Software Design Document

for

Computer Science Schedule Generating System

Version 1.0

Prepared by Jiaqi Yang, Runzhi Zhou, Zijun Liu, Tao Huang, Jieyu Ren

CWRU EECS393

October 9, 2019

Revision History

Date	Version	Description	Author(s)
1 Oct 2019	0.1	Initial Draft	Jiaqi Yang, Runzhi Zhou, Zijun Liu, Tao Huang, Jieyu Ren
3 Oct 2019	0.2	Initial Draft - Revision	Jiaqi Yang, Runzhi Zhou, Zijun Liu, Tao Huang, Jieyu Ren
5 Oct 2019	0.3	Assignment - Sequence Diagram	Jiaqi Yang, Runzhi Zhou, Zijun Liu, Tao Huang, Jieyu Ren
6 Oct 2019	0.4	Revision of flowchart and sequence diagram	Jiaqi Yang, Runzhi Zhou,

			Zijun Liu, Tao Huang, Jieyu Ren
8 Oct 2019	0.5	Final Draft	Jiaqi Yang, Runzhi Zhou, Zijun Liu, Tao Huang, Jieyu Ren
9 Oct 2019	1.0	Final Draft - Finalized	Jiaqi Yang, Runzhi Zhou, Zijun Liu, Tao Huang, Jieyu Ren

Table of Contents

1. INTRODUCTION

1.1 Purpose

This document is a high-level design document for the Computer Science Schedule Generating System(CSSGS), using a number of different architectural views to depict different aspects of the system. It is intended to capture and convey the significant architectural decisions that have been made on the system.

1.2 Scope

This program will be used as a computer science schedule generating system. Some of the key features of the system are the following. The website will collect BS student course information and import the information into the database, and automatically generate several recommended schedules for the students.

It will also enable the users to view the information of the course(professor, time and prerequisite), enable users to change the recommended courses and select the best one for them.

Lastly, the website will enable the administrator to have access to the database of the system - it will allow the administrator to update course information, delete the course from the list, read the course information and add the course into the list.

2. APPLICABLE DOCUMENTS

2.1 CSSGS Documents

Software Requirements Specification(SRS) for Computer Science Schedule Generating System, version 1.0, September 23, 2019.

2.2 NON-CSSGS Documents

None.

3. RESPONSIBILITIES

The requirements for the computer science schedule generating system include the following responsibilities:

R1. Checking and configuring the system

R2. Initiating system operation and monitoring system status

R3. Allowing users to register and log into the system

R4. Notifying users about the usage of the information collected

R5. Generating course schedules for users based on their inputs

R6. Handling subsystem failures

R7. Resuming operation after either controlled or uncontrolled shutdown

R8. Providing a user and an admin interface that supports system operation or system management

4. CLASS INTERFACES

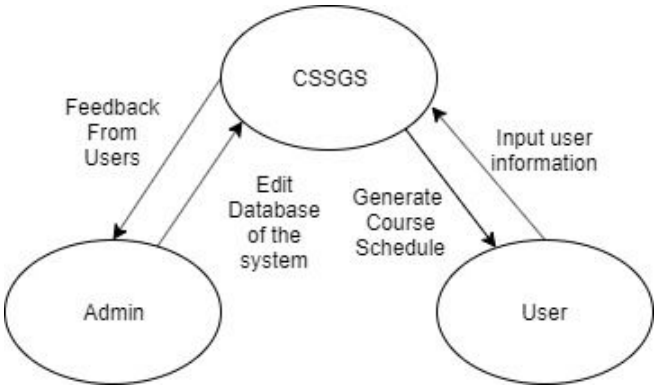

Figure 1
Context diagram for the first demo of the Computer Science Schedule Generating System

4.1 Class User

An instance of class User represents the unique user information used to log in the system and store the username and password in the account database.

4.1.1 Public Method GetUserName

getUserName()

The getUserName() method returns the user name stored in account database.

4.1.2 Public Method GetUserPassword

getUserPassword()

The getUserPassword() method returns the user password stored in account database.

4.1.3 Public Method CreateUserName

createUserName()

The createUserName() method appends a new username for the user and stores it in the account database.

4.1.4 Public Method CreateUserPassword

createUserPassword()

The createUserPassword() method appends a new password for the user and stores it in the account database.

4.1.5 Public Method CheckPassword

checkPassword(int password)

The checkPassword() method checks the user password using the account database. The user would enter the admin page if the password matches the admin password.

4.2 Class Admin

An instance of class Admin represents the admin's permission of editing the database of the system. The administrator will use the methods in it to add, delete, read and update the course

4.2.1 Public Method AddCourse

addCourse(Course)

The addCourse() method adds a new course to the course database.

4.2.2 Public Method DeleteCourse

deleteCourse(Course)

The deleteCourse() method removes the course from the course database.

