Software Design Specification

For

Automated Intelligent Advisement Interface

Version 1.0

11/07/2016

Michael Harootoonyan

Megan Meyers

Sina Eradat

Jonathan Carrasco

Izzy Hasson

Brandon Garcia

Andrea Lee

Prepared for

COMP 380 Section 15900—Introduction to Software Engineering

Instructor: R. Lingard

Fall 2016

Table of Contents

[1.0 Introduction](#_gjdgxs)

[1.1 Goals and objectives](#_30j0zll)

[1.2 Statement of scope](#_1fob9te)

[1.3 Software context](#_3znysh7)

[1.4 Major constraints](#_2et92p0)

[2.0 Data design](#_tyjcwt)

[2.1 Internal software data structure](#_3dy6vkm)

[2.2 Global data structure](#_1t3h5sf)

[2.3 Temporary data structure](#_4d34og8)

[2.4 Database description](#_2s8eyo1)

[3.0 Architectural and component-level design](#_17dp8vu)

[3.1 System Structure](#_3rdcrjn)

[3.2 Description for Components](#_26in1rg)

[3.3 Component Interface Descriptions](#_lnxbz9)

[4.0 User interface design](#_35nkun2)

[4.1 Description of the user interface](#_1ksv4uv)

[4.2 Interface design rules](#_44sinio)

[5.0 Restrictions, limitations, and constraints](#_2jxsxqh)

## 1.0 Introduction

This purpose of this document is to describe the high-level architecture and system design of the Automated Intelligent Advisement Interface (AI2) from various perspectives. The document describes the data design; architectural and component-level design; user interface design; and the restrictions, limitations, and constraints.

### 1.1 Goals and objectives

The primary purpose of AI2 is to facilitate the successful completion of courses toward a specified degree at California State University, Northridge (CSUN) in a minimum time-frame. Accordingly, the primary objective is to take all guess-work out of long and short-term planning by using algorithms to determine the optimal path to graduation. A student utilizing AI2 will take only the required courses in the correct order that ensures all prerequisites are fulfilled as needed, will know how to satisfy requirement categories efficiently, will be able to change majors with the least amount of wasted effort based on classes the student has already taken, and will be able to balance course loads appropriately (i.e. balance “easy” and “hard” classes effectively). The student will be able to modify the schedule to their own taste in regards to any areas where choices are applicable, such as general education; however, AI2’s algorithms will ensure that the student will proceed correctly to prevent their specified graduation date from being pushed back. Overriding error/warning notifications, while not recommended, will be available.

### 1.2 Statement of scope

The core functionality of AI2 is its ability to look at a student’s general education and major requirements, transcripts, and course catalog, and build a long-term plan for the student to help them graduate in the shortest time possible, while balancing unit count and each semester’s difficulty level through use of a proprietary database of information about each course. Each semester, the relevant courses from the long-term planner are imported into a short-term schedule creator where they are converted to specific sections based on scheduling requirements, modified if desired, and set to the shopping cart for final enrollment. If the student wishes to modify a particular class, AI2 will show them additional options meeting the parameters (e.g. all courses that fall under the Lifelong Learning subject area), or the student may input the class directly. If the student wishes to make modifications outside of AI2’s suggestions or delete a class without replacing it, the system will alert them of how the change will impact their graduation date. During the entire process, the schedule is checked for an acceptable difficulty rating balance, unit count, and unnecessary or incorrect classes; the student is then alerted to any errors. Upon successful completion of a semester, the completed classes are reported to the transcript, and any unsuccessful courses are factored back into the long-term planner with any necessary adjustments.

As an additional function, AI2 can also help a student who wishes to change their major by showing them which majors will make the best use of the classes they have already achieved, thus minimizing wasted time and money.

### 1.3 Software context

AI2 is a component of the CSUN web-based student portal and will interface with its existing databases and structures. The user will gain access to AI2’s functionality via their existing username and password on their myNorthridge Portal. AI2 is not stand-alone software; it is dependent upon the same resources as the current CSUN portal and uses the same protocols and services.

### 1.4 Major constraints

The most pressing constraint for AI2’s development is funding. The state of California provides most of the operating expenses for the California State University system, and students are facing possible tuition hikes due to the 2017 budget, which fell short by one-hundred million dollars. Discretionary funds for technology are currently limited.

In addition, AI2 must be compatible with CSUN’s existing portal and databases. The system must be written in the same language, and student information must be kept private to ensure only the student and authorized individuals have access to their data, as specified by the Family Educational Rights and Privacy Act (FERPA).

Finally, the system must support the following operating systems and platforms: Windows, Linux, IOS, Android, QT, Solaris, IE6-IE10, Chrome, Mozilla Firefox, and Internet Explorer. By extension, this means the system must be able to run on desktops, tablets, and phones.

## 2.0 Data design

An overview of data design is provided in this section, including internal software data structures, global data structures, temporary data structures, and database descriptions. For details on the data contained within each object, see their respective descriptions in section 3.2 Description for Components.

### 2.1 Internal software data structure

The system passes linked lists, array lists, and graphs amongst its components. In addition, some components serve as data structures themselves and may be passed from one component to another (e.g. the Course object may be sent to the Planner component which is then sent to the AI component). For additional details, see 3.3 Component Interface Descriptions.

### 2.2 Global data structure

The following hash tables contain information that is frequently read from but infrequently modified. All are assumed to be small enough to be kept in memory forever.

The keys for both majorTable, and courseTable are obtained from their respective name attributes.

majorTable := LinkedHashMap<String majorName, Major majorInfo>

courseTable := LinkedHashMap<String courseName, Course courseInfo>

The key, semester, shall be of the form [A-Z][a-z][0-9][0-9] where the letters denote season and digits denote year. The attribute called sectionTable is a table of sections for the given semester. Elements of each sectionTable are given their key by their name attribute.

sectionsBySemester := LinkedHashMap<String semester,

LinkedHashMap sectionTable>

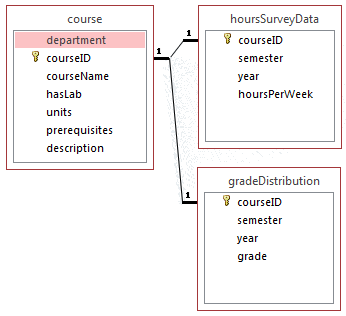
sectionTable := LinkedHashMap<String sectionName, Section sectionInfo>

### 2.3 Temporary data structure

Cookies stored on the user’s machine will be used to retain past data; that is, while the session is still current, users may freely return to a previous page without loss of data.

