enter Assigned Course Schedule option
2. Student enters academic year.
3. Student enters course identifier.
4. Student enters course name.
5. Student enters range of semesters.
6. Student enters schedule block.
7. Student enters instructor.
8. System saves information.

Use Case: ViewAssignedSchedule
Description: The student user views the assigned schedule they had previously entered.
Primary Actor: Student
Pre-conditions: Student must have created an Assigned Schedule
Primary Flow:

1. Student selects View Assigned Course Schedule option.
2. Student chooses to display schedule with or without friends included.
3. System displays students course schedule, displaying friends names in classes if that option was selected.

Use Case: EditAssignedSchedule
Description: The student makes changes to the assigned schedule they had previously entered.
Primary Actor: Student
Pre-conditions: Student must have created an Assigned Schedule
Primary Flow:

1. Student selects Edit Assigned Schedule option.
2. System displays the students schedule.
3. Student chooses a course entry to edit.
4. Student edits information of their choice.
5. System saves updated course schedule.

Use Case: ViewCourseOfferings
Description: Student user browses the courses offered by their school. The entry for each course includes information about the course and the number of students in the scheduler system that have been assigned to it.
Primary Actor: Student
Primary Flow:

1. Student selects View Course Offerings option
2. System displays a list of all courses offered for the specified academic year.

Use Case: GenerateDesiredSchedule
Description: Schedule System generates a potential course schedule for a Student user according to certain supplied criteria.
Primary Actor: Student
Trigger: Student user selects “Generate Schedule” button.
Post-conditions: Supplied criteria is saved to be reused or edited by the student in another generated schedule. If a set of acceptable schedules are found the system stores them for later viewing.
Primary Flow:

1. System prompts student user for a course to be included in the schedule.
2. Student supplies a valid course from the course offerings.
3. The student optionally specifies a desired section or instructor for the course.
4. Repeat steps 1-3 while the student has more desired courses to enter.
5. System prompts the student whether they desire lunch on each day.
6. Student enters their desired lunches.
7. System generates a schedule according to the supplied criteria, and displays “Schedule generated.”

Alternate Flow:

1. If a student has previously supplied criteria for generating a schedule, they may elect to use the previous criteria instead of supplying new criteria in steps 1-6.

2. If no acceptable could be generated, system displays “No acceptable schedule found. Please relax schedule criteria.”

Use Case: EditPreviousCriteria
Description: Student user can examine the criteria used in the last attempt to generate a schedule to try to generate again, rather than starting from scratch. The previous criteria can be modified before use.
Primary Actor: Student
Pre-conditions: Student has previously attempted to generate a schedule with some criteria.
Trigger: The student has chosen to generate a schedule (see “GenerateDesiredSchedule”) and has selected the “use previous criteria” option.
Post-conditions: If modifications were made to the criteria, the changes are saved.
Primary Flow:

1. System retrieves the criteria saved from the previous schedule generation operation and displays it to the student.
2. Student chooses to add, delete, or edit any criterion.
3. System saves the modified criteria.
4. Student chooses to continue with generating a schedule with these criteria or abort generation.

Use Case: ViewDesiredSchedule
Description: Student user can view an optimal course schedule that has previously been generated according to the student’s desired criteria. The optimal schedule can be determined with or without taking into consideration overlap with the student’s friends’ schedules.
Primary Actor: Student
Pre-conditions: Student must have previously successfully generated an acceptable schedule.
Trigger: Student user selects “View Generated Schedule” button.
Primary Flow:

1. System prompts student whether or not the potential schedule should consider friends’ schedules.
2. System displays an acceptable schedule according to the student’s criteria.

Alternate Flow:

1. If no schedule had previously been generated, display “Please generate a schedule first.”
2. If no acceptable schedule had previously been generated, display “No acceptable schedule found. Please relax schedule criteria.”

Use Case: ExportDesiredSchedule
Description: Student user can export a potential course schedule generated by the system so it can be saved or shared with others.

Primary Actor: Student
Pre-conditions: Student is viewing an optimal course schedule that has previously been generated (see “ViewDesiredSchedule”).
Trigger: Student selects “Export” button while viewing a generated schedule.
Primary Flow:

1. System prompts the student for the format in which the course schedule should be supplied.
2. Student selects the desired format option.
3. Schedule is transmitted to the student in that format.

Section 4: System Architecture

Below is a UML Component Diagram showing the basic components involved with the system architecture.