4.2.3 Public Method UpdateCourse

updateCourseTime(Course,int time)

updateCourseProf(Course,String professor)

updateCourseInfo(Course, String info)

The updateCourseTime() method changes the time of the course in the course database.

The updateCourseProf() method changes the professor of the course in the course database.

The updateCourseInfo() method changes the information about the course in the course database.

4.2.4 Public Method ReadCourse

readCourse(Course)

The readCourse() method returns all the information(time, professor, specific information) about the course.

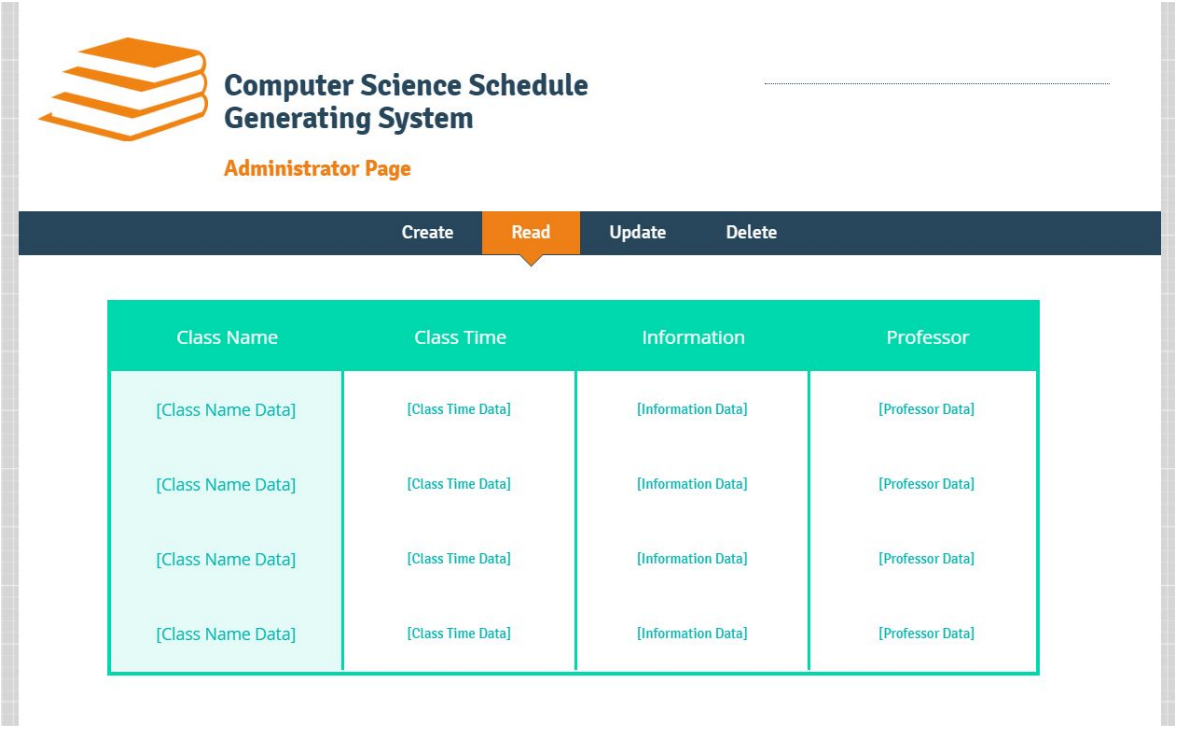

Figure 2
Concept graph of the administrator page for the first demo of the Computer Science Schedule Generating System

4.3 Class UserInfo

An instance of class UserInfo represents the collected user information used to generate the schedule. The system shall store the information in the user information database.

4.3.1 Public Method GetCoursesTaken()

This method should take a username as input and return a list of courses that he or she has taken.

4.3.2 Private Method GetTrack()

This method should take a username as input and return the track that he or she chose.

4.3.3 Private Method Display()

This method should display the courses and track the choice of a user.

4.3.4 Public Method CheckInformation()

This method should take a user choice as input (whether to update or not) and return a Boolean.

4.3.5 Public Method UpdateCoursesTaken()

This method should take a username and his or her choices as input and return the updated courses.

4.3.6 Public Method UpdateTrack()

This method should take a username and his or her choices as input and return the updated track.