### 2.4 Database description

The majority of the system’s data comes from CSUN’s databases. However, the system creates its own Course database, which holds the hash table of course information, including its relevant survey data and grade distribution. This database only needs to be accessed on rare occasions, as the included data is compact and relatively static. (While the Course object has an attribute for estimatedDifficulty, which provides the estimated difficulty rating for each course, it is not included in the database due to being a calculated value.)

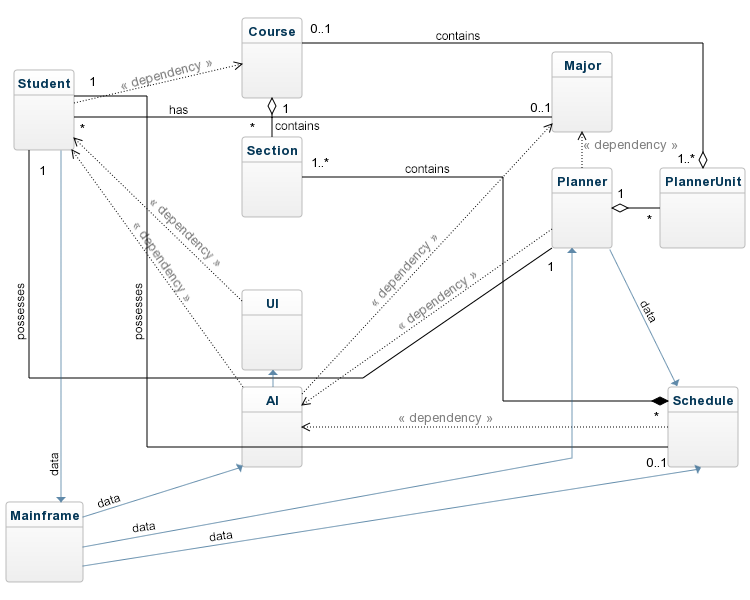


**Course Database Relationships**

## 3.0 Architectural and component-level design

The architecture and component design relies on object-oriented principles; by extension, this entails modularization. Much of the design will defer to CSUN’s existing architecture to upkeep maintainability for the entire organization as a whole. As such, the system will work off of a client-server setup to fall in line with CSUN’s architecture, as well as rely on the myNorthridge Portal user ID and password for authorization and authentication.

### 3.1 System Structure



**Class Diagram for AI2**

The system is composed of multiple components; it must obtain data from multiple databases, such as the list of CSUN’s offered courses and sections, to determine the optimal path to graduation and personalize it to each student. The majority of components serve this purpose and largely work to store information and provide it to the system as necessary. The exception is the AI, which possesses a higher degree of coupling with other components, but in return serves as the central decision-maker when it comes to balancing a student’s planner, schedule, and difficulty rating.

The UI serves as the main means by which the system will communicate with users. It will be displayed on browsers via HTML protocol. The Mainframe will serve as the interface between the system and CSUN’s portal.

### 3.2 Description for Components

An overview of each component is provided. They are shown in more detail in the following sections.

* Student: Possesses student information and serves as the repository for holding the system-generated (and student-modifiable) planner and schedule.
* Major: Possesses major information with courses stored in a graph data structure.
* Course: Possesses course information and the data necessary to calculate the difficulty rating of each course.
* Section: Possesses section information.
* Planner: Possesses planner information and contains planner units.
* PlannerUnit: Makes up the planner; each unit stores information for one semester and year combination.
* Schedule: Possesses schedule information; unlike the planner, the schedule deals with sections, rather than with courses. Schedule in the context of this section refers to the latest term a student is capable of enrolling into (e.g. today, it would be Spring 2017).
* UI: Serves as the interface between the system and the user.
* AI: Serves as the main decision-making body of the system.
* Mainframe: Serves as the interface between the system and CSUN’s portal.

#### **3.2.1 Detailed Description for Student**

|  |
| --- |
| **Student** |
| -studentID: Integer  -firstName: String  -lastName: String  -middleName: Char  -studentDateOfAdmittance: String  -studentMajor: Major  -studentTranscript: LinkedList<Course>  -schedule: Schedule  -planner: Planner |
| +Student(studentID: Integer)  -editSchedule()  -editPlanner()  -updateTranscript() |

The Student object’s predominant functionality is to act as the main data structure in the system, housing both the student’s schedule and planner to be used in the system. (To reiterate, the schedule attribute only refers to the latest semester that the student is able to enroll. Previous schedule information is stored in the transcript.) Each student will have a Student instance, and the object attributes will be mutable throughout the life of the object. Finally, this object is saved to the main student database.

If there is ever an instance where the Planner and Schedule stored in the Student object does not align with the Planner and Schedule object, it will be assumed that the versions stored in the Student object are the most accurate.

The Student class is responsible for holding basic student information, including student ID, first name, last name, middle name, the date they were admitted to the college, major, and transcript, as well as the aforementioned schedule and planner.

#### **3.2.2 Detailed Description for Course**

|  |
| --- |
| **Course** |
| -department: String  -courseID: Integer  -courseName: String  -hasLab: Boolean  -units: Integer  -prerequisites: ArrayList<String[]>  -description: String  /estimatedDifficulty: Integer  -hoursSurveyData: LinkedHashSet<Integer, Integer>  -gradeDistribution: LinkedHashSet<String, Integer> |
| +Course(department:String, courseID:Integer, courseName:String, hasLab:Boolean, units:Integer)  -prereqCheck(ArrayList<String[]>): Boolean  -inputSurveyData(ArrayList<Integer>)  -inputGradeData(ArrayList<String>) |