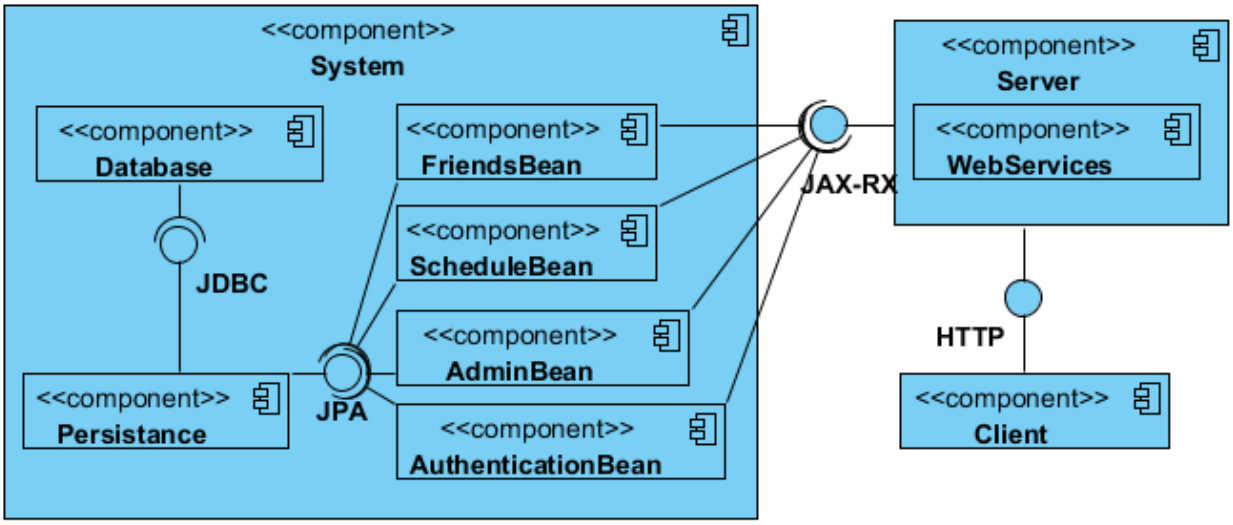

Figure 4.1 -- Component Diagram

The Schedule Bean will handle all logic involving Students setting up and editing their schedules.
The Friend Bean component will handle all logic associated with with friend list functionality. This includes adding, removing, and managing friend requests.

The Account Management component will handle logic associated with account creation and deletion.

The Admin Bean is a composition of two smaller component beans which will handle logic associated with school creation, editing and deletion as well as approving and denying student accounts.

The Authentication Bean will handle logic associated with logging in to the system such as account authorization.

The Persistence component will allow us to take our object oriented system and convert it into a database by using JPA

The Database component will store all data associated with the project by communicating with the Persistence component.

The System component is the combination of the components mentioned above.

The functionality implemented within the Beans will be exposed to RESTful Web Services hosted on the specifc URLs on the server. The Web Services component is the external access-point for these operations on the server.

The Client component is how the users interface with the system. The client communicates with the web services hosted on via HTTP.

