

Design Report - Deliverable

CS 487

Team B

- Mayank Bansal

- Robert Judka

- Paul Myers

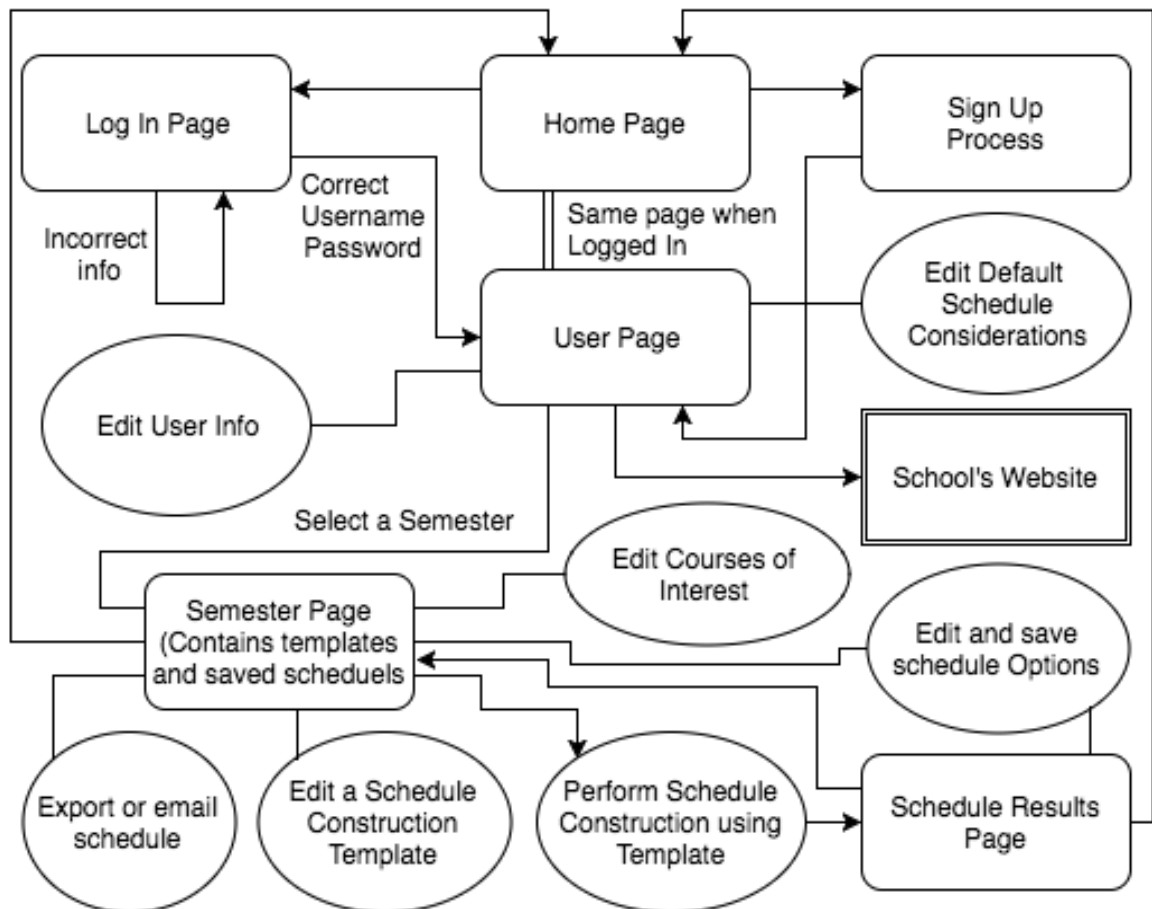
- Chanyu Wu

High Level Design

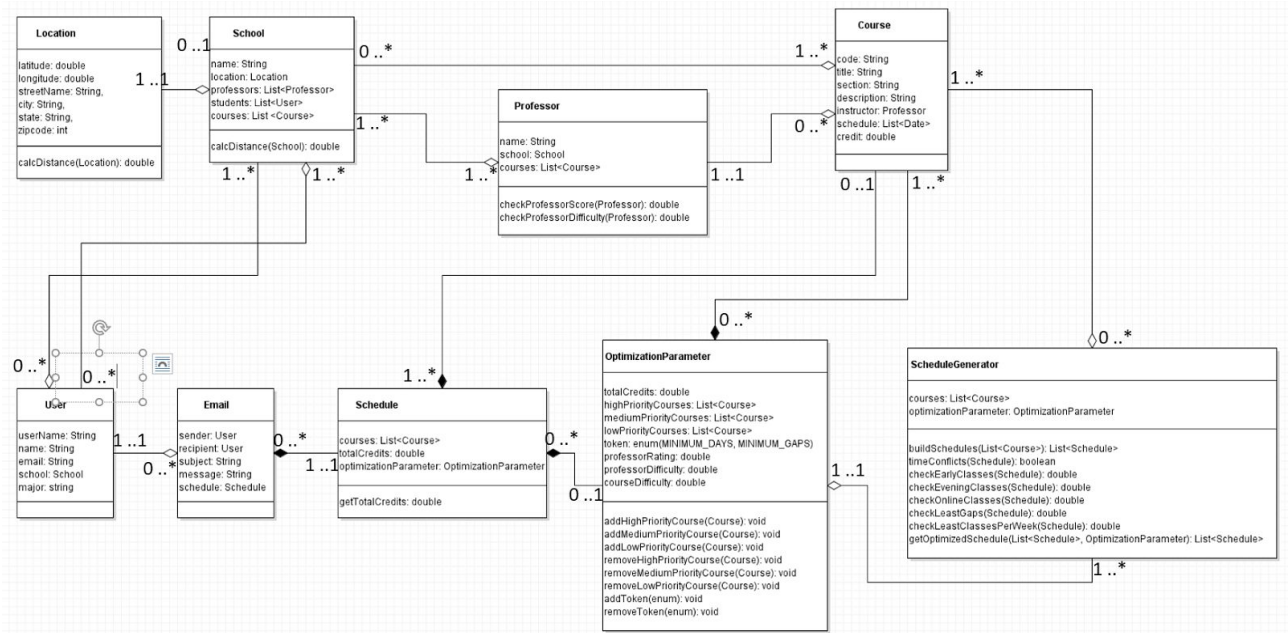
1. Data Table Design

Class	Attributes
USER	User name (string), Real name (string), email (string), school (school), major (string), year (int), address (location), scheduleCons, Semester(semester[]), User history (User History)
SCHEDULE CONS.	Credit hours (int), time of day pref (time.time), time of day unprof (time.time), RMPI score range (double.double), (double.double), Work hours (time.time)[][], days of week (string[]), locations (location[]), gaps (boolean)
SEMESTER	Season (string), year (int), start, end (date.date), Sched. tem. templates(template[]), (schedule[]), Titles of courses of interest (String[])
TEMPLATE	Schedule considerations (Schedule cons.), course priorities (string[] ordered)
COURSE	(reference to course through SWI, All fields acquired through it), Title (string), location (location), major (string), start, end times (time.time), end times (time.time), days of week (string[]), professor (professor)
DATE	month (int), day (int), year (int)
TIME	hour (int, 0-23), minute (int, 0-60)
LOCATION	City (string), State (string), Zip (int), Name (string)
PROFESSOR	(reference to professor through RMPI, All fields through it), Name (string), School (school), Average Rating (double), Average Difficulty (double)
SCHEDULE	Courses (course[]), days (string[])
SCHOOL	semesters sched (date[]), School Website Interface(SWI), Courses (course[]), professors(professor[]), campuses (location[])
SWI (School Website Interface)	Generates courses, Courses generated (course[])
RMPI(RateMyProfessor interface)	Generates professors, Professors generated (professor[])
USERHISTORY	searches (search[]), returned schedules (schedule[])
SEARCH	user(user), date and time (date.time), template(template), school(school)