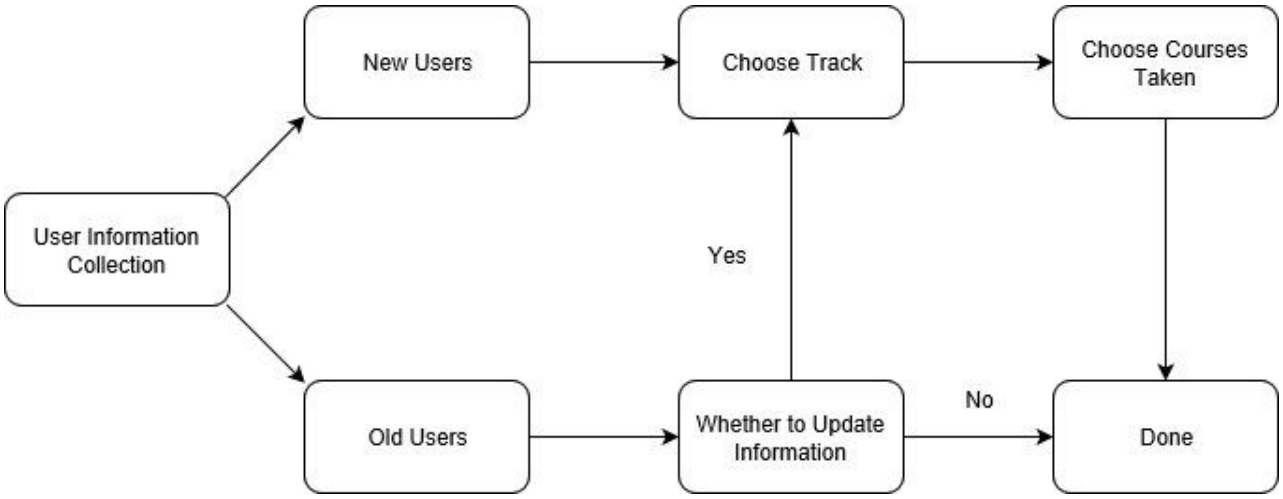

Figure 3
Flow Diagram of the user information collection process of the Computer Science Schedule Generating System

4.4 Class HighPriorityCourses

This class is responsible for generating several combinations/plans of high priority courses that the user needs to take during a semester. Users do not participate in this process until the end when they pick one plan from the plan list.

4.4.1 getUserInfo()

This method returns the user information that has just been entered on the User Information Collection page. User information includes: the student's intended degree (i.e., a Bachelor of Arts or a Bachelor of Science degree), the student's intended track (with its type as an enumeration class called Track) and the courses that this student has taken (with its type as an array of course id's).

Here class Track is a list of names that represent different types of CS tracks. (i.e., Software Engineering, Algorithms and Theory, Computer Systems, Networks and Security, Databases and Data Mining, Bioinformatics and Artificial Intelligence) and each course is represented by its primary key from the courses database instead of its course name. (Notice that the courses database does not just include CS courses)

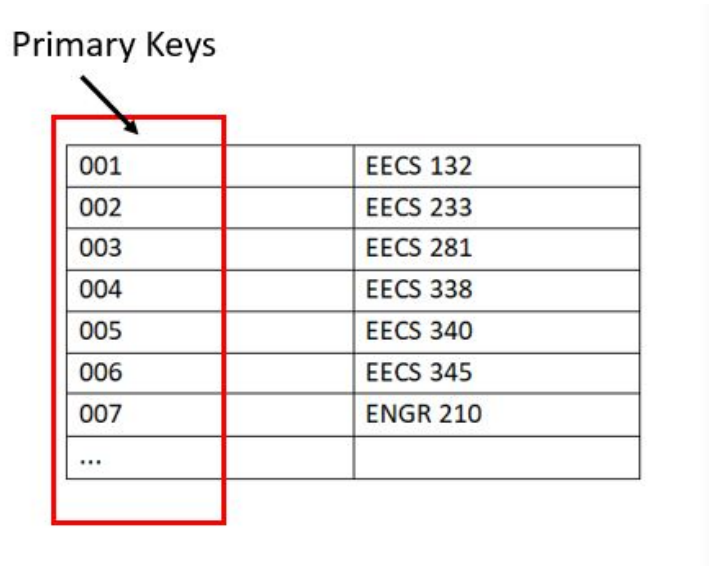

Figure 4

A sample list of primary keys and their corresponding course name

4.4.2 getHpCourses()

This method will return all the high priority courses. Such courses include courses from the engineering general education requirements (BS), Computer Science Major Requirements, Computer Science Core Requirements, Computer Science Depth Requirements, Computer Science Breadth Requirements, and Statistics Requirements. The return type is an array of Courses.

Note: Course is a class that we have defined that includes course id, code, name, time slot, information, prerequisites, type, and substitute id. (substitution id is used to distinguish substitute courses. For example, since MATH 122 and MATH 124 are the substitutes that can be used to fulfill the engineering general education requirements, these two courses will both have substitute id. Courses that do not have substitute courses will have a substitution id as -1).

See the table below to see the two Course's representing *EECS 393/493: Software Engineering.* Notice that since this course has two course code, we need to create two Courses to represent this course. Here we are only presenting one Course example. The table below is a sample of course with id 023 that represents EECS 393.