Section 5: Class Model

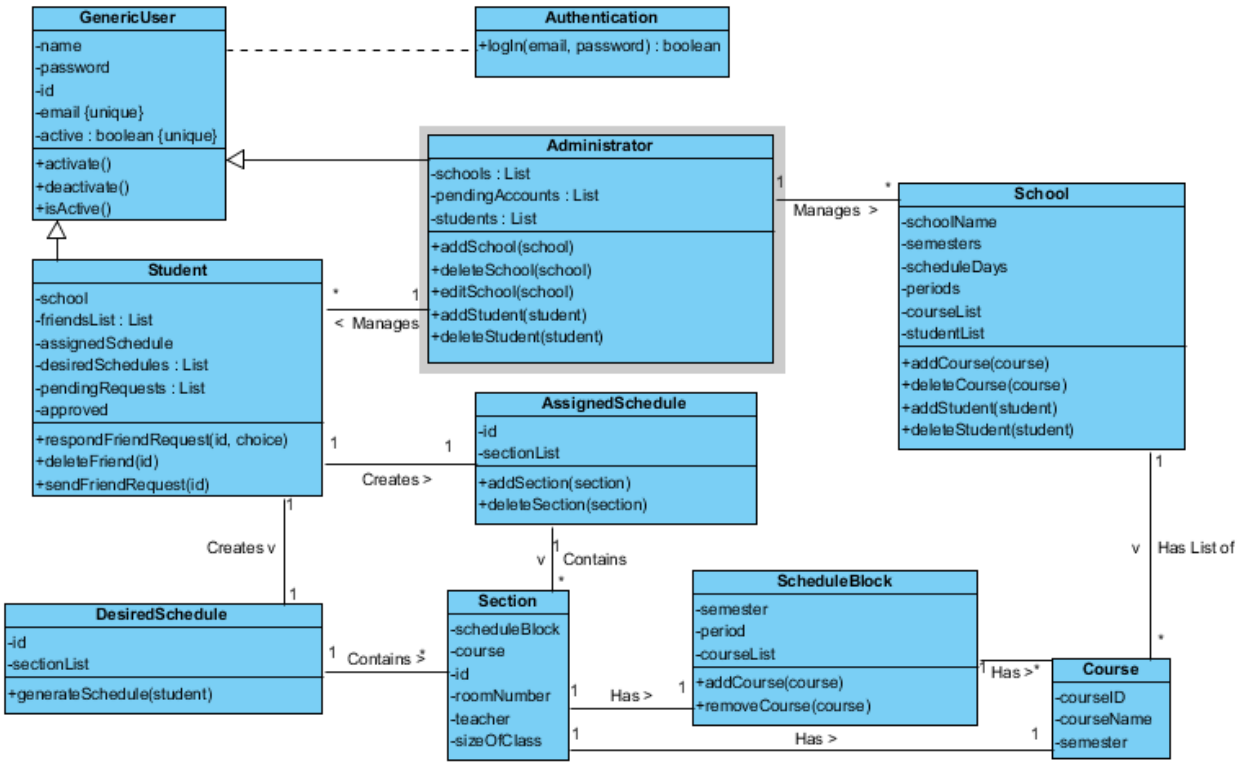

Figure 5.1 -- Abstracted class model for the School Scheduling System. This model illustrates how different notions within the system are related and interact. The actual implementation will be different due to the technologies and and tools we will be using.

