Master Schedulizer

Software Design Specification (SDS)

Created by:

Anh Huynh

Alex Marion

Mahmuda Liza

Purav Barot

Ryan Efendy

Version 1.0

February 1st, 2016

Revision History

Version	Date	Author	Change Description
1.0	2/1/2016	Anh Huynh, Alex Marion, Mahmuda Liza, Purav Barot, Ryan Efendy	Initial version.

TABLE OF CONTENTS

1. INTRODUCTION

1.1. Purpose

This document will outline in detail the software architecture and design for the Master Schedulizer. This document will provide several views of the system's design in order to facilitate communication and understanding of the system. It intends to capture and convey the significant architectural and design decisions that have been made for the Master Schedulizer.

1.2. Scope

This document provides the architecture and design of Master Schedulizer. It will show how the design will accomplish the functional and nonfunctional requirements detailed in the Master Schedulizer Software Requirements document.

1.3. Definitions and Abbreviations

POS - Plan of Study
TMS - Term Master Schedule
CS - Computer Science
JVM - Java Virtual Machine

2. SYSTEM OVERVIEW

Master Schedulizer serves to streamline the process of creating a plan of study, as well as to allow users to keep track of requirements and electives. Given a user's major(s) and minor(s), Master Schedulizer creates a generic template which outlines a full college plan. The user can then add tracks, minors, electives, and other classes of interest. Master Schedulizer will also track requirements and make sure that the plan of study doesn't violate university policy and is possible in the given timeframe (all courses must be available in the chosen term, etc.).

If integrated into Drexel's systems Master Schedulizer would allow advisors to track advisees with greater ease. Heads of departments could also track the number of students who are taking a specific course in any given quarter giving a better estimate of the needed sections.

2.1. Product Perspective

2.1.1. Design Method

The design of this product utilizes an object-oriented approach.

2.1.2. User Interface

The user will be interfacing with the Master Schedulizer System that will allow user to select tracks and help generate Plan of Study. Given the user's major and tracks, it will generate a sufficient Plan of Study for graduation.

Master Schedulizer will be designed such that the user can accurately self-report their own behavior. Overall interface will be intuitive to use and easy to remember.

2.1.3. Hardware Interface

Master Schedulizer will perform on any machines with the JVM (Java Virtual Machine) installed.

2.1.4. Software Interface

Master Schedulizer will be platform independent as a Java program (preferably running in Eclipse IDE).

2.2. User Characteristics

General characteristics of the intended users:

- Knowledge in compiling and running a Java program
- Undergraduate Drexel student status
- Computer Science Drexel student status

2.3. Constraints

The system can only run on a system with a JVM installed.

2.4. Assumptions and Dependencies

It is assumed that the user of Master Schedulizer is an undergraduate, Computer Science major student.

It is assumed that a list of courses taken by the user will be supplied.

2.5. Apportioning of Requirements

There are requirements apportioned to later releases of the simulator.

3. DESIGN CONSIDERATIONS

3.1. Operating Environment

Master Schedulizer is intended to run in any operating systems, as long as a JVM is installed on the local machine.

3.2. Architectural Design / Design Conventions

Master Schedulizer will follow Object Oriented methodologies to maximize encapsulation and modularity.