Course id (an integer)	023
Course code (a string)	EECS 393
Course name (a string)	Software Engineering
Time slot (a string)	MWF11:40-12:30

Information (a string)	Instructor: Andy Podgurski Course Location: Bingham 103 Course Description: Software engineering is both a synonym for the systematic use of well-founded software development practices and the name of the area of computer science and engineering that deals with the specification, design, implementation, validation, and maintenance of complex software systems and applications. This course covers the main elements of software engineering methodology.
Prerequisites (a string that represents the course code)	EECS 132
Type (an integer)	4 (Here type 4 represents a type that fulfills both depth requirements and breadth requirements.)
Substitution id (an integer)	-1

4.4.3 getEGERoptions (ArrayList<Integer> coursesTaken)

This method takes as input a list of courses that the student has taken and returns a list of courses that the student still needs to take to fulfill the engineering general education requirements. The way our system does it is to first obtain all of the engineering general education requirements courses, go through all the courses taken, and remove the ones that are on the whole requirement list.

4.4.4 getCSMRoptions (ArrayList<Integer> coursesTaken)

This method takes as input a list of courses that the student has taken and returns a list of courses that the student still needs to take to fulfill the CS major requirements. The algorithm works similar to that described in 4.4.3.

4.4.5 getCSCRoptions (ArrayList<Integer> coursesTaken)

This method takes as input a list of courses that the student has taken and returns a list of courses that the student still needs to take to fulfill the CS core requirements. The algorithm works similar to that described in 4.4.3.

4.4.6 getCSDRoptions (ArrayList<Integer> coursesTaken)

This method takes as input a list of courses that the student has taken and returns a list of courses that the student still needs to take to fulfill the CS depth requirements. The algorithm works similar to that described in 4.4.3.

4.4.7 getCSBRoptions (ArrayList<Integer> coursesTaken)

This method takes as input a list of courses that the student has taken and returns a list of courses that the student still needs to take to fulfill the CS breadth requirements. The algorithm works similar to that described in 4.4.3.

4.4.8 getSRoptions (ArrayList<Integer> coursesTaken)

This method takes as input a list of courses that the student has taken and returns a list of courses that the student still needs to take to fulfill the Statistics requirements. The algorithm works similar to that described in 4.4.3.

4.4.9 generatePlans (Track track, ArrayList<Integer> coursesTaken)

This method takes as input a list of courses that have been taken by the student and the intending track that the student has chosen. It returns a list of lists of high priority courses (each sub-list represents a plan consisting of courses that can be taken, all at the same time, during one semester).

To explain our algorithm, we need to first present our system's ranking for all the high priority courses: see the diagram on the next page for the detailed priority ranking.

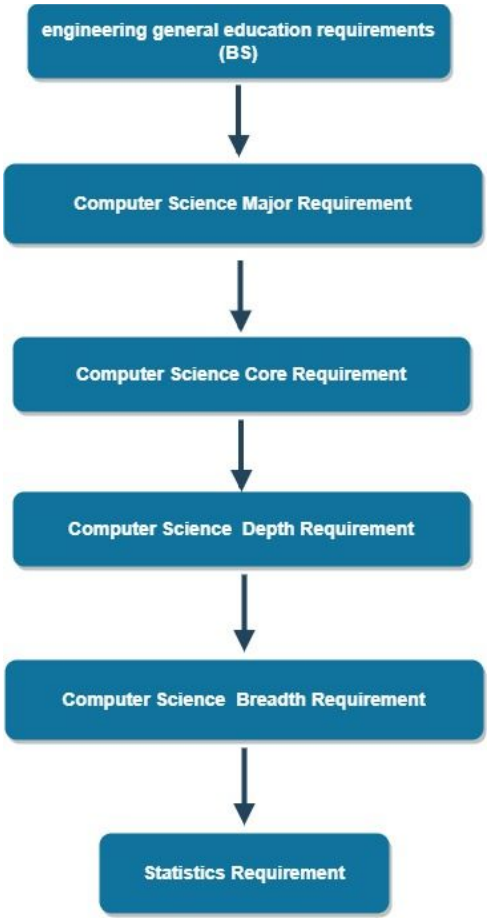

Figure 5

The diagram of the detailed priority ranking of high priority courses

As specified in the diagram, the most prioritized courses are courses that fulfill the engineering general education requirements (for BS students). Since these courses usually build the foundation for (or are the prerequisites of) the more advanced courses, we want to make sure students take these classes first. For a similar reason, the second prioritized group of courses is from the Computer Science major requirement, which builds the foundation for the CS major requirement courses (our third prioritized group).

After the CS major requirement courses, our system values most of the CS core requirement courses. For the six courses from this group, we again want students to complete all of them

except EECS 395, *Senior Project in Computer Science*, before considering other types of CS courses.

The fourth prioritized course type is the depth courses. We ranked it before the breadth courses (the fifth prioritized type of courses) because several breadth courses are also in the depth lists.