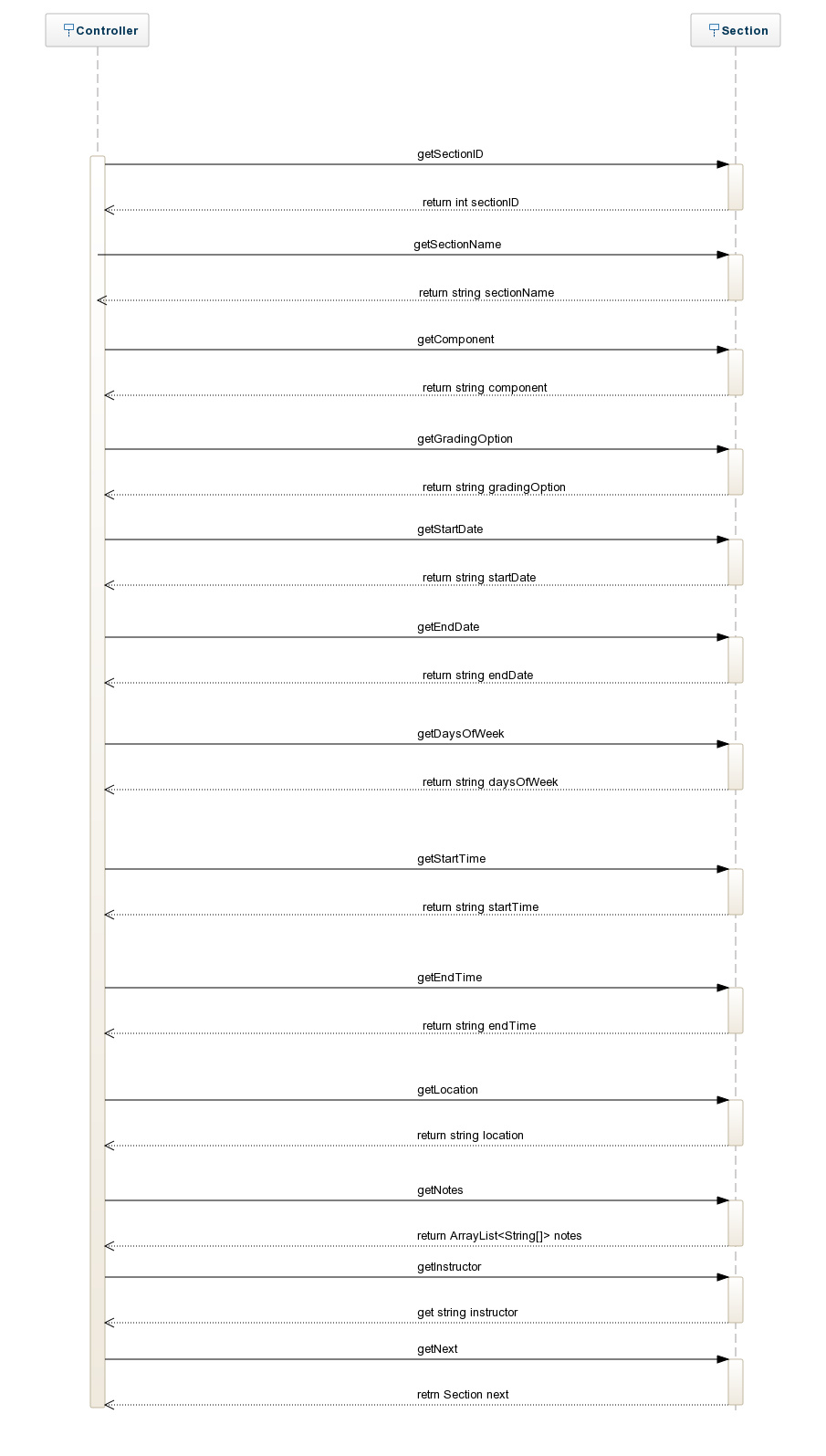
The Course object is responsible for holding course information. In addition, it keeps a record of survey data and grade distribution per course and possesses the functionality to check prerequisites.

The Course object stores the department name, course ID, course name, components of the course, units, prerequisites, course description, difficulty rating for the particular course, and grade distribution.

#### **3.2.3 Detailed Description for Section**

|  |
| --- |
| **Section** |
| -sectionID: Integer  -sectionName: String  -component: String //e.g. LEC, LAB, ACT  -gradingOption: String //e.g. GRD - Graded, CNC - Credit/No Credit  -startDate: String  -endDate: String  -daysOfWeek: String  -startTime: String  -endTime: String  -location: String  -notes: ArrayList<String[]> //e.g. 5207 - Registration limited to majors requiring the course, E - Applied Arts and Sciences  -instructor: String //default is Staff |
| +Section(sectionID: Integer, sectionName: String, component: String, gradingOption: String) |

This Section object is responsible for holding section information. It is a pure storage object that inherits from the Course object, and is utilized for display purposes.



The section object stores the section ID, section name, component, grading option, start date, end date, days of week, start time, end time, location, notes, and instructor. The default value for instructor is “Staff.”

#### **3.2.4 Detailed Description for Major**

|  |
| --- |
| **Major** |
| -majorID: Integer  -majorName: String  -majorReq: Graph<Courses[]> |
| +Major(majorID: Integer, majorName: String)  -addMajorReq(courseID: Integer)  -displayMajorReq() |

This Major object is responsible for holding major information. It stores requirements in a graph data structure, and is utilized for display purposes.