Section 6: GUI

LogIn Page

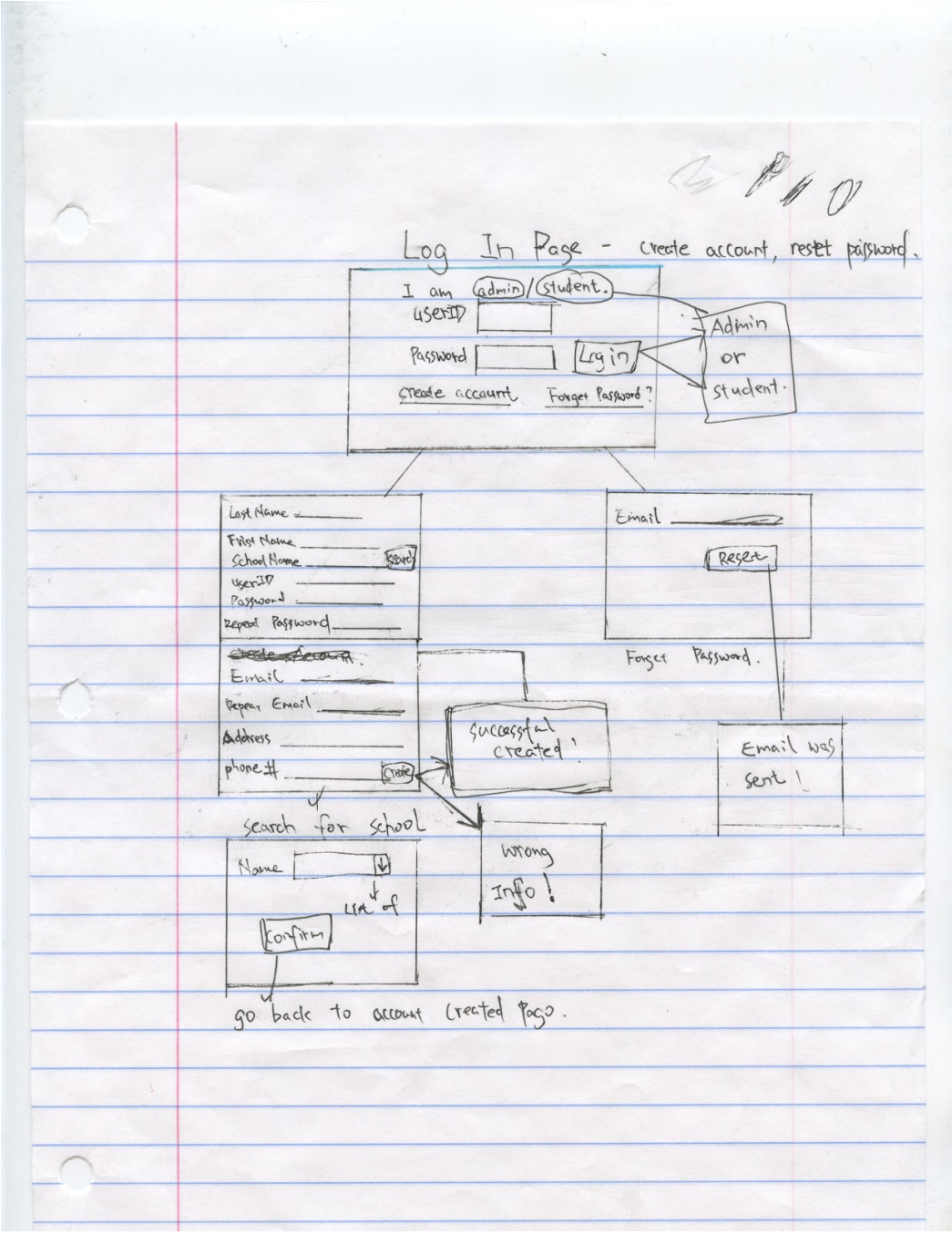

Figure 6.1 -- Log-in GUI. This is the first page a user sees when accessing the website. Depending on the type of user, the login button leads to the administrator hub or the student hub, detailed in the succeeding figures.

