Project Specifications

This document contains the project specifications that will be shared.

On Start

1. Data Manager reads courses from the UofT's timetable API: https://timetable.iit.artsci.utoronto.ca/api/20229/courses?org=&code=CSC
2. Data Manager reads any existing timetables or degree plans from files
3. Command line:
    a. "ProgramName.java" to start the program
    b. Prompts user to login
    c. After login, prompt user to enter "open [*one of the 3 commands below*]":
        i. "dashboard" - open dashboard view
        ii. "timetable" - open timetable view
        iii. "degree" - open degree planner view
    d. if nothing entered after "open", enter dashboard view by default

While Running

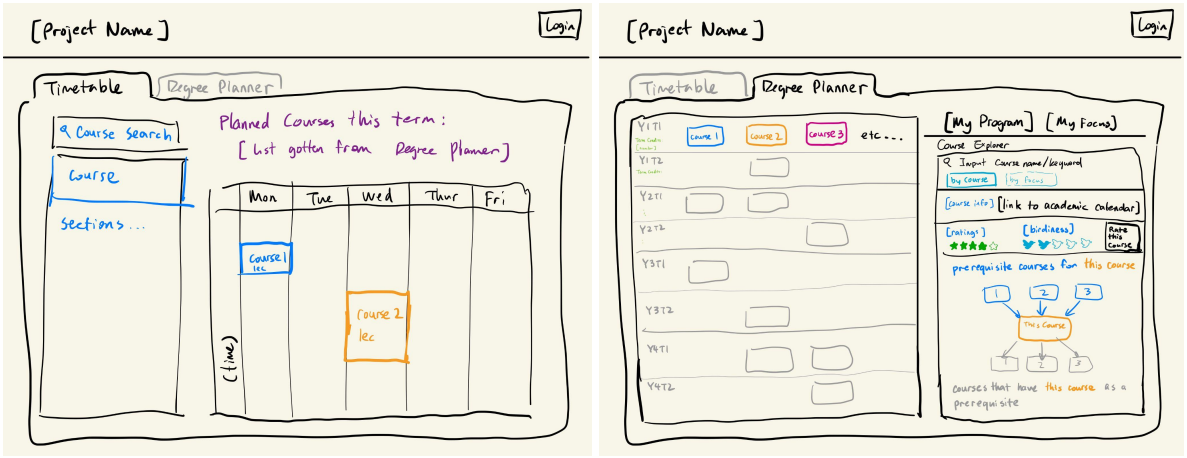
1. Timetable Planner tab:
    a. Calendar, with courses + sections. clicking on one shows info about this lec/tut/prac section
    b. List of planned courses this term
    c. Course Search tab where the user can search, view and add courses sections to the calendar by [weekday] and [time slot]
2. Degree Planner tab:
    a. Like UofT Degree Explorer but the term order is flipped
    b. Course Explorer: Lets users search for courses.
        i. If user clicks "by Course" option:
            1. When keyword entered, show single course
            2. Displays information with link to academic calendar
            3. Displays unofficial information like rating, birdiness, etc
            4. Prerequisite tree: shows what courses are prereq for the selected course, and what courses have this selected course as prereq
        ii. If user clicks "by Focus" option:
            1. When keyword entered, show a list of courses that are required for this focus
            2. clicking on a course in the list shows the course's information, exactly like if the user searches this course using the "by Course" option
    c. user can add [course] to the degree plan (the table in the reference below with "Y1T1, Y1T2, …") by [year] and [term]
    d. POTENTIAL EDITS: Using the word "year" is pretty unclear. UofT defines a student's year on a **session** basis and how many credits they've completed. So instead of Y1T1, we should use 20229, 20231, 20235, 20237, 20239,

20241 OR 20229-F, 20229-S, 20235-F, 20235-S. Also, beware that session code Y courses take up two spots.

On Exit

1. Data Manager saves the latest information into files
2. Logs the user out

Reference for UI:



Modules to be created

There are different modules to be implemented.

Term Planner

1. The timetable app itself. Users can select courses and their lecture sections, and a timetable will be visually generated.
2. Same as 1, but users do not select lecture sections - the app does it for the user.
3. Course List to be planned in this term, this list is determined by Degree Planner

Degree Planner

4. Planner parallel to the Term Planner, but focused on 4 year course planning like UofT's Degree Explorer. The planner does prerequisite checks, and may be used to calculate a student's GPA.
5. Course choices according to program requirements
6. Course choices according to focus requirements and recommendations
7. Must support specialist, major major, and major minor credit requirements
8. Course recommendations for certain interests (e.g. game design, AI, natural language processing, etc)
9. Tree generator: generates a tree graph of courses for a specific program or interest.