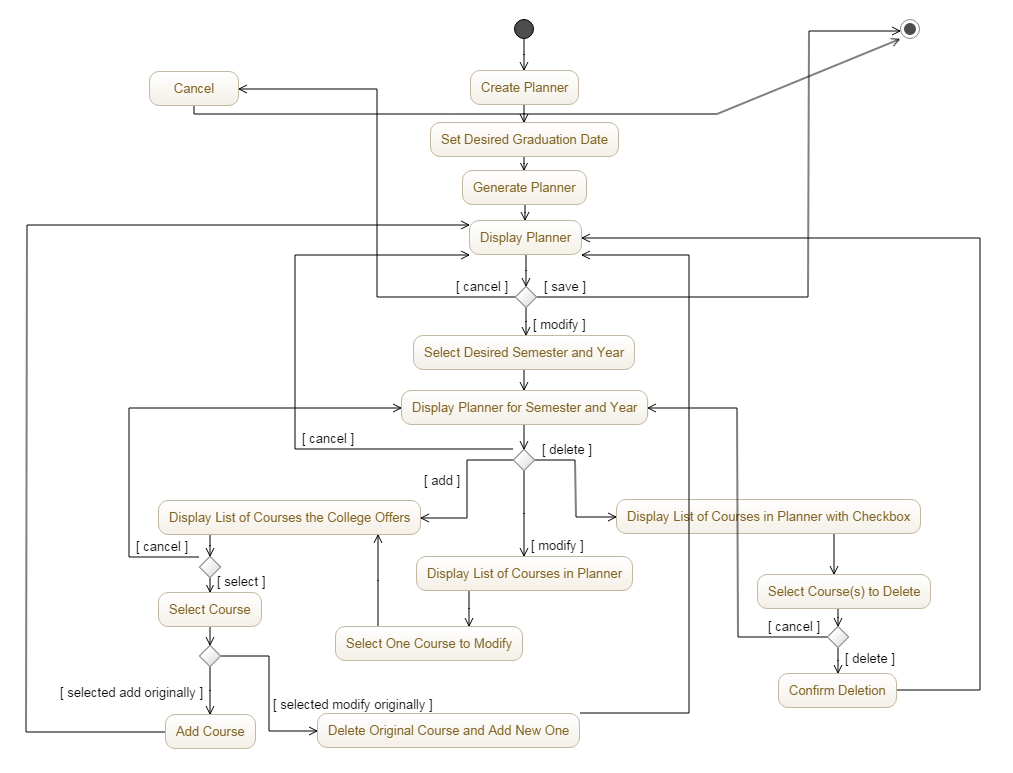
The major object stores the major ID and major name; in addition, it stores the major requirements in a graph data structure.

#### **3.2.5 Detailed Description for Planner**

|  |
| --- |
| **Planner** |
| -PlannerUnits: PlannerUnit[]  -goalGraduationSemeter: String  -goalGraduationYear: Integer //default is four years after the student was admitted to the college |
| +Planner()  -editPlanner(semester: String, year: Integer)  -orderChronologically() |

The Planner object is responsible for maintaining a start-to-finish graduation plan tailored to each user. It serves predominantly as a data structure.

The Planner object is composed of planner units, with each unit having a unique semester and year combination. In addition, it stores the user’s graduation semester and year goal. The default graduation semester and year will be four years since the user was accepted into the institution; however, the user may manually adjust it, such as if they intended to take twice as long and maintain a part-time student status.



**Activity Diagram for Viewing and Confirming a Planner**

#### **3.2.6 Detailed Description for PlannerUnit**

|  |
| --- |
| **PlannerUnit** |
| -semester: String  -year: Integer  -maxUnits: Integer  -difficultyRating: Integer  -courses: Course[] |
| +PlannerItem(semester: String, year: Integer)  -addCourse(courseID: Integer): Boolean  -removeCourse(courseID: Integer): Boolean  -swapCourses(addCourseID: Integer,  removeCourseID: Integer): Boolean |

The PlannerUnit object is a mutable storage object that facilitates the updating of an array list of Course objects. Essentially, the PlannerUnit objects are one of the building blocks for the Planner object.

The PlannerUnit object will house semester and year it applies to, the max units it needs to take into account, the difficulty rating of the specific semester and year combination (to be calculated based on the courses that compose the unit), and an array list of Course objects.

The PlannerUnit object has the ability to add, remove, and swap course objects from the array list of Course objects; after modifications have been made, it will be returned to the Planner object.

#### **3.2.7 Detailed Description for Schedule**

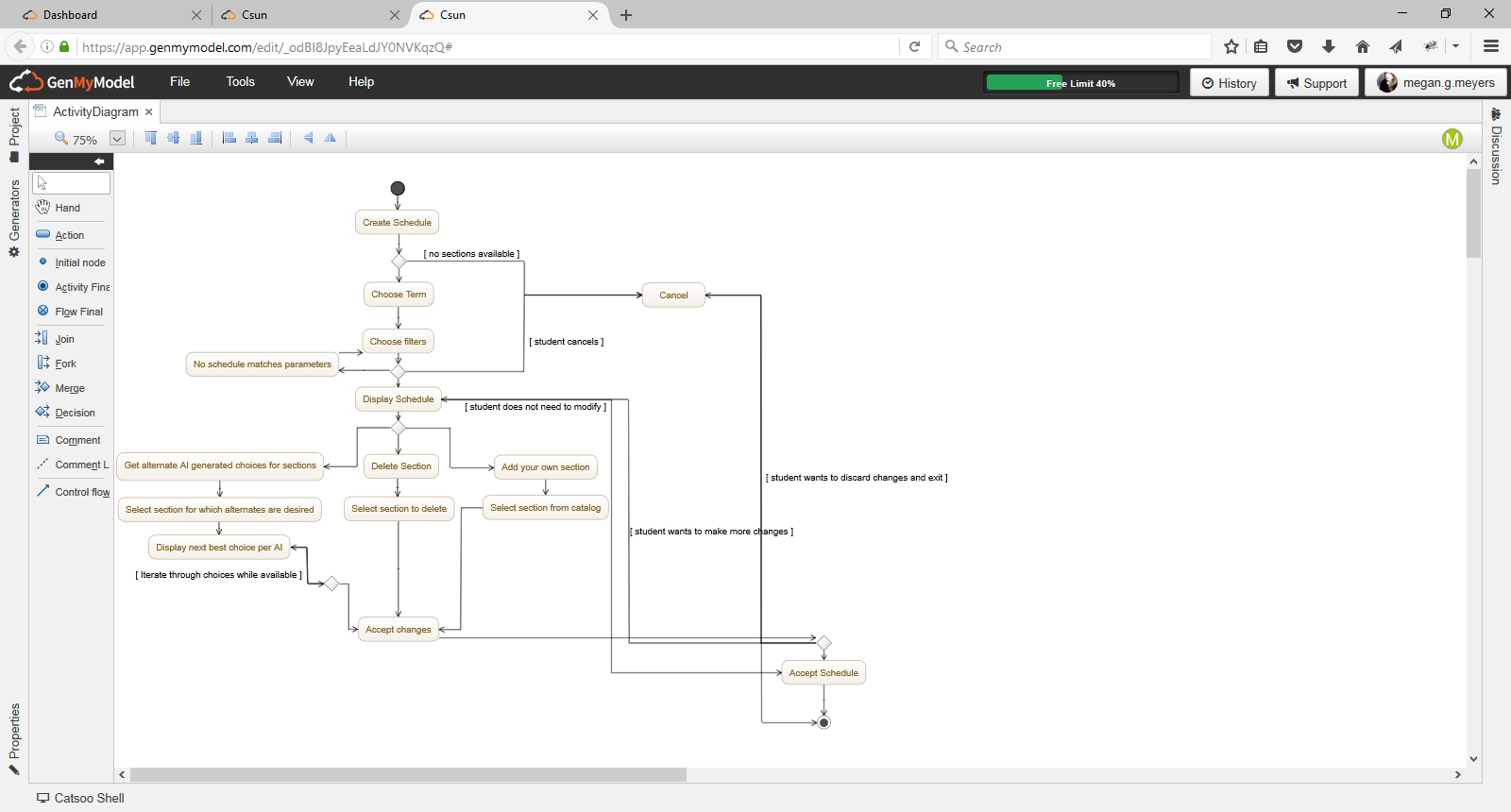
|  |
| --- |
| **Schedule** |
| -semester: String  -year: Integer  -targetDifficulty: Integer  -totalDifficulty: Integer  -maxUnits: Integer  -countUnits: Integer  -numSections: Integer  -sections: Section[] |
| +Schedule(semester: String, year: Integer)  -importPlanner(semester: String, year: Integer): Schedule //pulls courses from the planner for the latest available semester  -addSection(sectionID: Integer)  -deleteSection(sectionID: Integer)  -suggestAlternative(): Section[] //suggests list of alternative courses that will fulfill the same requirement; e.g. if it’s for a GE fulfillment |