Administration Page -- Manage Schools

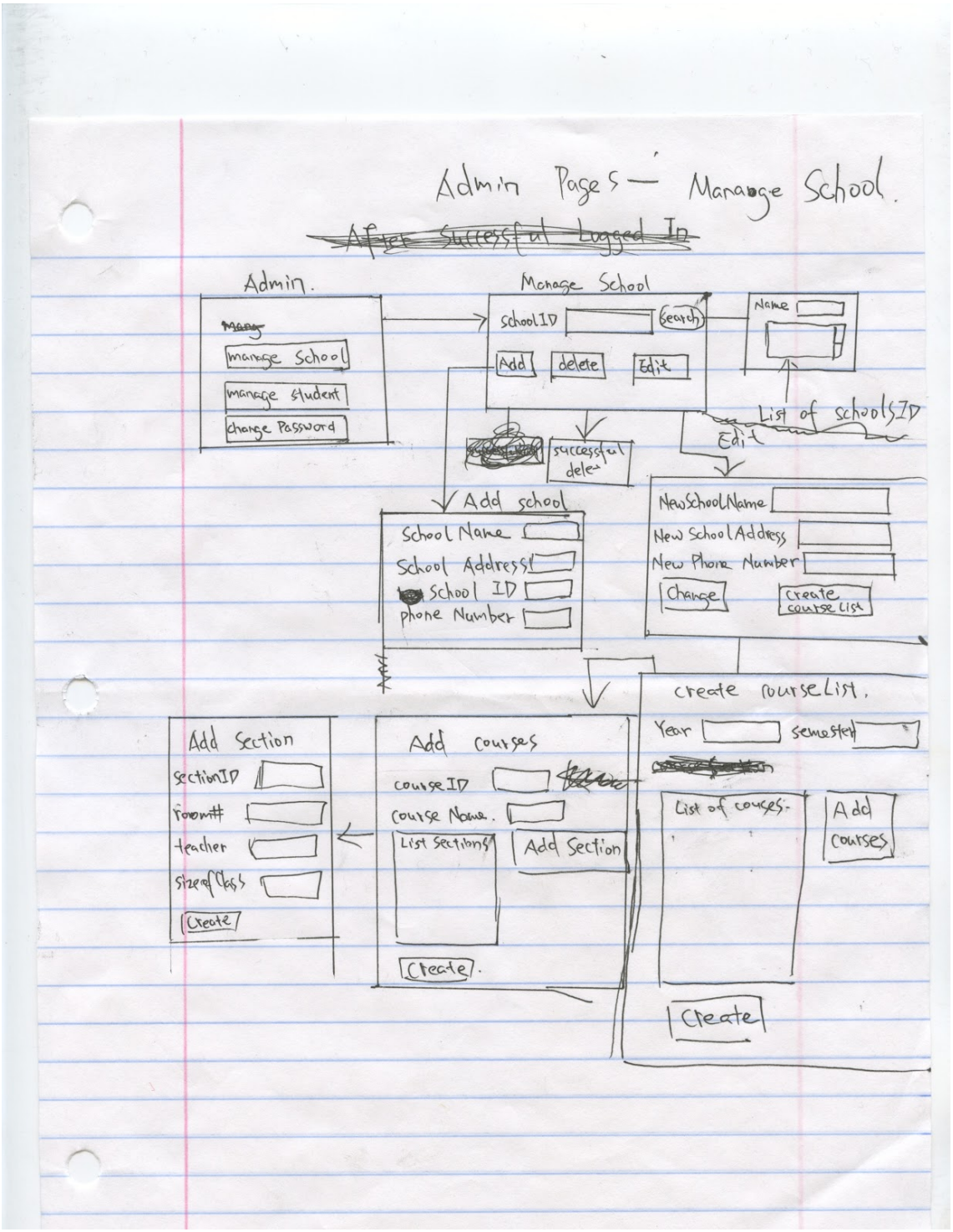

Figure 6.2 -- Manage schools GUI for administrators. Manage schools functionality is accessed from the main administrator hub.