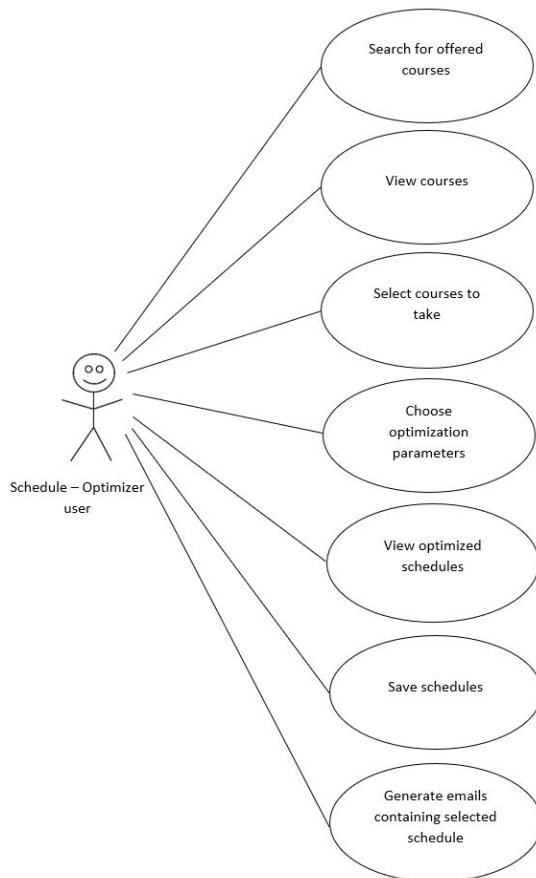
2. State Transition Diagram



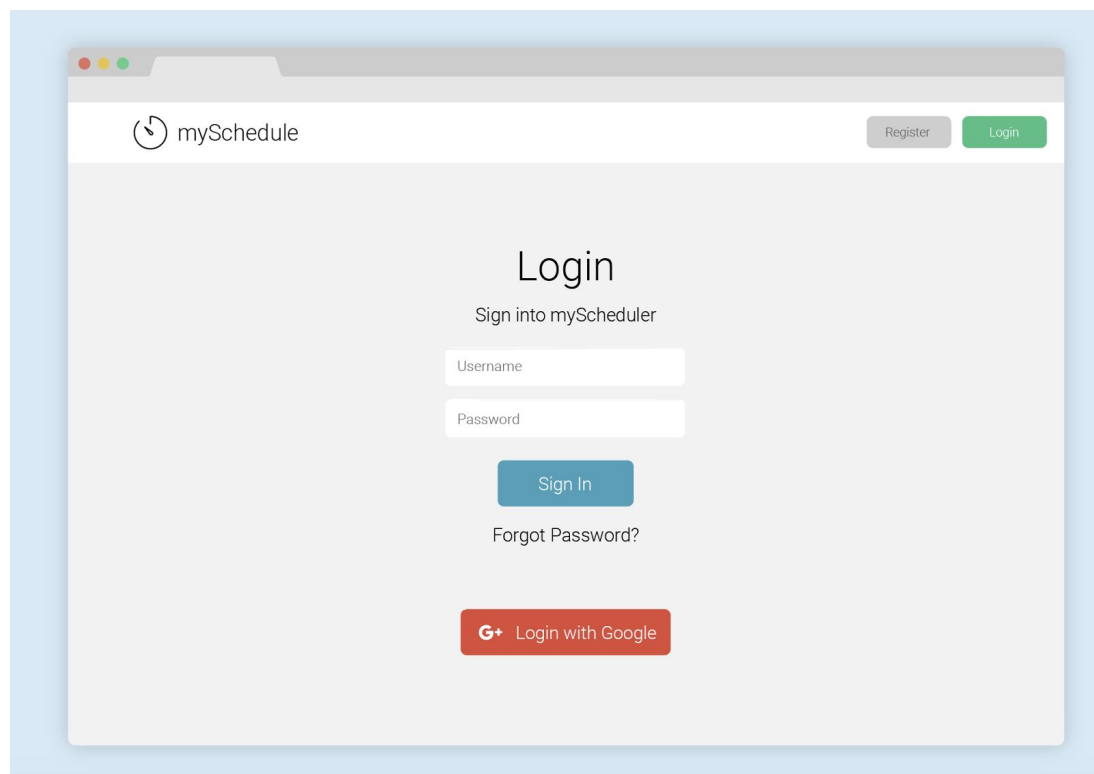
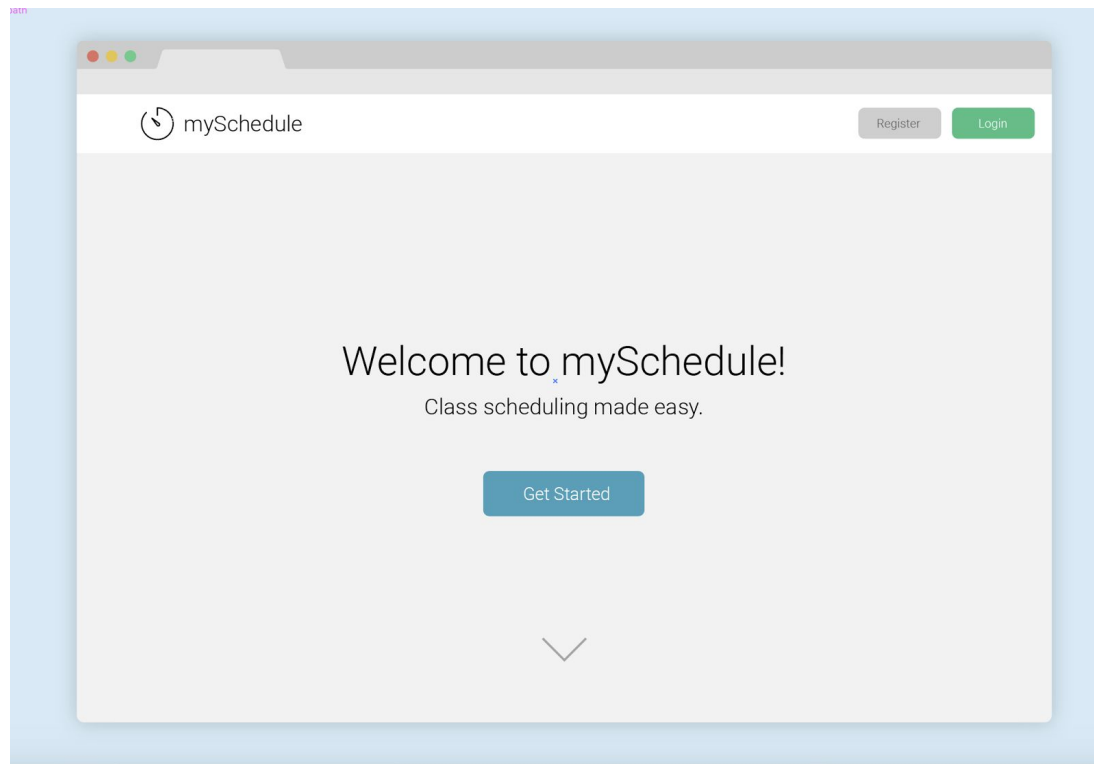
3. Class Association Diagram



4. Use Case Diagram



UI Prototypes



mySchedule

RegisterLogin

Register

Sign up for myScheduler

Select College

▼


Select Major

▼


Select Year

▼

Sign Up


 Sign up with Google

mySchedule

Welcome, Robert JudkaLog Out

Dashboard

My Account



Robert Judka

rjudka@hawk.iit.edu

Illinois Institute of Technology

B.S. in Computer Science | 4th Year

Account Settings

My Schedules

You haven't made any schedules.