The Schedule object is responsible for maintaining schedule information for each user.

It is capable of importing course objects from the planner class based on semester and year. If the user wishes to modify a class, they select that section and ask to see the next best algorithmically generated sections for that requirement category. They may also delete a class or manually add a class.

This class must be able to access the schedule of available sections for a particular semester. Thus, the Schedule class functionality may not be available to students at all times. If the user tries to use it prior to the publishing of the upcoming semester’s schedule, error handling will send the user back to the main menu with a notification.

The Schedule object stores the semester, year, target difficulty, total difficulty, max units, count units, number of sections, and an array list of sections that the schedule is composed of.



**Activity Diagram for Viewing and Confirming a Schedule**

#### **3.2.8 Detailed Description for UI**

|  |
| --- |
| **UI** |
| +Planner: ArrayList<String[]>  +Schedule: ArrayList<String[]> //these arrays serve as output only |
| +UI(studentID: Integer)  -viewPlanner()  -assignColorCode()  -viewSchedule() |

The responsibility of the UI object is to serve as the front-end modus operandi for communication between a user and the system. The UI class will instantiate once per call to the system and will persist through the life of the system.

The UI has the capacity to retrieve and display the student’s planner and schedule; when it comes to the planner, the UI will be able to add color-coding to differentiate between the courses the student has already taken, the required courses they are eligible to take based on perquisite fulfillment, and the required courses they are ineligible to take.

#### **3.2.9 Detailed Description for AI**

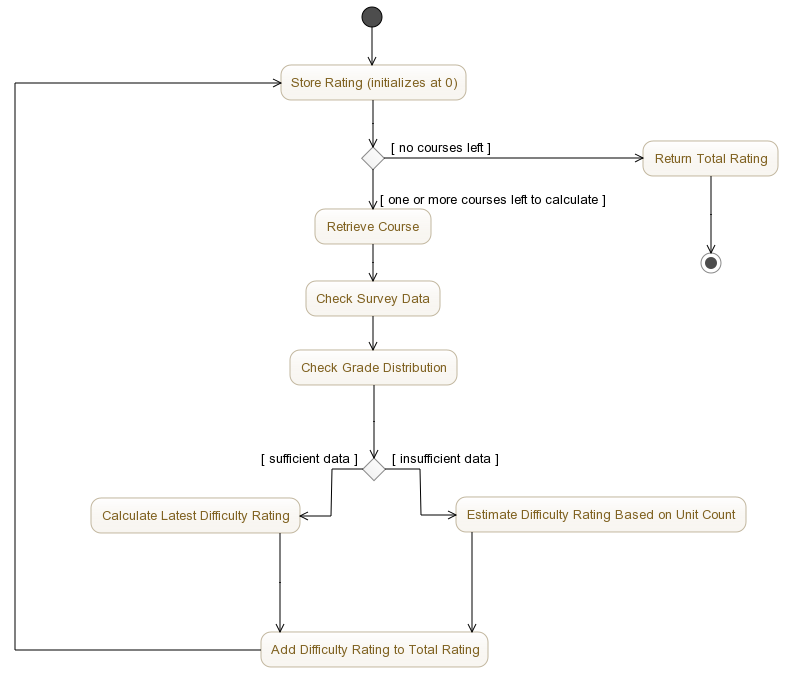
|  |
| --- |
| **AI** |
| None |
| +generatePlanner(studentID: Integer): Planner  +generateSchedule(Planner: Planner): Schedule  +calculateDifficultyRating(Planner: Planner)  +checkChangeOfMajor(studentID: Integer, majorID: Integer): Course[]  +optimizeChangeInMajor(studentID: Integer, majorID: Integer)  +impactedGradWarning(studentID: Integer, Planner: Planner)  -createChain(courseID: Integer)  -addAll(courseID: Integer) //takes one course and links ALL pre- and post-requisites  -findPrereq(courseID: Integer)  -findPostreq(courseID: Integer)  -compareMajor(majorID: Integer) //this would compare the chain with the major to know where to cut off; e.g. certain majors only require you to go up to COMP182 so it would remove COMP282, COMP380, etc. from the chain  -compareTranscript(studentID: Integer) //same as above, except it cuts off links in the chain if the student already took the class  -linkExists(courseID: Integer) //checks to see if a specific course is in the chain; e.g. a COMP 110 to COMP 490 chain wouldn’t include a HIST 101 link  -cutLinks(courseID: Integer, prerequisite: Boolean) //this is the function that would sever links. It begins at the relevant course and either removes the entire left side of the chain (Boolean = true), or the entire right side (Boolean = false); e.g. cutLinks(COMP182, false) would cut ALL the postrequisites for 182 while leaving 182 intact  -countLinks(): Integer //more links means a longer chain  -displayChain() |

The AI class has no attributes of its own. Instead, the responsibility of the class is to serve as the final decision-maker of the system. It is capable of finding and resolving prerequisites for a given course. It resolves course prerequisites by creating a topological ordering by finding the prerequisites and corequisites for the course then recursively resolving the prerequisites and corequisites of all courses found.