3.3. User Interface

3.3.1. Expected Input

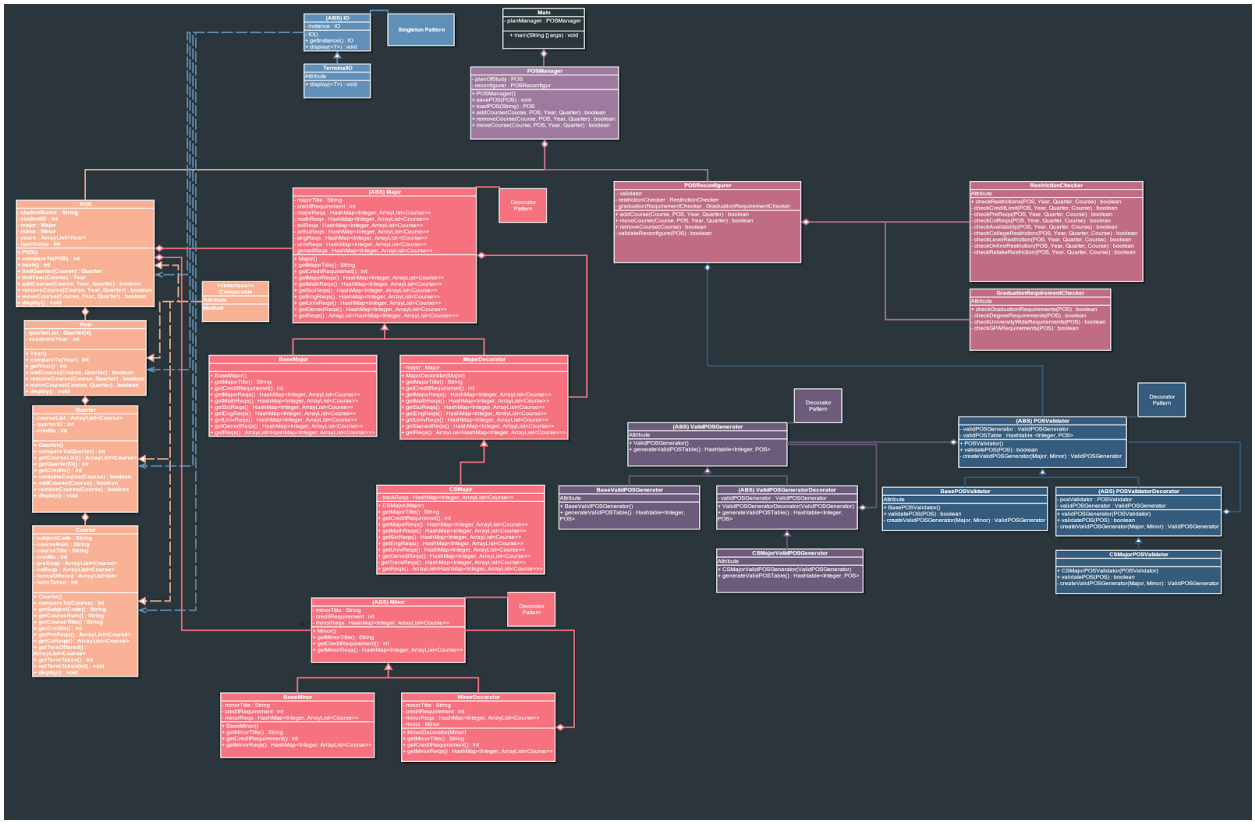
The system expects input in three forms. The first is numerical input from the user for selecting menu options through the terminal interface. The second is a typed file path from the user specifying where their saved plan of study if a previous plan of study exists in the correct CSV format. The third input that the system expects is the CSV file itself to load information from a previous session.

3.3.2. Expected Output

The user will see his/her optimized Plan of Study printed out on the command line, with the option to download to a CSV file. All output messages will be printed to the terminal interface.

4. SYSTEM ARCHITECTURE

4.1. Classes Overview



Link for this Diagram: http://www.pages.drexel.edu/~pbb34/labs/UML.png

4.2. Individual Class Breakdown

4.2.1. POS

4.2.1.1. Attributes of POS

4.2.1.1.1. String studentName: The student's name

4.2.1.1.2. String studentID: The student's Drexel ID

4.2.1.1.3. Major major:
The student's major (can have multiple values)

4.2.1.1.4. Minor minor:
The student's minor (can have multiple values)

4.2.1.1.5. <ArrayList>Year years:
list of years student will spend at Drexel

4.2.1.2. Functions of POS

4.2.1.2.1. compareToPOS(POS): Compare two POS's

4.2.1.2.2. hash():
Hash POS to its appropriate bucket

4.2.1.2.3. findQuarter(Course):
Find which quarter a course was (or might be) taken

4.2.1.2.4. findYear(Course):
Find which year a course was (or might be) taken

4.2.1.2.5. addCourse(Course, Year, Quarter):
Add a course to POS

4.2.1.2.6. removeCourse(Course, Year, Quarter):
Remove a course from POS

4.2.1.2.7. moveCourse(Course, Year, Quarter):
Move a course from one term to another