0

×

View Schedules

Make Schedule

Professor Search

Search Now

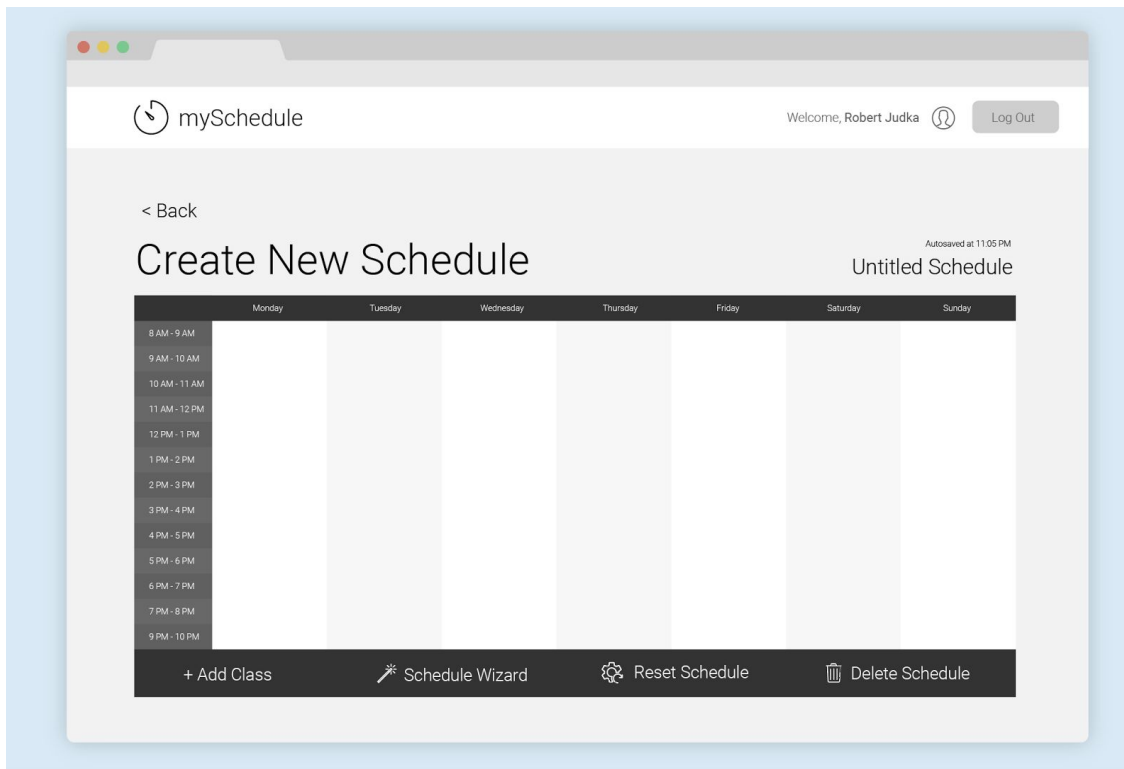
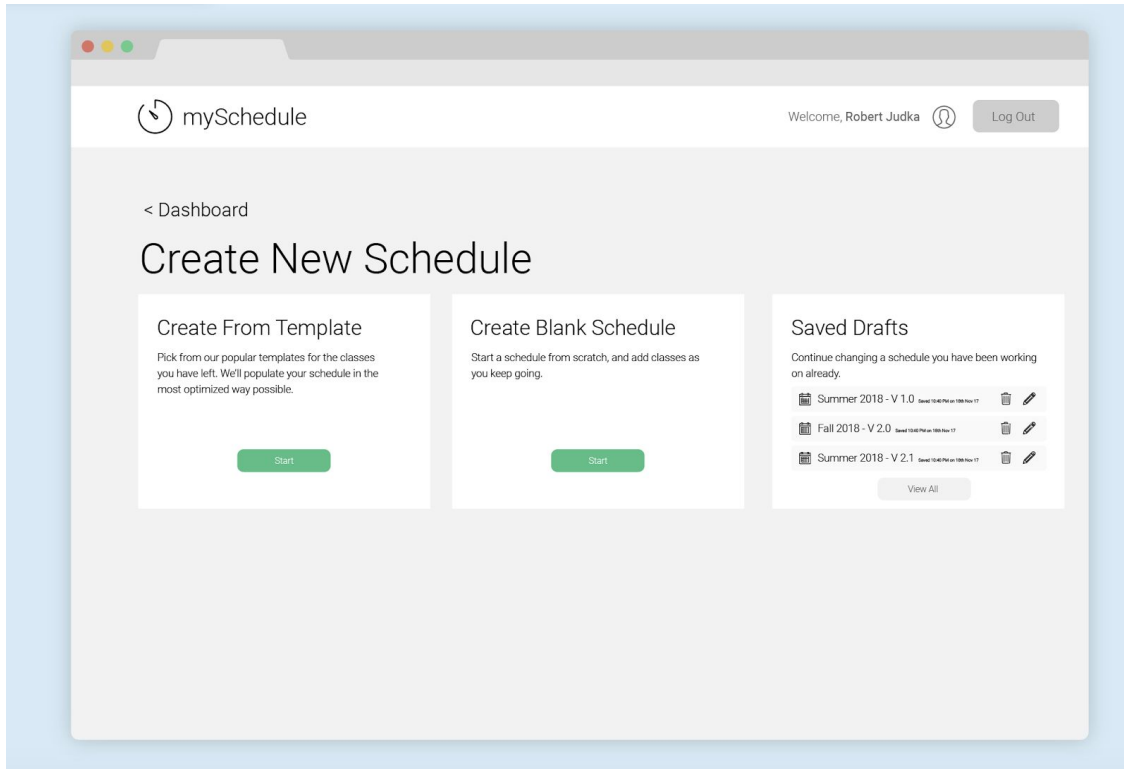
Class Search