Data Manager

10. Gets courses from the UofT websites, also reads and writes files containing user's information

Specific information about modules

Timetable creator

A timetable tool written in Java. Users choose a course by course code (e.g. `CSC110Y1F`), then choose its lecture session `LEC0101`, tutorial session if applicable, and so on. Users do so multiple times. Then, they can choose to generate a timetable. The timetable looks similar to that in Acorn.

Timetable generator

Same as timetable creator except the user does not state lecture sessions. This program then picks the lecture and tutorial slots for the student.

Degree Planner

A tool similar to UofT's degree explorer's planner module, but the order is reversed. So it starts from year 1 and goes down to year 4.

It first requests the courses a user has taken. Then, users can state courses they want to take during a year. The degree planner will then state whether there are any issues (prerequisites, corequisites, exclusions, 7 100-level credit limit violated, and so on).

Differences from UofT's degree explorer:

- Aim to make the full 4 year plan all at once, instead of 1 year at a time like the current degree explorer
- flip the order of the years. (ie. Fall 2021 at the top of the grid, Summer 2025 at the bottom)
- Has course recommendations for the full 4 years according to the student's interests, such as machine learning, system design, etc. Then choose courses according to constraints including "mandatory for program", "mandatory for focus", "recommended for program/focus", and "elective".
- Also has more information like rating by students, birdiness, whether it's recommendations by students and professors, etc.

Tree Generator

Similar to students asking on the UofT reddit about what courses they should take given a certain interest. Includes mandatory courses and recommended courses.

The individual courses should indicate whether they're **Mandatory for Program** (ie MAT137 for CS), **Mandatory for Focus** (ie CSC485 for Computational Linguistics), **Recommended for Program/Focus** (ie COG250 for AI), or **Elective**

Data Manager

1. Gets courses from the UofT websites

2. Reads and writes files storing the user's term plans and degree plans
3. Manages storing the courses and their information, including user ratings

The CRC Card diagram

Course data looks like what is provided in the link:

https://timetable.iit.artsci.utoronto.ca/api/20229/courses?org=&code=CSC

Entities

CourseList *Contains information on every course that has or will be offered ever in UofT.* Superclasses: Subclasses: Implements: Type: Entity	
Responsibilities	Interacts with the following classes
• Stores all courses based on course code (Question: should MAT135H1F and MAT135H1S be treated as different courses? Should different sessions (20229/20235) have its own set of independent courses?) • Has methods to obtain course information given course code.	• Course ○ And its inheritors

StudentData *Contains all the data a student can have.* Superclasses: Subclasses: Implements: Type: Entity	
Responsibilities	Interacts with the following classes
• Holds all personal information relating to a student that a regular account wouldn't store, such as student ID • Holds information about the student's current degree, which is contained in the Degree class. • Holds information on the courses the student has planned for the next session (as of the time phase 0 has been submitted, the target session to plan for is 20229)	• Course ○ And its inheritors • Degree • Account • SessionPlannedCourses

Course
Any course that can be described with the regular expression
`[A-Z]{3}[0-4][0-9]{3}[H|Y][0159]|[A-Z]{3}[A-D][0-9]{2}[H|Y][3]`
Example: CSC110Y1
Note that graduate courses are **not** supported. Note that the above Regex does not support unspecified transfer credits.

Superclasses:
Subclasses:
Implements:
Type: Entity

Responsibilities	Interacts with the following classes
● Course Code ● Course Name ● Course Description ● Prerequisites ● Distribution Requirements ● Breadth Requirements ● Sections ○ Gives us a history of every lecture ever offered by this course. This means that given a session (20229), F/S/Y, and lecture number, we should be able to extract a meeting out of it. This may be updated continuously. More attributes from unofficial sources (ie community/users) ● Course Rating ● Birdiness ● Mandatory (for program / focus) ● Recommended (for program / focus) ● etc…	● Degree

Meeting
Represents a LEC, TUT, or PRA. I may plan to make this class abstract and create classes representing LECs, TUTs, and PRAs.
Example: CSC110Y1-F-20229-LEC0101

Superclasses: Course
Subclasses:
Implements:
Type: Entity

Responsibilities	Interacts with the following classes

We should be able to reproduce the text in this screenshot given an instance of this class.  Stores: ● Type : LEC/TUT/PRA ● Number: 0101, 0201, 5101 ● Meeting times: MO10-12,TU10-12 ● Delivery mode ● Enrollment controls (P/R1/R2…)	● Course

SessionPlannedCourses (Alias: Session) *A session is a time block between September-April OR May-August, and can be represented by the ID 20229 / 20235 / 20239 / 20245 / …* *Session number breakdown: the "2022" in 20229 is the year, and the "9" stands for September, the start month of the session. Note that while UofT considers 202X1 to be the Winter sessions, the timetable API does not consider it to be like that.* ***This particular class stores information on the courses a student has picked to take for a session.*** Superclasses: Subclasses: Implements: Type: Entity	
Responsibilities	Interacts with the following classes
● Course List → ○ Includes F, S, and Y courses ● Term Total Credit → ● F credit count → ● S credit count → ● GPA Calculator (+ term GPA) ● Year code (20229 for FW 2022-2023)	● Course