4.2.1.2.8. display(): Print POS

4.2.2. Year

4.2.2.1. Attributes of Year

4.2.2.1.1. quarterList: list of four quarters

4.2.2.1.2. academicYear: list of years

4.2.2.2. Functions of Year

4.2.2.2.1. compareTo(Year): compare two Year objects

4.2.2.2.2. getYear(): function that will allow us to get a chose year

4.2.2.2.3. addCourse(Course, Quarter): add course to POS

4.2.2.2.4. removeCourse(Course, Quarter): remove course from POS

4.2.2.2.5. moveCourse(Course, Quarter): move course within POS

4.2.2.2.6. display(): display all the years

4.2.3. Quarter

4.2.3.1. Attributes of Quarter

4.2.3.1.1 courseList : ArrayList
Array of all the courses in particular quarter

4.2.3.1.2 quarterID
Specific integer ID for a quarter eg.201515

4.2.3.1.3 credits
Total amount of credits taken in that quarter

4.2.3.2. Functions of Quarter

4.2.3.2.1 comareTo(Quarter)
function to compare one quarter with another

4.2.3.2.2 getCourseList() : ArrayList<Course>
function to get list of courses in that quarter

4.2.3.2.3 getQuarterID : int
function that returns quarter ID

4.2.3.2.4 getCredits() : int
returns total amount of credit taken in that quarter

4.2.3.2.5 containsCourse(Course) : boolean
boolean function that checks if quarter contains course

4.2.3.2.6 addCourse(Course) : boolean
boolean function that checks if course is added,
if it is added then return true, else return false

4.2.3.2.7 removeCourse(Course) : boolean
boolean function that checks if course is removed,
if it is removed then return true, else return false

4.2.3.2.8 display() : void
display all the quarters

4.2.4. Course

4.2.4.1. Attributes of Course

4.2.4.1.1 String subjectCode
specific string code for that course eg.CS, CI

4.2.4.1.2 String courseNum
specific string course number for that course eg. 281, 265

4.2.4.1.3 String courseTitle
specific string course title eg. Systems Architecture

4.2.4.1.4 int credits
number of credits

4.2.4.1.5 preReqs: ArrayList<Course>
ArrayList of pre-requisites for that course

4.2.4.1.6 coReqs : ArrayList<Course>
ArrayList of courses that must be registered too . . .
eg. EXAM080

4.2.4.1.7 termsOffered : ArrayList<int>
describes when course will be offered

4.2.4.1.8 int termTaken(int)
term selected to take that specific course

4.2.4.2. Functions of Course

4.2.4.2.1 Course()

4.2.4.2.2 compareTo(Course) : int

4.2.4.2.3 getSubjectCode() :String
returns subject code of the course

4.2.4.2.4 getCourseNum() : String
returns course number of the course

4.2.4.2.5 getCourseTitle() : String
return course title

4.2.4.2.6 getCredits() : int
return credit

4.2.4.2.7 getPreReqs() : ArrayList<Course>
makes sure that student meets prerequisites for the course

4.2.4.2.8 getCoReqs() : ArrayList<Course>
describes courses that need to be taken with selected course

4.2.4.2.9 getTermsOffered() : ArrayList<Course>
describes terms in which that course will be offered

4.2.4.2.10 getTermTaken() : int

4.2.4.2.11 setTermTaken(int) : void

4.2.4.2.12 display() : void
prints out that course to the POS

4.2.5. Major

4.2.5.1. **Attributes of Major**

4.2.5.1.1 String majorTitle
represents name of the major

4.2.5.1.2 int creditRequirement
represents total credit requirement for that major

4.2.5.1.3 majorReqs : HashMap<Integer, ArrayList<Course>>
Specific major requirements eg. Computer Science(CS . . reqs)

4.2.5.1.4 mathReqs : HashMap<Integer, ArrayList<Course>>
represents Math requirements

4.2.5.1.5 sciReqs : HashMap<Integer, ArrayList<Course>>
represents Science requirements

4.2.5.1.6 arthuReqs : HashMap<Integer, ArrayList<Course>>
represents Art and Humanities requirements

4.2.5.1.7 engReqs : HashMap<Integer, ArrayList<Course>>
Represents all the english requirements