As for the statistics requirement, only BS students are required to take one of the five listed statistics courses, which will be offered every semester. In addition, statistics courses are not the prerequisites of any CS course. Therefore, it's the least prioritized type of course.

Now we can talk about the algorithm of generating plans of high priority courses. The algorithm works as follows:

1) Our system starts off from the top of the ranking tree and with an empty list that will save different semester plans. For each level the system checks whether or not the requirement has been fulfilled.

1a) If a requirement has been fulfilled, the system proceeds to the next level of requirement.

1b) If not, our system generates possible combinations of courses. Specifically, our system firstly uses methods described in 4.4.3 to 4.4.8 to obtain the list of courses needed to fulfill the current requirement. Then our system will use a nested loop to go through every

plan from the plan list and adds courses from the requirement list after making sure that there's no time conflicts.

2) After checking the bottom level of the priority tree, the system can return the plan list.

4.5 Class LowPriorityCourses

An instance of class LowPriorityCourses represents the low priority courses an user-selected, which is necessary for generating the final schedule.

4.5.1 Public Method GetLPCourses()

This method should return a list of all low priority courses.

4.5.2 Private Method Display()

This method should display the low priority courses the user selected.

4.5.3 Public Method GetSelection()

This method should take a username as input and return the low priority courses he or she has selected.

4.6 Class FinalSchedule

An instance of class FinalSchedule represents the final schedule generated by the system.

4.6.1 Public Method GetSelectedCourses()

This method should take selected high-priority courses to plan and selected low-priority courses as input and return a list of courses.

4.6.2 Private Method Display()

This method should display the final schedule generated.

4.6.3 Public Method GetFeedback()

This method should take a Boolean as input (choose to save or not) and save it in the user information database if the input is true.

5. DATABASE DESIGN

Two databases are constructed in the system.

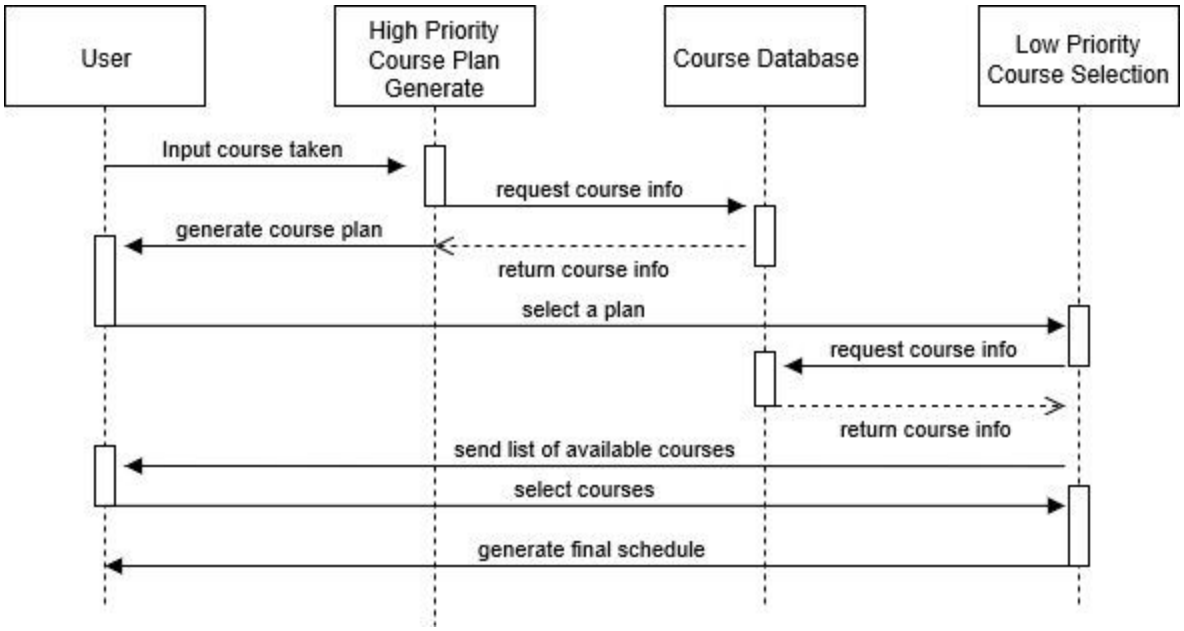

Figure 6
Sequence Diagram of the Computer Science Course Schedule Generating System

5.1 Course Database

The course database is constructed to store information on all the computer science courses given in a specific semester. It only contains a single entity. The ER diagram for the course database is shown below.