In addition, it may automatically generate a planner based on the student’s major; it works similarly to the roadmaps CSUN current offers, save for the fact that it takes the student’s transcript into consideration and make adjustments based on the classes they have already completed.

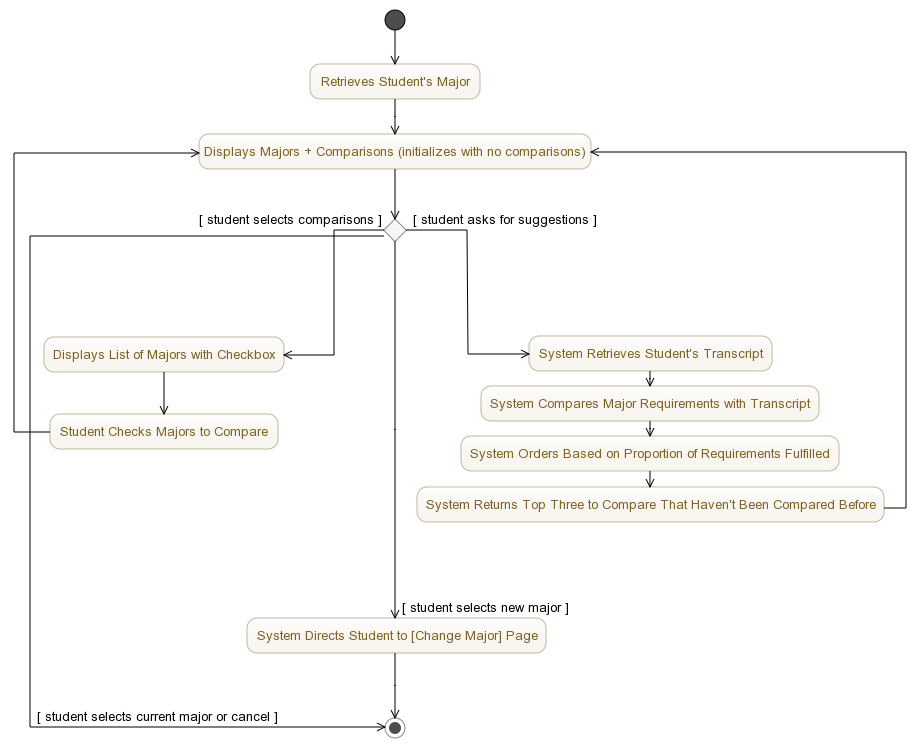
It is capable of generating a schedule as well; first by calling the Schedule class (which would have imported data from the Planner to create a preliminary schedule) and then by making adjustments. It uses the filter options set forth by the user such as schedule blocks (e.g. no class on Fridays) and class status (open, waitlisted, closed) to convert each course into a specific section. In addition, if a course is not being offered in the chosen semester, the AI will use an algorithm to offer a new suggestion.

As the AI makes adjustments, it sends the updated information for storage in their respective classes. It also calculates and sends the difficulty rating for each semester to be stored as necessary in the PlannerUnit and Schedule objects.



**Activity Diagram for Calculating Difficulty Rating**

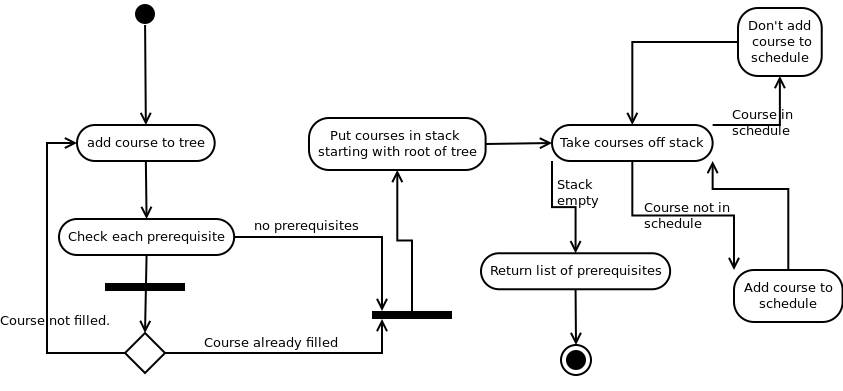
The AI class is able to aid users in optimally changing majors as well. Users have the option to select majors they’re interested in, and the AI will compare the major requirements to the courses the user has already taken to determine which major will require the least rework. Alternatively, users may ask the AI to compare the courses they’ve taken with all majors offered by CSUN, and the AI will return the three most optimal choices.



**Activity Diagram for Changing Majors**

The AI class is also responsible for issuing warnings. When the AI automatically generates a planner or schedule, or when a user manually makes adjustments, the AI will check that prerequisites are filled (or planned to be filled) and that the user will still be able to graduate on the date they specified. If the user makes any changes that prevents them from meeting the specified graduation date, the AI will display a warning, as well as alternatives the user may choose to maintain the same graduation date. If the user opts not to accept an alternative, they may override the warning so long as they accept their graduation date will be impacted. If it is impossible to retain the graduation date (for instance, the student set it to one year after they were admitted to the college), the system will display a warning and automatically recalculate the date to four years after the user was admitted to the college.

Finally, the AI class is capable of performing “chaining,” which is a function that checks the dependencies of each course (the prerequisites, corequisites, and postrequisites) to better assign priorities. For instance, if a course is a prerequisite to three other courses, it will be given a higher priority than a course that is only a prerequisite to one. Similarly, if a course is at the head of a long chain (that is, the user will need more semesters to go from the start of the chain to the end), it will be given a higher priority than a course that is at the head of a smaller chain.



#### **3.2.10 Detailed Description for Mainframe**

The Mainframe class is responsible for serving as the interface between the system and CSUN’s portal. It ensures that the user has received authorization and authentication via the myNorthridge Portal.