Administration Page -- Manage Student Accounts and Change Password

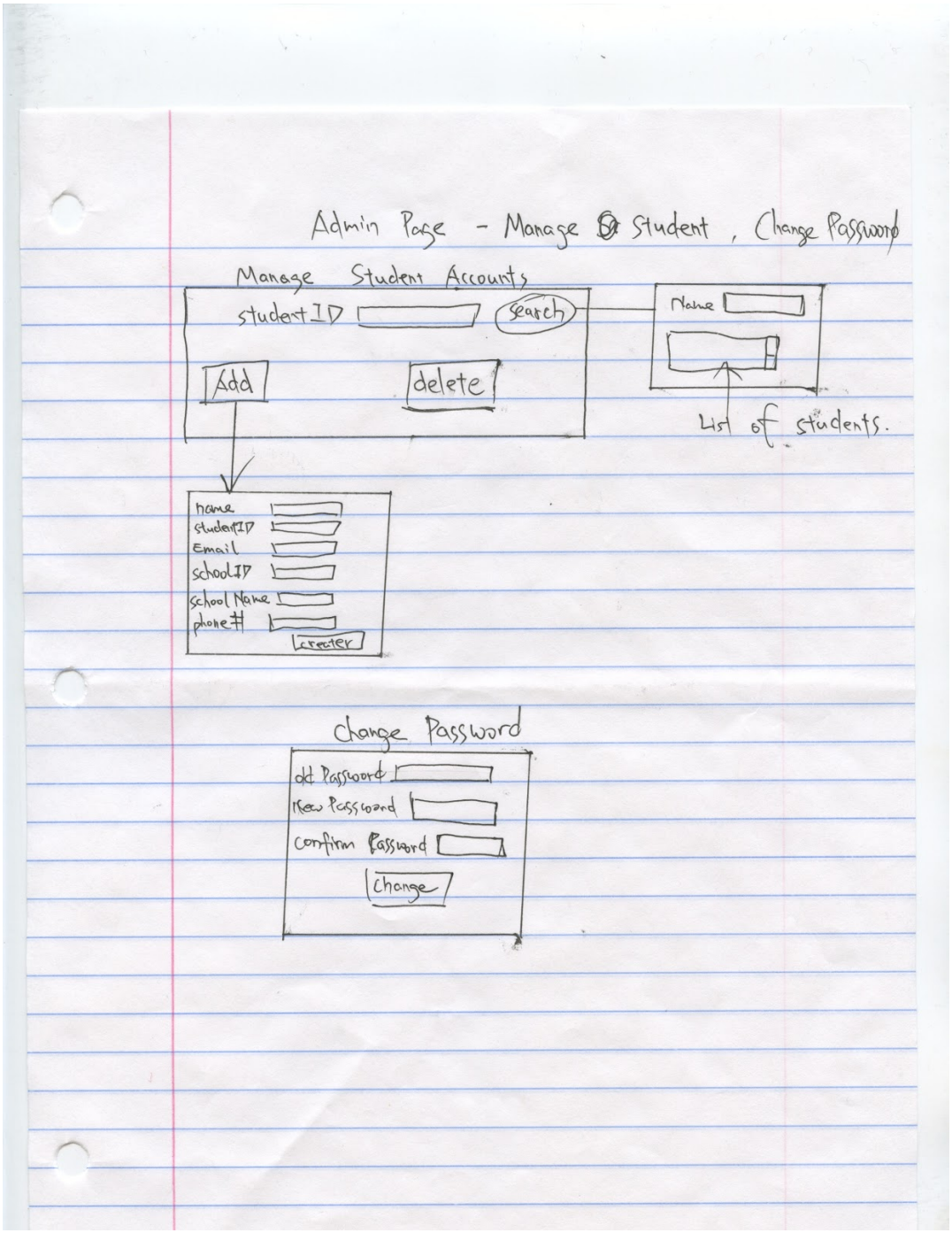

Figure 6.3 -- Manage Students and Change Password GUIs. These pages are accessed from the main administrator hub (see Fig 6.2).

Student Page

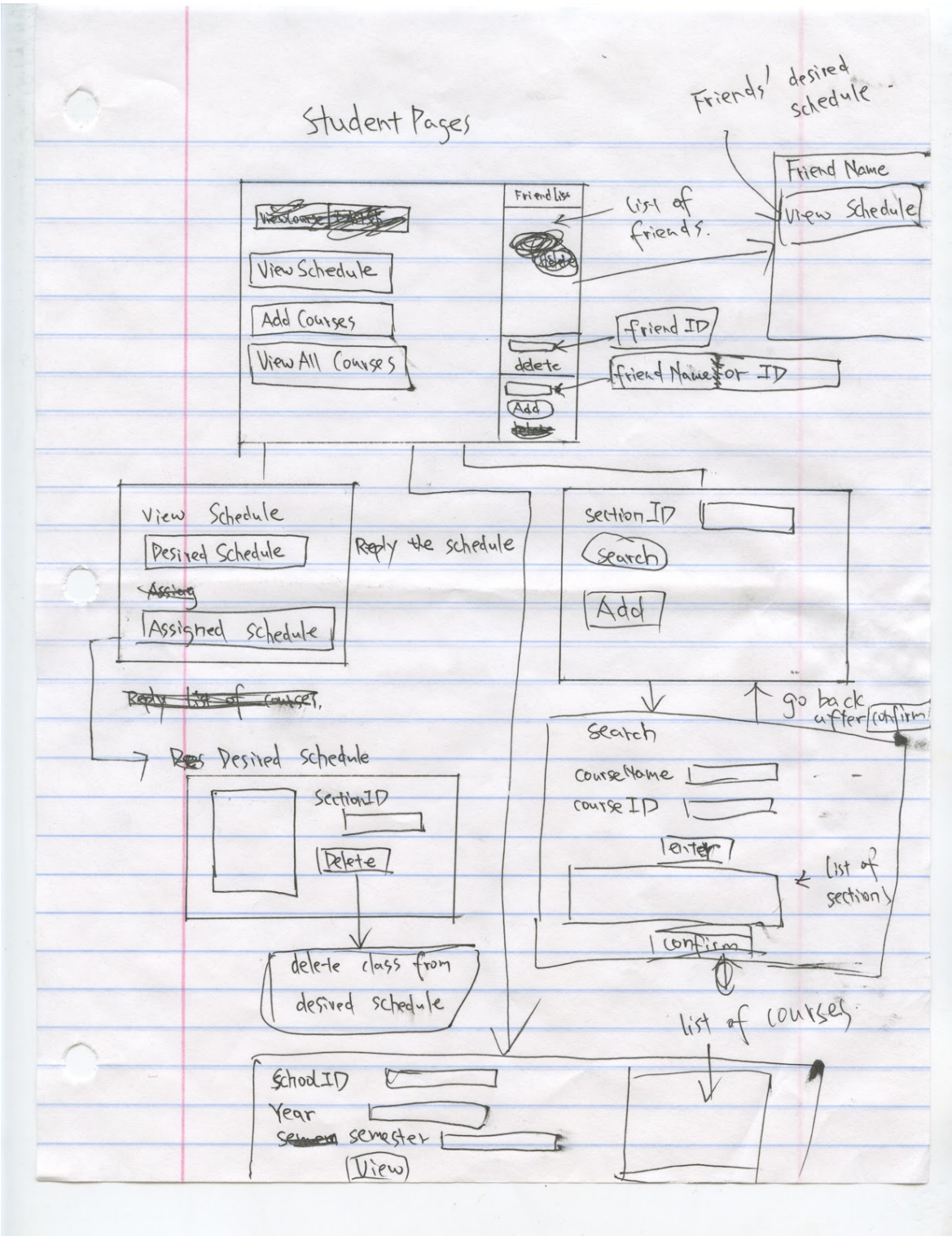

Figure 6.4 -- GUI for Student users.