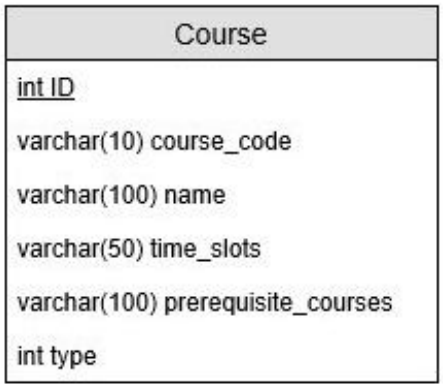

Figure 7
Course Database ER diagram

5.1.1 Course

The course entity is the only entity in the course database. It stores information about all the courses in the current semester. The database is managed by the administrators and updated via the administrator interface. The attributes of the entity are described below.

5.1.1.1 ID

This ID attribute is an integer. It is the primary key of the Course entity and therefore used to identify a course. The attribute has automatically incremental property.

5.1.1.2 course_code

This course_code attribute is varchar with a maximum length of 10 characters. It stores the code of the course that is widely used by Case Western Reserve University. For example, the course code for Software Engineering is “EECS393”.

5.1.1.3 name

This name attribute is varchar with a maximum length of 100 characters. This attribute stores the name of the course. For example, the name for EECS 393 is “Software Engineering”.

5.1.1.4 time_slots

This time_slots attribute is varchar with a maximum length of 50 characters. This attribute stores the weekly schedule of the course in a specific char format. The software will have a class to decode the stored time_slots and then pass to the inner classes. For example, if a course is scheduled for Monday, Wednesday, Friday from 1 pm to 1:50 pm, then the time_slots will be stored as “MWF13:00-13:50”.

5.1.1.5 prerequisite_courses

This prerequisite_courses attribute is varchar with a maximum length of 100 characters. This attribute stores the course code(s) of a course’s prerequisite courses. For example, the prerequisite_courses for EECS 340 Algorithm will be stored as “EECS233, EECS302”.

5.1.1.6 type

This type of attribute is an integer. This attribute represents the type of course. The types of course requirements, specified in Appendix A: Acronym and Terminology Dictionary, are

general, core, breadth, depth, technical elective. Some courses are able to satisfy multiple requirements but not both, such as the EECS 345 Programming Language Concept, which can satisfy either breadth requirement or depth requirement.

Type	Course Property
0	Satisfy General Requirement
1	Satisfy Core Requirement
2	Satisfy Breadth Requirement
3	Satisfy Depth Requirement
4	Satisfy Breadth and Depth Requirement
5	Satisfy Elective Requirement

5.2 User Information Database

The user information database is constructed to store the information of the users. It contains two entities and one relation. The ER diagram for the database shown below.

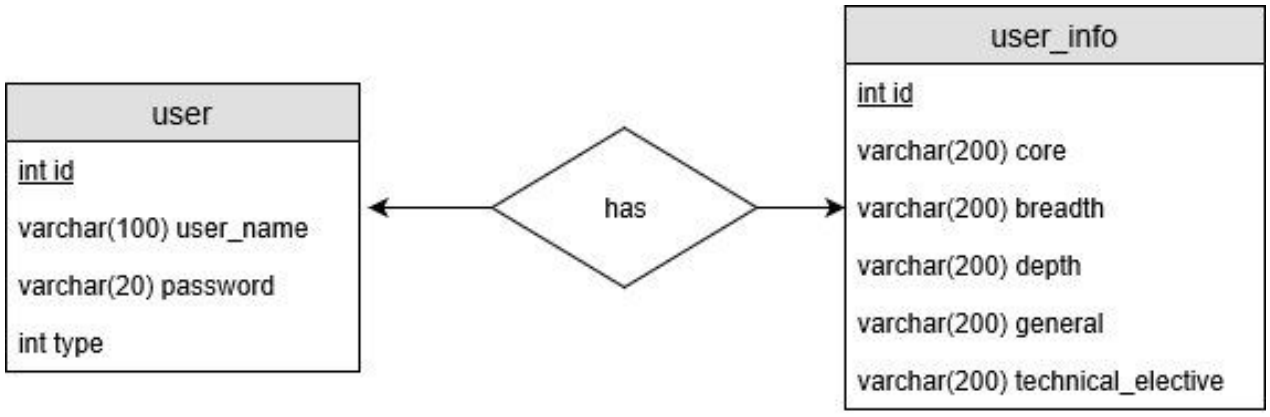

Figure 8
ER diagram for the user information database

5.2.1 user

The user entity stores information about a user account, including id, user name, password, and account type. The attributes of the entity are described below.

5.2.1.1 ID

This ID attribute is an integer. It is the primary key of the user entity and therefore used to identify a user. The attribute has automatically incremental property.

5.2.1.2 username

This username attribute is varchar with a maximum length of 100 characters. It is unique for every user and used to login in and retrieve the account information in the system.

5.2.1.3 password

This password attribute is varchar with a maximum length of 20 characters. This attribute represents the password for the corresponding username. It needs to be provided by the user when logging into the system.

