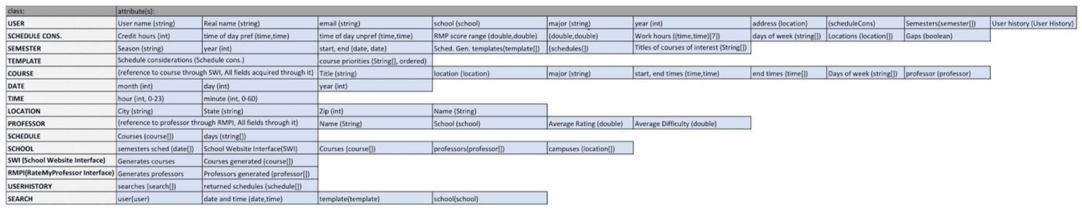
Design Report - Deliverable
CS 487
Team B
- Mayank Bansal
- Robert Judka
- Paul Myers
- Chanyu Wu

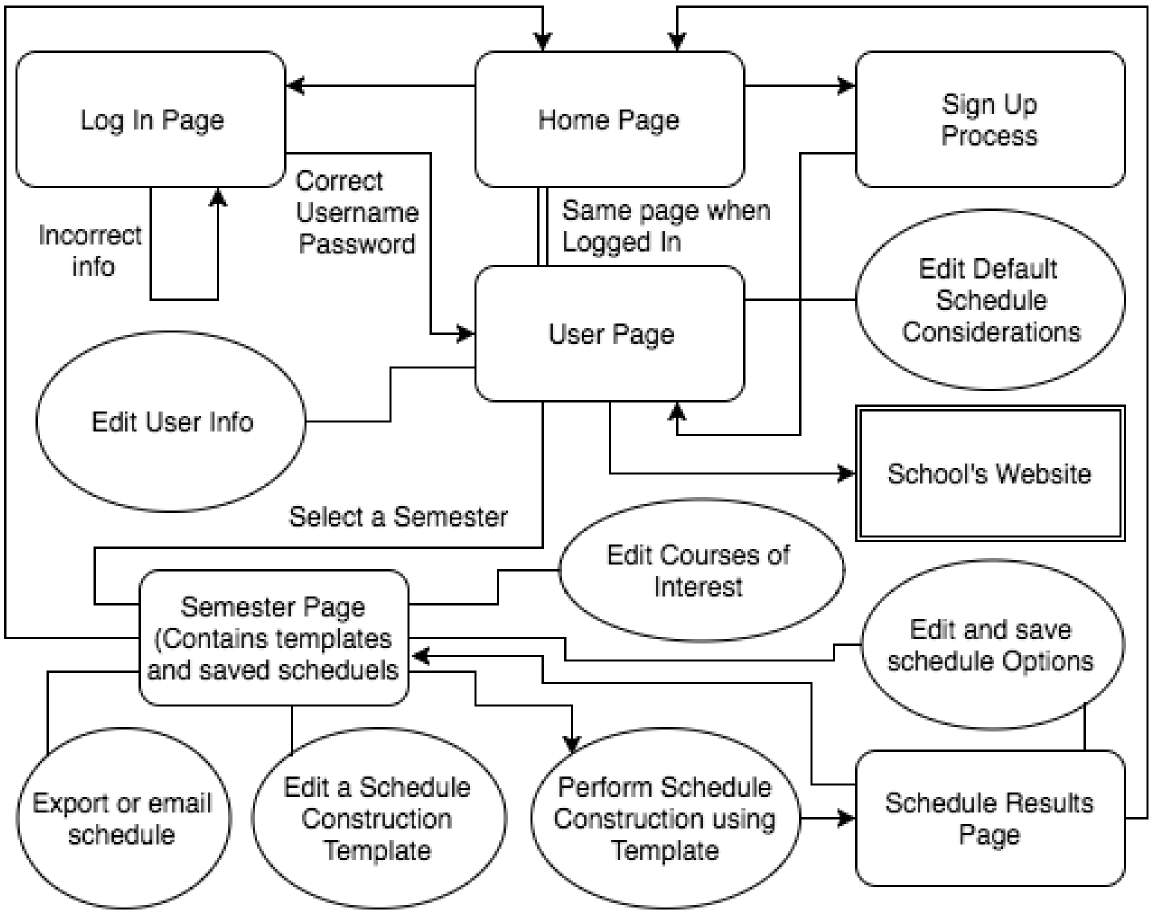
High Level Design

1. Data Table Design

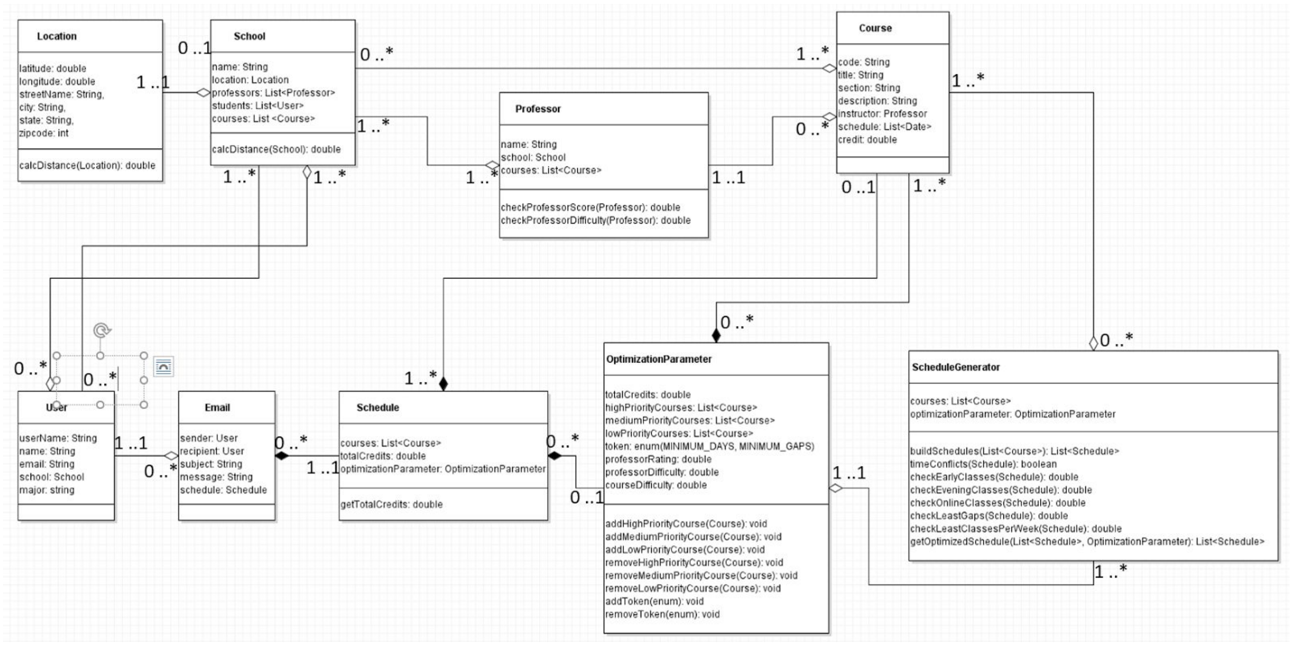


class:	attribute(s):									
USER	User name (string)	Real name (string)	email (string)	school (school)	major (string)	year (int)	address (location)	(scheduleCons)	Semesters(semester[])	User history (User History)
SCHEDULE CONS.	Credit hours (int)	time of day pref (time,time)	time of day unpref (time,time)	RMP score range (double,double)	(double,double)	Work hours ((time,time)[7])	days of week (string[])	Locations (location[])	Gaps (boolean)	
SEMESTER	Season (string)	year (int)	start, end (date, date)	Sched. Gen. templates(template[])	(schedules[])	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[])	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)						