### 3.3 Component Interface Descriptions

#### **3.3.1 Student**

The Student object will have direct access to the CSUN student database system. The Student object will perform a database retrieve operation by passing a MySQL query request to the connected CSUN student database to retrieve data for an individual’s student ID, full legal name, date of admittance, major, and array list for an individual’s transcripts which will be stored in the appropriate Student object attributes.

When a student’s planner is first generated, the Student object will pass its date of admittance to the Planner object so a default graduation date may be calculated.

The Student object will pass its major and transcript attributes to the AI, so that they may be used to process the student’s necessary course requirements and see which requirements have already been fulfilled. If the student’s information is modified via the AI (for instance, the user switched majors or modified their planner or schedule), the updates will be pushed onto and stored within the Student object.

The Student object will have the ability to trigger functions in the Schedule and Planner objects. The Student object will have the capacity to call upon the Schedule object with an editSchedule function which will pass a Schedule object back to the Student object to store or update the Schedule attribute. The Student object will have the capacity to call upon the Planner object with an editPlanner function which will pass a Planner back to the Student object to store or update the Planner attribute.

#### **3.3.2 Course**

The Course object will access CSUN’s course database by passing a MySQL query request to retrieve the data for department, course IS, course name, whether or not it has an accompanying lab, prerequisites, and description, which will be stored in the appropriate Course object attributes.

In addition, the Course object keeps a record of survey data and grade distribution per course. This may be passed onto the AI, after which they can be used to calculate the difficulty rating of a particular course.

#### **3.3.3 Section**

The Section object will access CSUN’s section database by passing a MySQL query request to retrieve the data for section ID, section name, component (such as whether or not it’s a lecture, a lab, or something else), starting date, ending date, days of the week the class meets, start time, end time, location, and any additional notes, which will be stored in the appropriate Section object attributes.

In addition, the Section object may pass on the above information to the AI so they may be used to determine which sections are available for the user.

#### **3.3.4 Major**

The Major object will access CSUN’s major database by passing a MySQL query request to retrieve the data for major ID, major name, and major requirements, which will be stored in the appropriate Major object attributes.

When the user chooses a major, the Major object passes an array list of Course objects to the AI object in order to evaluate it against the student’s transcript array list and generate a planner and schedule.

#### **3.3.5 Planner**

Upon request by the UI to display Planner, the Planner structure will be passed to the UI for display to the user.

To reiterate, the Planner object is largely a data structure. Information from this class is sent to the AI, wherein modifications may be made based on user activity (e.g. the plan will make accommodations if the user was unable to enroll in a necessary class, if they failed, or if they purposely chose to deviate from the proposed plan). If modifications are made, the updates will be pushed back onto the Planner object and stored.

In addition, the AI will check the overall Planner as well as the composition of its PlannerUnits to ensure all prerequisites have been fulfilled and that the ordering is allowed (that is, it does not jump from COMP 110 to COMP 380, but goes through the proper steps of 110 to 182 to 282 and so forth). The AI will also compare any proposed changes to the graduate date stored in the Planner object and alert the user if any actions on their part will push back their graduation date. If the graduation date needs to be changed, the update will be pushed back onto the Planner object.

#### **3.3.6 PlannerUnit**

Upon completion of auxiliary operations by the user or AI object (e.g. adding or removing courses), an array list of PlannerUnit objects will be returned to the Planner object. This will, in turn, be passed to the AI so that the AI object may calculate the difficulty rating for the different semesters.

If the user makes any modifications to a particular semester via the AI, the AI will push the updates back to the PlannerUnit, and then to the Planner object. While the AI may make suggestions and warnings (for instance, if the difficulty rating is too high or the student skips a prerequisite), the user is ultimately free to add or remove courses as desired.

#### **3.3.7 Schedule**

The Schedule imports courses from the Planner object for the latest available semester (e.g. if done today, it would pull courses for the Spring 2017 semester).

Like the Planner, information from the Schedule class will be sent to the AI. If the user chooses to deviate from the system-generated schedule, the AI will check the schedule automatically for errors in the user’s choices. If errors are detected, the student will not be able to continue until they correct the schedule or override the warnings. The student may cancel and discard the schedule at any time.

Any updates made to the Schedule via the AI will be pushed back to the Schedule object, and then the Planner object.

#### **3.3.8 UI**

The UI will have access to the Student object and will primarily serve as a display mechanism for the Student object’s attributes.

The UI will be able to check the student’s general education and major requirements. It can cross check this with the student’s transcript to color-code the display appropriately (e.g. required courses that the student has already fulfilled will be visually different than required courses the student has not fulfilled).

The UI will be capable of receiving the Student object’s call to edit the Schedule which will place a call to the AI object in order to evaluate the changes in Schedule and return an array list of Course objects to the UI to display, which in turn will return an array list of Course objects to the Student object to update the appropriate Student attribute.

The UI will have the capability to place a call to the AI object to generate either a Schedule or a Plan. In either case, the AI object will retrieve the Student object’s array list of course objects in the Transcripts attribute. Relative to what the array list of Course objects return from the Student object, the AI object will generate a Schedule and a Planner and return the array lists of Course objects to the UI to display.

#### **3.3.9 AI**

The AI object defines the operational component of the system. The AI object will be instantiated in conjunction with the User Interface object, and will persist through the life of the system. The AI object will have access to the Student, Planner, Major, and Course objects in the system and will have the ability to return strings, integers, and array lists of objects or strings to the UI for display.

The AI component will generate the difficulty score of a Schedule and Planner, and return an integer to the UI as well as their respective Schedule or Planner object. The operation will evaluate the array list of Course objects data provided by the CSUN records which will include average grade in the Course, time spent studying outside of class per week, and the pass/fail ratio, after which it will return a cumulative difficulty rating to the UI for display alongside the Schedule object. With the difficulty rating, the AI object will have the capacity to alert the user of possible risks of the Schedule impacting the user’s grades (for instance, if the difficulty rating is so high that the student risks failing one or more sections).