4.2.5.1.8 univReqs : HashMap<Integer, ArrayList<Course>>

represents all the university course requirements

4.2.5.1.9 genedReqs : HashMap<Integer, ArrayList<Course>>
represents all the general electives requirements

4.2.5.2. Functions of Major

4.2.5.2.1 getMajorTitle() : String
returns title of the major

4.2.5.2.2 getCreditRequirement() : int
returns total credit requirements of the major

4.2.5.2.3 getMajorReqs() : HashMap<Integer, ArrayList<Course>>
returns all the major specific course requirements

4.2.5.2.4 getMathReqs() : HashMap<Integer, ArrayList<Course>>
returns total amount of math courses required for that major

4.2.5.2.5 getSciReqs() : HashMap<Integer, ArrayList<Course>>
returns all the science courses required for that major

4.2.5.2.6 getEngReqs() : HashMap<Integer, ArrayList<Course>>
returns total amount of english courses required for that major

4.2.5.2.7 getUnivReqs() : HashMap<Integer, ArrayList<Course>>
returns total amount of all the university courses required

4.2.5.2.8 getgenedReqs() : HashMap<Integer, ArrayList<Course>>
returns total amount of all the general electives that can be taken

4.2.5.2.9 getReqs() : ArrayList<HashMap<Integer, ArrayList<Course>>>

4.2.6. Minor

4.2.6.1. Attributes of Minor

4.2.6.1.1 String minorTitle
represents name of the minor

4.2.6.1.2 int creditRequirement
represents total number of credits needed

4.2.6.1.3 minorReqs : HashMap<Integer, ArrayList<Course>>
list of all the courses needed for minor

4.2.6.2. Functions of Minor

4.2.6.2.1 Minor()

4.2.6.2.2 getMinorTitle() : String
returns title of the minor

4.2.6.2.3 getCreditRequirement() : int
returns total number of credits needed for minor

4.2.6.2.4 getMinorReqs(): HashMap<Integer, ArrayList<Course>>
returns list of all courses needed for particular minor

4.2.7. (ABS)ValidPOSGenerator

4.2.7.1. Functions of (ABS)ValidPOSGenerator

4.2.7.1.1 generateValidPOSTable() : Hashtable<Integer, POS>
Generates valid plans of study for the students based on major, year, quarter and courses.

4.2.8. POSValidator

4.2.8.1. Attributes of POSValidator

4.2.8.1.1 validPOSGenerator : ValidPOSGenerator

4.2.8.1.2 validPOSTable : Hashtable <integer, POS>
Hash table that represents all the generated POS

4.2.8.2. Functions of POSValidator

4.2.8.2.1 POSValidator()

4.2.8.2.2 ValidatePOS(POS) : boolean
validates POS if all the requirements are met

4.2.8.2.3 ValidPOSGenerator(major, Minor) : ValidPOSGenerator
Generates valid POS for major and minor

4.2.9. RestrictionChecker

4.2.9.1. Functions of RestrictionChecker

4.2.9.1.1 checkRestrictions(POS, Year, Quarter, Course) : boolean
boolean function that checks all the restrictions

4.2.9.1.2 checkCreditLimit(POS, Year, Quarter, Course) : boolean
checks if credit limit is met and returns true or false

4.2.9.1.3 checkProreqs(POS, Year, Quarter, Course) : boolean
checks all the pre-requirements are met

4.2.9.1.4 checkCoReqs(POS, Year, Quarter, Course) : boolean
checks all the co-requirements are met

4.2.9.1.5 checkAvaivability(POS, Year, Quarter, Course) : boolean
Checks if courses is offered at particular quarter

4.2.9.1.6 checkCollegeRestrictions(POS, Year, Quarter, Course) :
Checks if all the college restrictions are met

4.2.9.1.7 checkLevelRestrictions(POS, Year, Quarter, Course) : Checks if all the level restrictions are met

4.2.9.1.8 checkOnlineRestrictions(POS, Year, Quarter, Course) : Checks if all the online restrictions are met

4.2.9.1.9 checkRetakeRestrictions(POS, Year, Quarter, Course): Checks if course can be retaken or not