Section 7: Dynamic Model

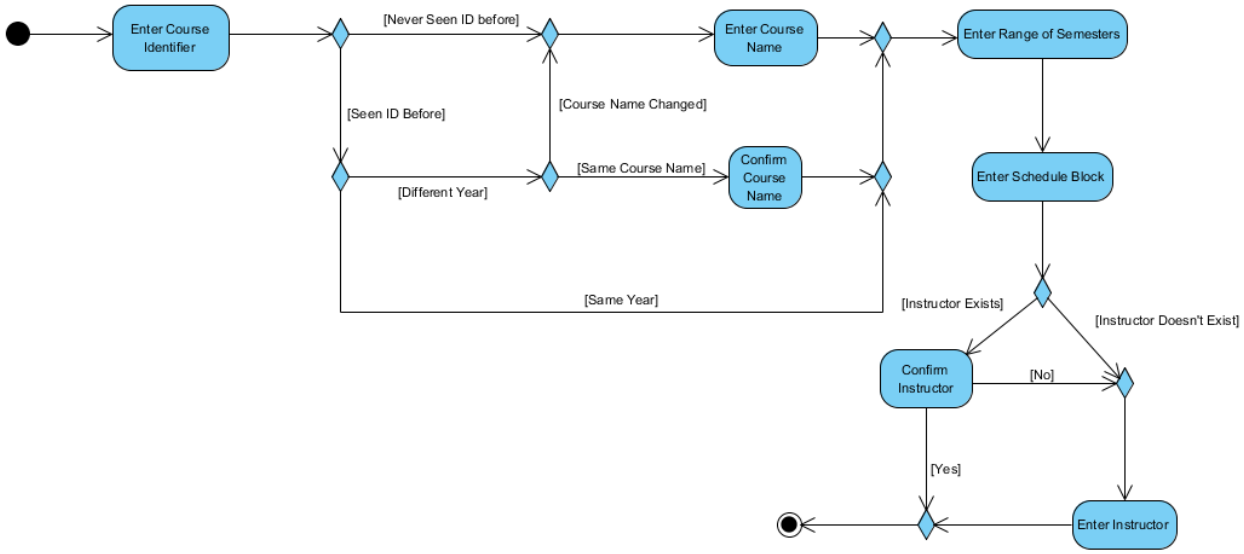

Figure 7.1 -- **Activity Diagram for Entering a Course into an Assigned Schedule.**

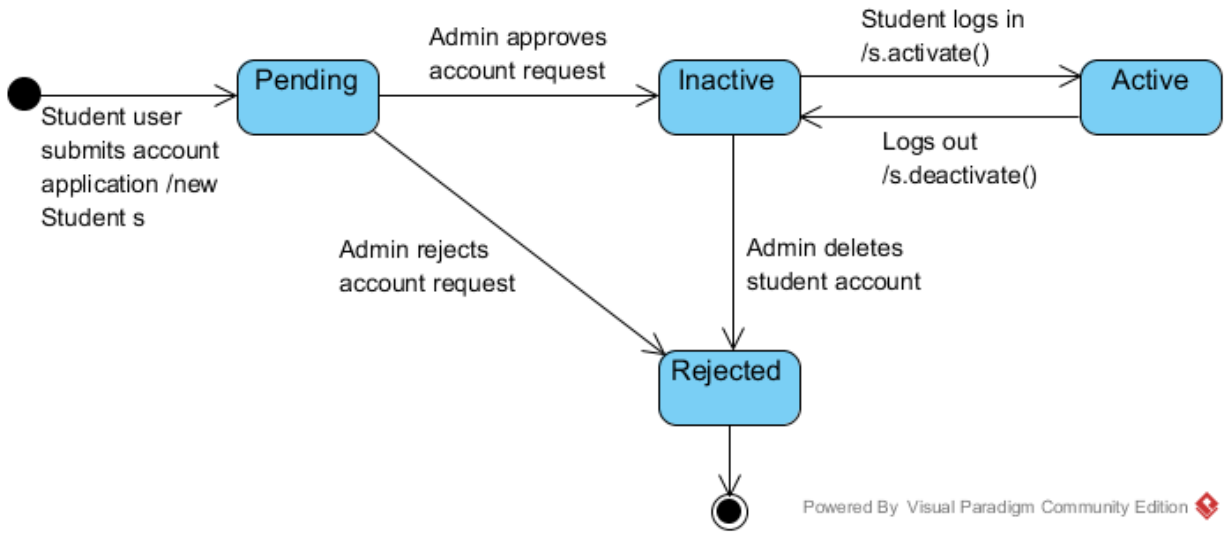

Figure 7.2 -- **State Diagram for a Student account.** We enforce that student accounts may only be active in one session at a time to simplify synchronization. If an administrator deletes a student with an active session they are automatically logged out.