Search Now

Degree Progress

75% Done

View Progress



mySchedule

Welcome, Robert Judka

Log Out

[< Back](#)

Schedule Wizard

Autosaved at 11:05 PM

Untitled Schedule

How many credits do you want to take?

3-66-99-1212-1512-1818-21Auto

What should the difficulty level be?

EasyAverageHardVery HardAuto

What times do you prefer? Select Multiple

Early MorningLate MorningAfternoonEveningNightAuto

Optimizations

☐ Minimize Gaps

☐ Maximize Gaps

☐ Popular Classes

☐ High Rated Professors

Generate

mySchedule

Welcome, Robert Judka

Log Out

[< Back](#)

Create New Schedule

Autosaved at 11:05 PM

Fall 2018 - Finish to Graduate

18 Credit Hours

	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
8 AM - 9 AM	CS 330 8:35 AM - 9:50 AM		CS 330 8:35 AM - 9:50 AM		CS 330 8:35 AM - 9:50 AM		
9 AM - 10 AM							
10 AM - 11 AM		MATH 474 10:00 AM - 11:15 AM		MATH 474 10:00 AM - 11:15 AM			
11 AM - 12 PM							
12 PM - 1 PM	CS 411 11:25 AM - 12:40 PM		CS 411 11:25 AM - 12:40 PM				
1 PM - 2 PM							
2 PM - 3 PM							
3 PM - 4 PM							
4 PM - 5 PM							
5 PM - 6 PM							
6 PM - 7 PM							
7 PM - 8 PM	IPRO 497 6:25 PM - 9:05 PM		PS 332 6:25 PM - 9:05 PM				
9 PM - 10 PM							