4.2.10. POSReconfigurer

4.2.10.1. Attributes of POSReconfigurer

4.2.10.1.1 validator

4.2.10.1.2 restrictionChecker : RestrictionChecker
Checks if all the restrictions are met

4.2.10.1.3 graduationRequirementChecker :
Checks if POS leads to timely graduation

4.2.10.2. Functions of POSReconfigurer

4.2.10.2.1 addCourse(Course, POS, Year, Quarter) :boolean

4.2.10.2.2 moveCourse(Course, POS, YEar, Quarter) : boolean

4.2.10.2.3 removeCourse(Course) : boolean

4.2.10.2.4 validateReconfigure(POS) : boolean

4.2.11. POSManager

4.2.11.1. Attributes of POSManager

4.2.11.1.1 planOfStudy : POS

4.2.11.1.2 reconfigurer : POSReconfigurer

4.2.11.2. Functions of POSManager

4.2.11.2.1 POSManager()

4.2.11.2.2 savePOS(POS) : void

4.2.11.2.3 loadPOS(String) : POS

4.2.11.2.4 addCourse(Course, POS, Year, Quarter) : boolean

4.2.11.2.5 removeCourse(Course, POS, Year, Quarter) : boolean

4.2.11.2.6 moveCourse(Course, POS, Year, Quarter) : boolean

4.2.12. IO

4.2.12.1. **Attributes of IO**

4.2.12.1.1 instance : IO

4.2.12.2. **Functions of IO**

4.2.12.2.1 IO

4.2.12.2.2 getInstance() : IO

4.2.12.2.3 display(<T>) : void

4.2.13. Main

4.2.14. **Attributes of Main**

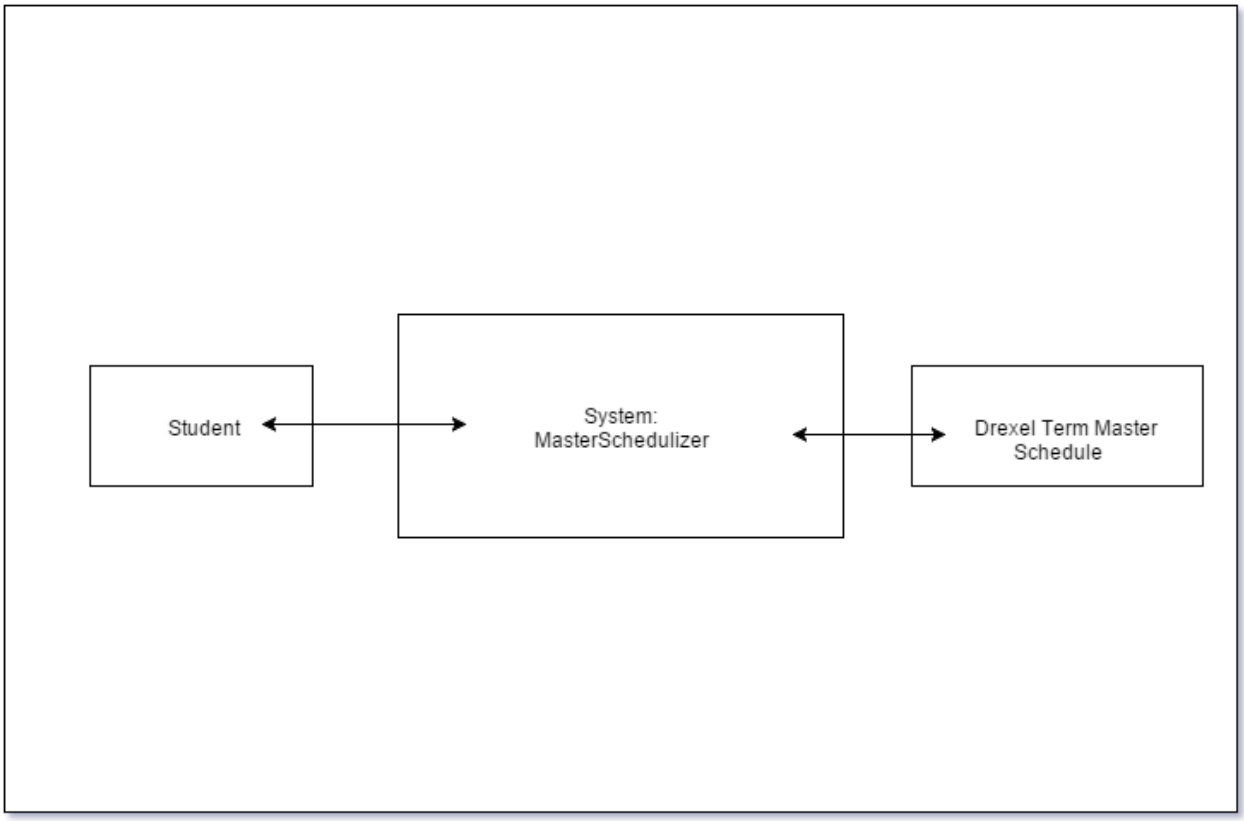
4.2.14.1 planManager : POSManager

4.2.15. **Functions of Main**

4.2.15.1 main(String []args) : void

5. FIGURES

5.1. System Boundaries



5.2. System Components

1	Master Schedulizer Application - Application for Plan of Study generation and editing
2	CSV Parser - Application for loading saved plans of study and saving new or edited plans of study
3	Database - Stores saved Plans of Study and a list of courses on a local database
4	Terminal Output - Prints messages to the user and displays a plan of study

5	CSV Output - Saves a modified or created plan of study to the local database

Figure 5.2 provides an initial list of system components. Components 1 and 2 are program code which support the major functional aspects of the system. The Master Schedulizer Application contains the core code for the performance of the main functional requirements. Component 3 stores all files and information for use by components 1 and 2. Components 4 and 5 are the output components. Component 4 displays messages to the user. Component 5 allows the user to save a plan of study to component 3.