5.2.1.4 type

This type attribute is an integer. It is used to identify the account type: whether the account is an administrator account or user account.

Type	Account Property
0	User Account
1	Administrator Account

5.2.2 has

The has relation stores information about the relationship between user entity and user_info entity. The attributes of the relation are described below.

5.2.2.1 user_id

This user_id attribute is an integer. It is the primary key of the user entity and therefore used to identify a user.

5.2.1.1 user_info_id

This user_info_id attribute is an integer. It is the primary key of the user_info entity and therefore used to identify a user_info.

5.2.3 user_info

5.2.3.1 id

This user_info_id attribute is an integer. It is the primary key of the user_info entity and therefore used to identify a user_info.

5.2.3.2 core

This core attribute is varchar with a maximum length of 200 characters. It records all the courses of type “core” that the associated user has taken. It is stored in the form of binary code to represent courses. For example, the binary code “111100” will represent that the student takes EECS 132, EECS 233, EECS 281, EECS 302, but has not taken EECS 340 and EECS 395.

EECS 132	EECS 233	EECS 281	EECS 302	EECS 340	EECS 395
1	1	1	1	0	0

5.2.3.2 breadth

This breadth attribute is varchar with a maximum length of 200 characters. It records all the courses of type “breadth” that the associated user has taken. It is stored in the form of binary code to represent courses. The design of the binary code will be similar to the core requirement’s design. The order of the course will be in the increasing order of the course code’s numeric part.

5.2.3.2 depth

This depth attribute is varchar with a maximum length of 200 characters. It records all the courses of type “depth” that the associated user has taken. It is stored in the form of binary code to represent courses. The design of the binary code will be similar to the core requirement’s design. The order of the course will be in the increasing order of the course code’s numeric part.

5.2.3.2 general

This general attribute is varchar with a maximum length of 200 characters. It records all the courses of type “general” that the associated user has taken. It is stored in the form of binary code to represent courses. The design of the binary code will be similar to the core requirement’s design. The order of the course will be in the increasing order of the course code’s numeric part.

5.2.3.2 technical_elective

This technical_elective attribute is varchar with a maximum length of 200 characters. It records all the courses of type “technical elective” that the associated user has taken. It is stored in the form of binary code to represent courses. The design of binary code will be similar to core requirement’s design. The order of the course will be in the increasing order of course code’s numeric part.

6. INSPECTION REPORT

Name	Inspection Findings	Resolution
Jiaqi Yang	• Redundant table in part 4.4.2 with similar examples but two one-page table • Some terminologies' definitions are not clear and not specified in the context • Some diagrams are too large and they couldn't be displayed right after the context that uses them.	• Remove one of the redundant table • Add Appendix A:Acronym and Terminology Dictionary • Make the diagrams smaller
Zijun Liu	• Ignore the situation that a course could have a substitute course such as MATH 201 and MATH 307 in the method getEGERoption. • Ambiguous description of types in user entity and course entity • Missing introduction to part 4.4, HighPriority class	• Add a new variable substitute id in the course class • Add a table that contains the type number and its corresponding meaning for each entity • Add the introduction
Tao Huang	• Missing information in the sequence diagram • Wrong data type in the database diagram	• Appended missing information to the diagram • Corrected the wrong data type
Jieyu Ren	• Descriptions of class methods not compatible • Missing descriptive information for each figure in the report	• Went through the whole section and fixed the issue • Add description and number below each figure in the report
Runzhi Zhou	• Miss several important methods in User Class • Some grammar issues	• Added the missing methods in the user class • Fixed grammar issues

Appendix A: Acronym and Terminology Dictionary

- **Track** - Track entered by a user includes 6 tracks listed as Areas of Computer Science Depth Requirement in Case Western Reserve University official bulletin [1].
- **Substitute Courses** - Substitute courses include courses in General Requirements, Core Requirements, Depth Requirements, Breadth Requirements that can substitute for each other.
- **Course Priority** - Course priority, defined by Computer Science Course Generating System, ranks the different requirements in the decreasing order of General Requirements, Core Requirements, Depth Requirements, Breadth Requirements, SAGES Requirements, Technical Requirements.
- **High priority courses** - High priority courses include courses in General Requirements, Core Requirements, Depth Requirements, and Breadth Requirements.
- **Low priority courses** - Low priority courses include courses in Technical Requirements.
- **General Requirements** - General Requirements include 10 courses listed as Major Requirements and 1 course listed as Statistics Requirement in Case Western Reserve University official bulletin [1].
- **Core Requirements** - Core Requirements include 6 courses listed as Computer Science Core Requirements in Case Western Reserve University official bulletin [1].
- **Depth Requirements** - Depth Requirements include 4 courses of the user-input track listed as Computer Science Depth Requirements in Case Western Reserve University official bulletin [1]. If a course can be used to meet both Depth Requirements and

Breadth Requirements, it will be preferentially counted toward Depth Requirements if the user's Depth Requirements have not been satisfied.