+ Add Class

Schedule Wizard

Reset Schedule

Delete Schedule

Key Features

```
////////////////////////////////////////
//////// Building Schedules //////////
////////////////////////////////////////

schedules = [] // list of all possible schedules

// find all class lists for credits required
// only schedules with no time conflicts
// classes: list of all classes chosen
build_schedules(classes):
    for i in range(0, classes.length):
        for schedule in classes.combinations(i):
            if total_credits(schedule) == credits_required:
                if time_conflicts(schedule) == False:
                    schedules.add(schedule)

///// HELPER FUNCTIONS FOR build_schedules /////

// calculates total credits for a schedule
total_credits(schedule):
    credits = 0
    for class in schedule:
        credits += class.credits
    return credits

// finds time conflicts for a schedule
time_conflicts(schedule):
    classes_slots = []
    for class in schedule:
        classes_slots.add(class.times)
    for classes sorted by start date/time in classes_slots:
        current_end_time = current.class.end_time
        next_start_time = next.class.start_time
        if current_end_time > next_start_time:
            return True
    return False
```

```
////////////////////////////////////////
////////// Find Optimized Schedules //////////
////////////////////////////////////////
```

```
// find schedule with highest optimization score
// only schedules that have no time conflicts and meets total credits required
// schedules: list of all possible schedules
// options: contains the list of classes seperated by required/optional orded by importance and
           list of selected options ordered by importance
```

```
get_optimized_schedules(schedules, optimizations):
    scored_schedules = [] // stored as (score, schedule)
    for schedule in schedules:
        score = 0
        for every schedule.classes in optimizations.required:
            score += optimizations.required.level
        for every schedule.classes in optimizations.optional:
            score += optimizations.optional.level
        for option in optimizations.options:
            score += switch(option.type):
                case early_classes: check_early_classes(schedule) * option.level
                case evening_classes: check_evening_classes(schedule) * option.level
                case online_classes: check_online_classes(schedule) * option.level
                case professor_score: check_professor_score(schedule) * option.level
                case professor_difficulty: check_professor_difficulty(schedule) * option.level
                case least_gaps: check_least_gaps(schedule) * option.level
                case least_classes_per_week: check_least_classes_per_week(schedule) *
option.level
            case default: 0
        scored_schedules.add(score, schedule)
    return scored_schedules sorted by score
```

```
// HELPER FUNCTIONS FOR get_optimized_schedules //
```

```
// counts how many classes are considered morning classes (start no later than 1:50pm)
// returns average as a value 0-1
```

```
check_early_classes(schedule):
    early_classes = 0
    for every schedule.class.start_time <= 1:50pm:
        early_classes += 1
    return early_classes / schedule.length
```



```

// counts how many classes are considered evening classes (start no earlier than 5:15pm)
// returns average as a value 0-1
check_evening_classes(schedule):
    evening_classes = 0
    for every schedule.class.start_time >= 5:15pm:
        evening_classes += 1
    return evening_classes / schedule.length

// counts how many classes have online sections
// returns average as a value 0-1
check_online_classes(schedule):
    online_classes = 0
    for every schedule.class.has_online:
        online_classes += 1
    return online_classes / schedule.length

// gets the total score of all professors through Rate My Professor
// returns average as a value 0-1
check_professor_score(schedule):
    total_score = 0
    for every schedule.class:
        total_score += RateMyProfessor(schedule.class.professor).score // referencing Rate My
Professor interface
    return total_score / (schedule.length * 5) // score out of 5

// gets the total difficulty of all professors through Rate My Professor
// returns average as a value 0-1
check_professor_difficulty(schedule):
    total_difficulty = 0
    for every schedule.class:
        total_difficulty += RateMyProfessor(schedule.class.professor).score
    return 1 - (total_difficulty / (schedule.length * 5)) // difficulty score out of 5

// counts number of gaps between classes
// returns average as a value 0-1
check_least_gaps(schedule):
    gaps = 0
    for every schedule.class sorted by start date/time:
        if (current.schedule.class.end_time + 10 min) != next.schedule.class.start_time:
            gaps += 1
    return 1 - (gaps / schedule.length)

```

```
// counts number of days classes are during a week
// returns average as a value 0-1
check_least_classes_per_week(schedule):
    days = Mon:0, Tues:0, Wed:0, Thurs:0, Fri:0, Sat:0
    for every schedule.class:
        days[schedule.class.day] += 1
    return all(days.not_none) / days.length
```