The AI object will have the capability to receive a call from the UI to evaluate a potential change of major and/or optimize a change of major request. In both cases, the AI object will retrieve an array list of Course objects from the Major object and evaluate against the Student object’s array list of Course objects in the Transcripts attribute. Using comparison functionality, the AI object will discern from the already taken classes in the potential new major; if the user chooses to officially change their major, the AI will generate an array list of Course objects of classes that need to be taken as well as the classes that fulfill requirements for the new major. With this array list of Course objects, the AI object will generate a new Planner and Schedule for the user.

The AI object will be capable of passing an array list of Courses to the Student object, which would then be stored or updated as an array list of Courses objects for the Schedule, Planner, and Transcripts attributes.

The AI object will have the ability to call upon the Schedule object to add a section or remove a section. The Schedule object will handle modifications and the new array list of Section objects will be returned to the AI to check for warnings before being passed to the UI for display to the user.

#### **3.3.10 Mainframe**

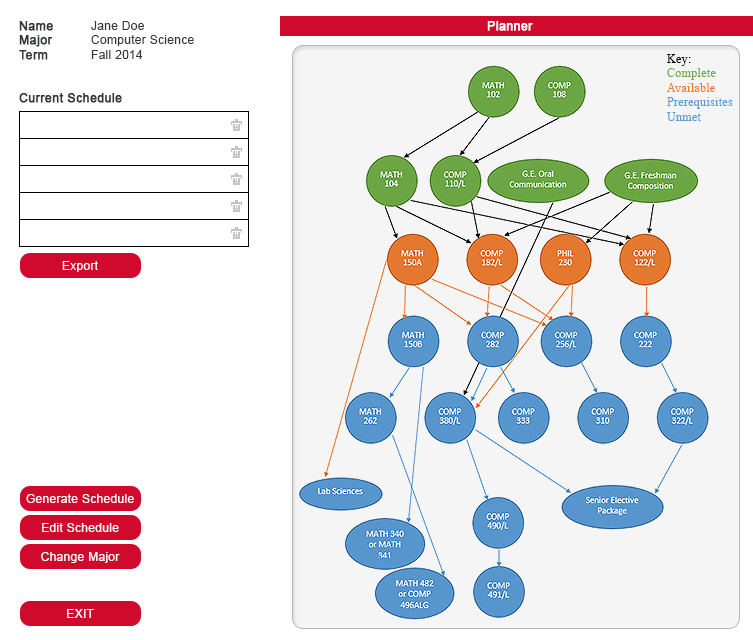
The mainframe receives proof of authorization and authentication and passes it on to the AI.

## 4.0 User interface design

The user interface will follow myNorthridge Portal’s existing conventions in terms of font, style, color, and organizational choices.

### 4.1 Description of the user interface

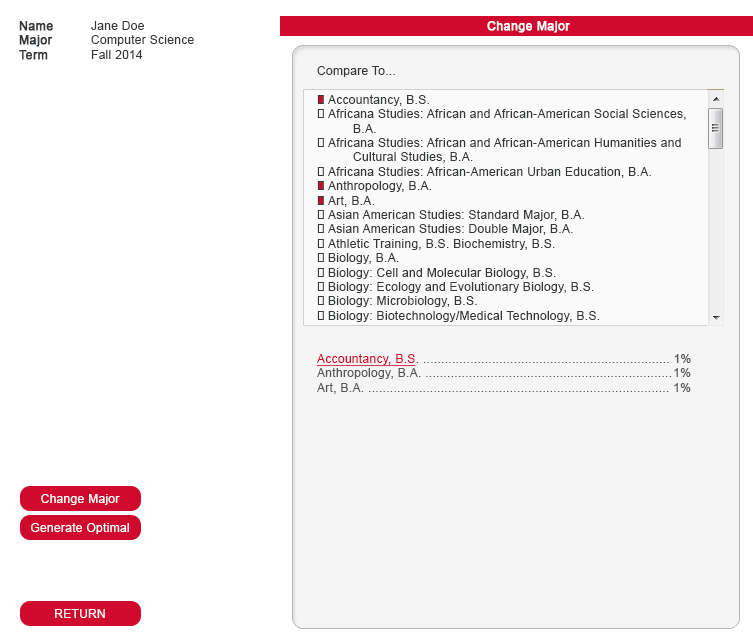
The user interface will be provided via web. The main page will provide a simple and straightforward view of basic student information, with an emphasis on schedule and planner view. It shall provide a list of the Student’s current schedule, if any, as well as a list of completed courses and courses to be taken in the form of a coherent tree diagram.



**Prototype of Main Page**

It will also contain a way to generate a schedule; once the schedule is generated, the interface will allow the student to make modifications and recalculate the schedule difficulty as they deem necessary. Once the difficulty has been calculated and the student is done editing their schedule, they will have the option to print the new schedule and send it to SOLAR.

In addition, the interface will allow to user to quickly consider alternate majors and change their major if necessary.



**Prototype of Changing Majors**

### 4.2 Interface design rules

Sommerville’s HCI principles will be followed when implementing the user interface, which include user familiarity, consistency, minimal surprise, recoverability, user guidance, and user diversity (108). In addition, Shneiderman's "Eight Golden Rules of Interface Design" will be referred to, which include striving for constituency, enabling frequent users to use shortcuts, offering informative feedback, designing dialog to yield closure, offering simple error handling, permitting easy reversal of actions, supporting internal locus of control, and reducing short-term memory load (112).

## 5.0 Restrictions, limitations, and constraints

As mentioned earlier, the project will be constrained when it comes to possessing low funding, having to be compatible with CSUN’s existing portal and database, enforcing FERPA, and being accessible on multiple platforms and browsers.

Low funding means the team is limited to its existing skillset; it cannot afford to hire outside contractors or purchase licenses for high-end tools. In addition, the team will favor open-source coding, software, and graphics.

Compatibility means the code will be written in Java, with the user interface being written in Java Swing. Thus, the design must adhere to all constraints and functionalities inherent in the Java and Java Swing languages. Databases will be written or accessed via MySQL.

Should there be an individual in our team who has a low level of expertise with Java Swing or MySQL, pair programming may be mandatory to speed up development time, or different personnel may be used for a portion of the project that requires personnel that are more familiar with Java Swing or MySQL.

Finally, improvements are already being made to the registration process, as can be seen with CSUN’s recent announcement and introduction of the Registration Planner. To avoid obsolescence, it must be completed before the start of the Fall 2017 semester.