- **Breadth Requirements** - Breadth Requirements include 5 out of 7 courses listed as Computer Science Depth Requirements in Case Western Reserve University official bulletin [1].
- **SAGES Requirements** - SAGES Requirements include 2 out of 3 SAGES courses from USNA, USSO, USSY listed as University Seminar Requirements in Case Western Reserve University SAGES Requirements bulletin [2].
- **Technical Requirements** - Technical Requirements include at least 3 courses from Group 1 and at most 2 courses from Group 2 listed as List of Approved Technical Electives in Case Western Reserve University official bulletin [1].

Appendix B: Application Flowchart

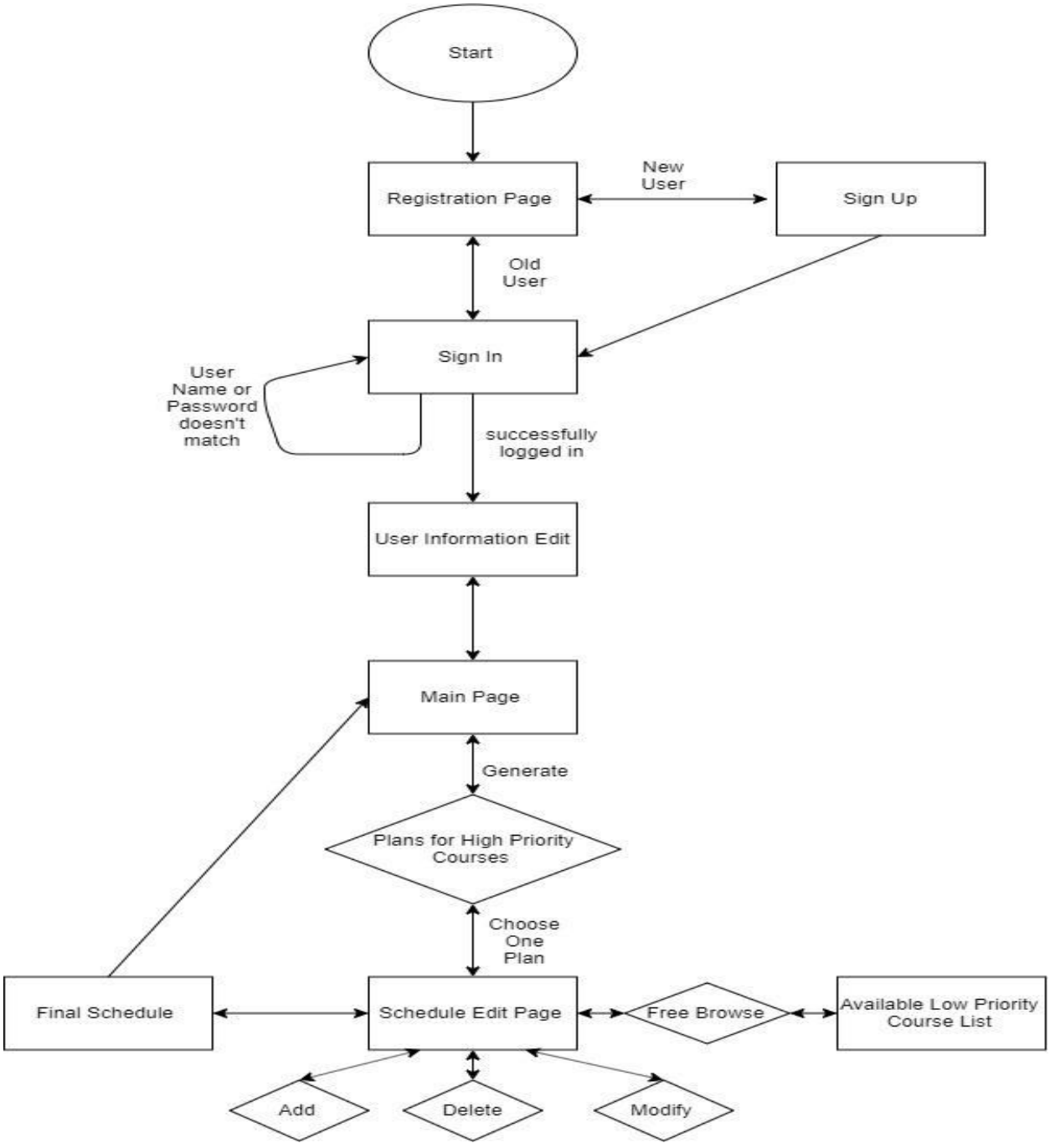

Figure 9

Application Flowchart for the first demo of the Computer Science Schedule Generating System