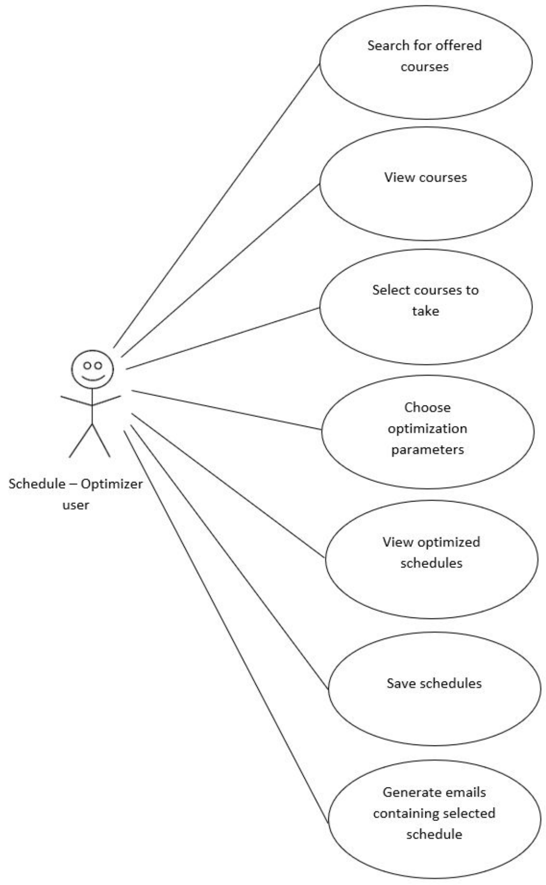
2. State Transition Diagram


Log In Page
Home Page
Sign Up Process
Incorrect info
Correct Username Password
Same page when Logged In
User Page
Edit Default Schedule Considerations
Edit User Info
School's Website
Select a Semester
Edit Courses of Interest
Semester Page (Contains templates and saved scheduels
Edit and save schedule Options
Export or email schedule
Edit a Schedule Construction Template
Perform Schedule Construction using Template
Schedule Results Page