Degree *Stores a student's existing progress* Superclasses: Subclasses: Implements: Type: Entity	
Responsibilities	Interacts with the following classes

• Planned Course List
  ○ Should be separated into sessions (20229…)
• Completed Course List
  ○ Should be separated into sessions
• Program (specialist, major, minor)
• Focus
• Credit So far
• Total Credit
• GPA Calculator
• Start Session
	• Student
	• Course
	• Term

Program *An abstract class representing a program. A program is anything you can add in ACORN as a program - at least something that has a program code like ASSPE1689.* Superclasses: Subclasses: Implements: Type: Entity	
Responsibilities	Interacts with the following classes
• Program name → • Program code → • Description → • Required Courses → • Recommended Courses →	• Course ○ And its inheritors • Degree

These classes may inherit from the Program class. They will not have any additional methods and attributes other than an enum allowing us to tell whether a program is a major, minor, specialist, or focus.

- Major
- Minor
- Specialist
- Focus

Use cases

TimetableGenerator *Use case to generate some sort of timetable given a list of sections a student might want to take (e.g. give [CSC110Y1-F-20229-LEC0101, CSC111H1-S-LEC0101] and I need a timetable.)* Superclasses: Subclasses: CompletedCoursesModifier, PlannedCoursesModifier + its subclasses Implements: Type: Use case

Responsibilities	Interacts with the following classes
• generateTimetable(courses: a collection of courses) • Allow conflicts? Y/N - to be specified by the user. If conflicts are not allowed, timetable generation fails if no option but to have conflicts	Dependencies: • Course

CourseModifier (Abstract) *Use case to specifically modify courses at least somewhere to be determined by its subclasses.* Superclasses: Subclasses: CompletedCoursesModifier, PlannedCoursesModifier + its subclasses Implements: Type: Use case	
Responsibilities	Interacts with the following classes
• addCourse() • removeCourse()	Dependencies: • Course • Degree

CompletedCoursesModifier *Use case to specifically modify completed courses in a Degree instance.* Superclasses: Subclasses: Implements: Type: Use case	
Responsibilities	Interacts with the following classes
• addCourse() • removeCourse()	Dependencies: • Course • Degree

PlannedCoursesModifier *Use case to specifically modify planned courses in a Degree instance.* Superclasses: Subclasses: Implements: Type: Use case	
Responsibilities	Interacts with the following classes

• session (stores the session it should modify) These methods below will have bulk options. • addCourse() • removeCourse()	Dependencies: • Course • Degree

ModifyPlannedStrict *Same as its superclass, but enforces prerequisites and exclusions. Maybe corequisites given that two courses are not corequisites of each other.* Superclasses: ModifySessionCourses Subclasses: Implements: Type: Use case	
Responsibilities	Interacts with the following classes
Contains all responsibilities in its superclass. • user/account/degree information ○ Stores information relating to the student. It should store completed course information alongside other student information.	Dependencies: • Course • Degree

A class that constructs the following above so that the controller does not need to access entities

Controllers

DegreeController *Controller which tells use cases to do "something"* Superclasses: Subclasses: Implements: Type: Controller	
Responsibilities	Interacts with the following classes

• Options to do the following:
  ○ Add completed courses to degree
  ○ Add pursued courses to degree
  ○ Take and ask to present information about the current student's degree
	Dependencies:
	• CourseModifier + its subclasses
	• Degree + any student related information

○ Get information about a course	

Presenters

Presenter *Presents information onto the command line and determines the text that gets presented* Superclasses: Subclasses: Implements: Type: Presenter	
Responsibilities	Interacts with the following classes
● startView() ○ Prints a message to a screen for users who are logged out. ● printAndAskPrompt() ○ For logged in users, based on which "window / state" a logged-in user is in, prints out something and asks for user input. This method returns what the user wrote. ● dashboardView() ○ Prints the message passed into the method and requests for user input, which it returns. ● timetableView() ○ switches the current "view" to the timetable ● degreePlannerView() ○ switches the current "view" to the degree planner These methods ask the user to type in something and returns a string or an array of strings based on what the user typed. ● enterCompletedCourse() ○ Prompts for a completed course and returns it. ● enterPlannedCourse() ○ Prompts for a planned course and returns it. All these methods affirm that an action has been done to a user. It prints to the user whether the following action was successfully enacted to the user. ● addCompletedCourses()	DegreeController

• addPlannedCourses()	