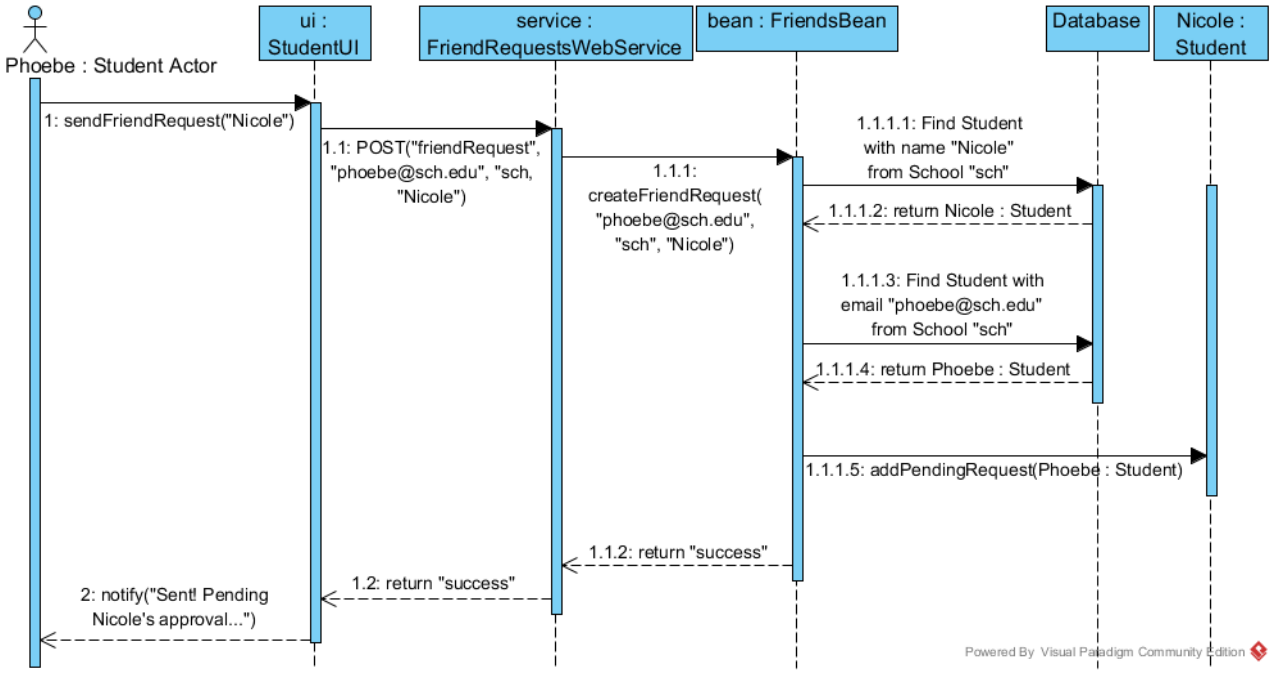

Figure 7.3 -- **Sequence Diagram for Sending a Friend Request.** The user sends a message to the client program by entering some text in a field and clicking a component. The GUI transmits an HTTP request to a URL addressing the corresponding webservice. The webservice parses the body of the HTTP request and passes the user's transmitted info to the EnterpriseJavaBean process running on the server. The Bean executes a database query to retrieve an object needed for the user's action. The object is updated accordingly and refreshed to the database.

Section 8: Technologies, and Tools

We will be using the Netbeans ecosystem for creating our project. The Client and Server applications will both be written in Java, using the JavaFX and JavaEE libraries, respectively. We'll use the Maven project manager to help maintain consistent project builds between our group. GlassFish will provide our server implementation and Hibernate will be our JPA provider We'll use JAX-RS to expose the web resources and handle the HTTP communication between the client and server. The Jackson library will help us translate our Java objects to JSON to be transferred between client and server.

Github will provide our version control system.

We used Visual Paradigm for UML to create the UML diagrams in this design document.