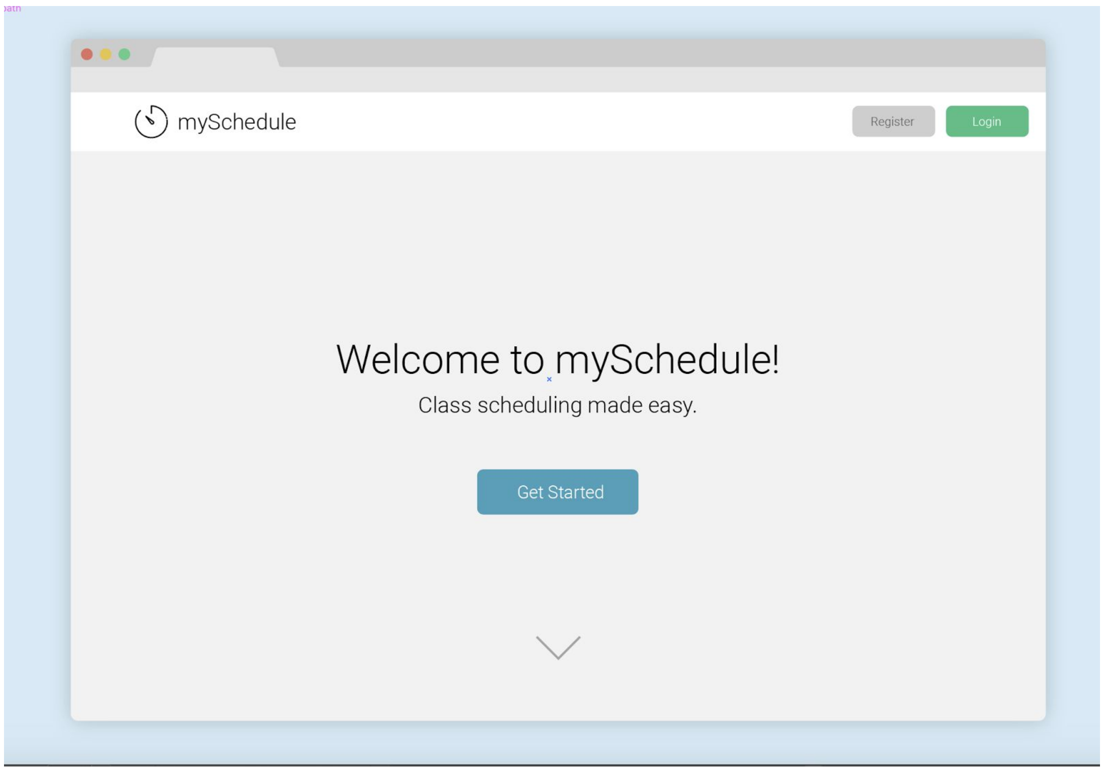
3. Class Association Diagram


Location
latitude: double
longitude: double
streetName: String,
city: String,
state: String,
zipcode: int
calcDistance(Location): double
School
name: String
location: Location
professors: List<Professor>
students: List<User>
courses: List <Course>
calcDistance(School): double
Professor
name: String
school: School
courses: List<Course>
checkProfessorScore(Professor): double
checkProfessorDifficulty(Professor): double
Course
code: String
title: String
section: String
description: String
instructor: Professor
schedule: List<Date>
credit: double
User
userName: String
name: String
email: String
school: School
major: string
Email
sender: User
recipient: User
subject: String
message: String
schedule: Schedule
Schedule
courses: List<Course>
totalCredits: double
optimizationParameter: OptimizationParameter
getTotalCredits: double
OptimizationParameter
totalCredits: double
highPriorityCourses: List<Course>
mediumPriorityCourses: List<Course>
lowPriorityCourses: List<Course>
token: enum(MINIMUM_DAYS, MINIMUM_GAPS)
professorRating: double
professorDifficulty: double
courseDifficulty: double