5.3. System Overview Diagram

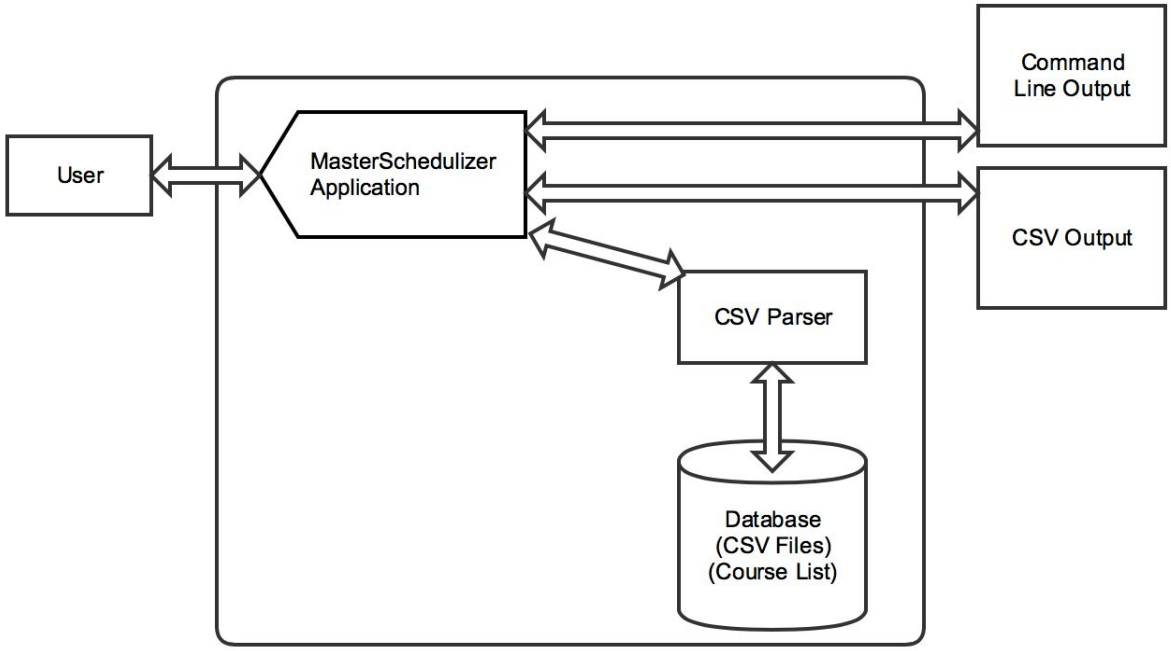
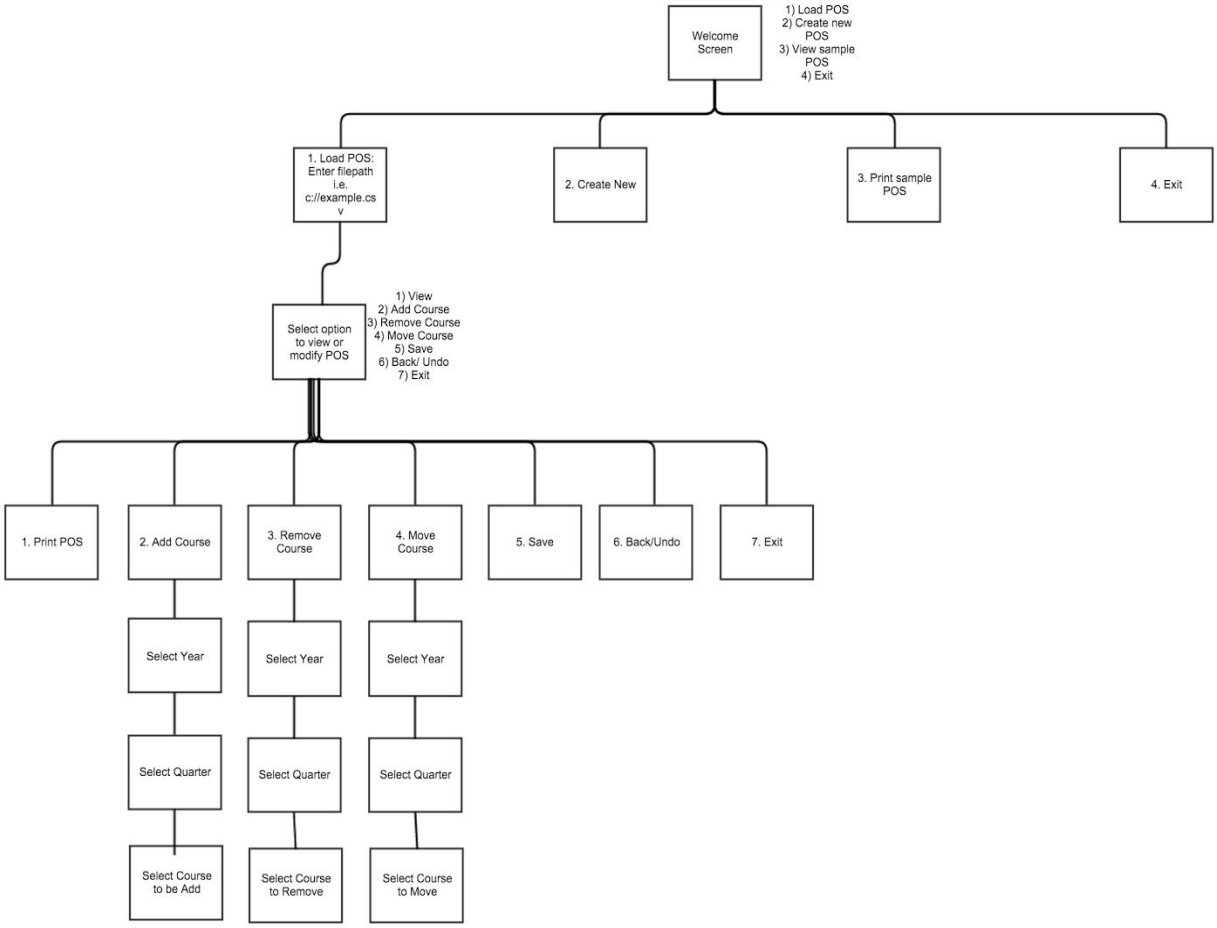


Figure 5.3 shows the major components of the system and their connections.

5.4. Screen Hierarchy Diagram



Interactive web viewer: http://www.gliffy.com/go/publish/9954537