addHighPriorityCourse(Course): void
addMediumPriorityCourse(Course): void
addLowPriorityCourse(Course): void
removeHighPriorityCourse(Course): void
removeMediumPriorityCourse(Course): void
removeLowPriorityCourse(Course): void
addToken(enum): void
removeToken(enum): void
ScheduleGenerator
courses: List<Course>
optimizationParameter: OptimizationParameter
buildSchedules(List<Course>): List<Schedule>
timeConflicts(Schedule): boolean
checkEarlyClasses(Schedule): double
checkEveningClasses(Schedule): double
checkOnlineClasses(Schedule): double
checkLeastGaps(Schedule): double
checkLeastClassesPerWeek(Schedule): double
getOptimizedSchedule(List<Schedule>, OptimizationParameter): List<Schedule>
0 ..1
1 ..1
0 ..*
1 ..*
1 ..*
1 ..*
1 ..*
1 ..1
0 ..*
1 ..*
0 ..1
1 ..*
0 ..*
0 ..*
0 ..*
1 ..*
0 ..*
1 ..1
0 ..*
0 ..*
1 ..1
0 ..*
0 ..1
1 ..1
1 ..*

4. Use Case Diagram

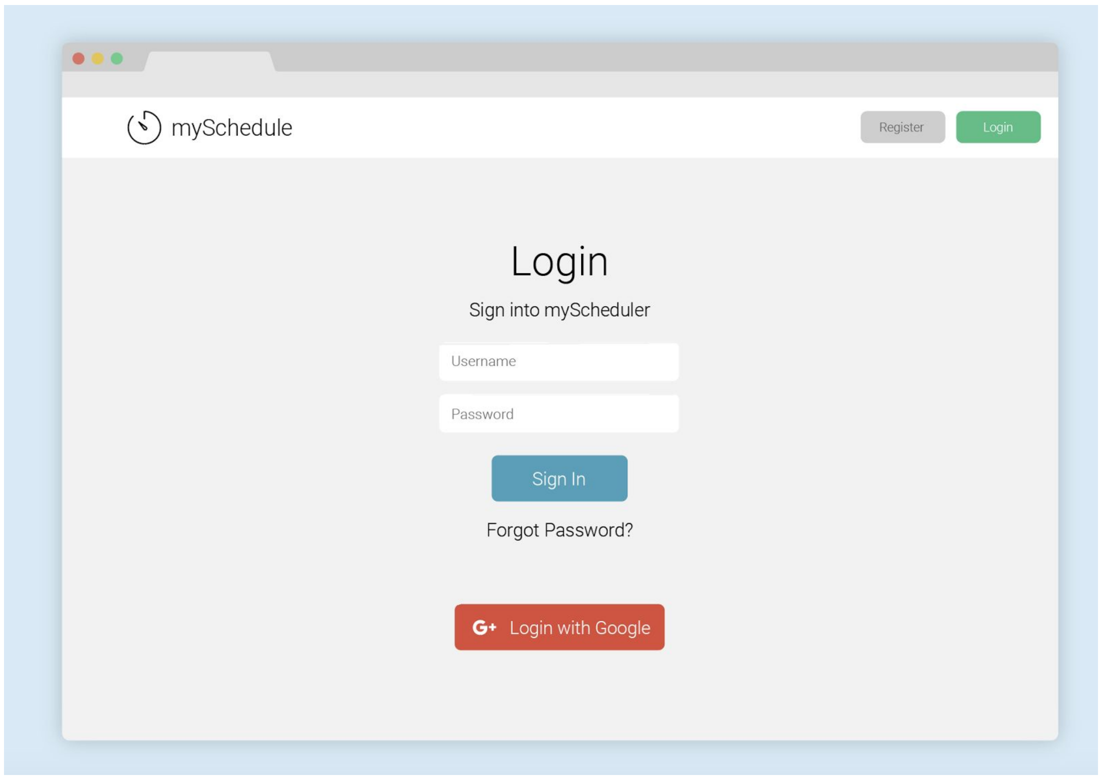

Search for offered courses
View courses
Select courses to take
Choose optimization parameters
View optimized schedules
Save schedules
Generate emails containing selected schedule
Schedule – Optimizer user

UI Prototypes



Welcome to mySchedule!

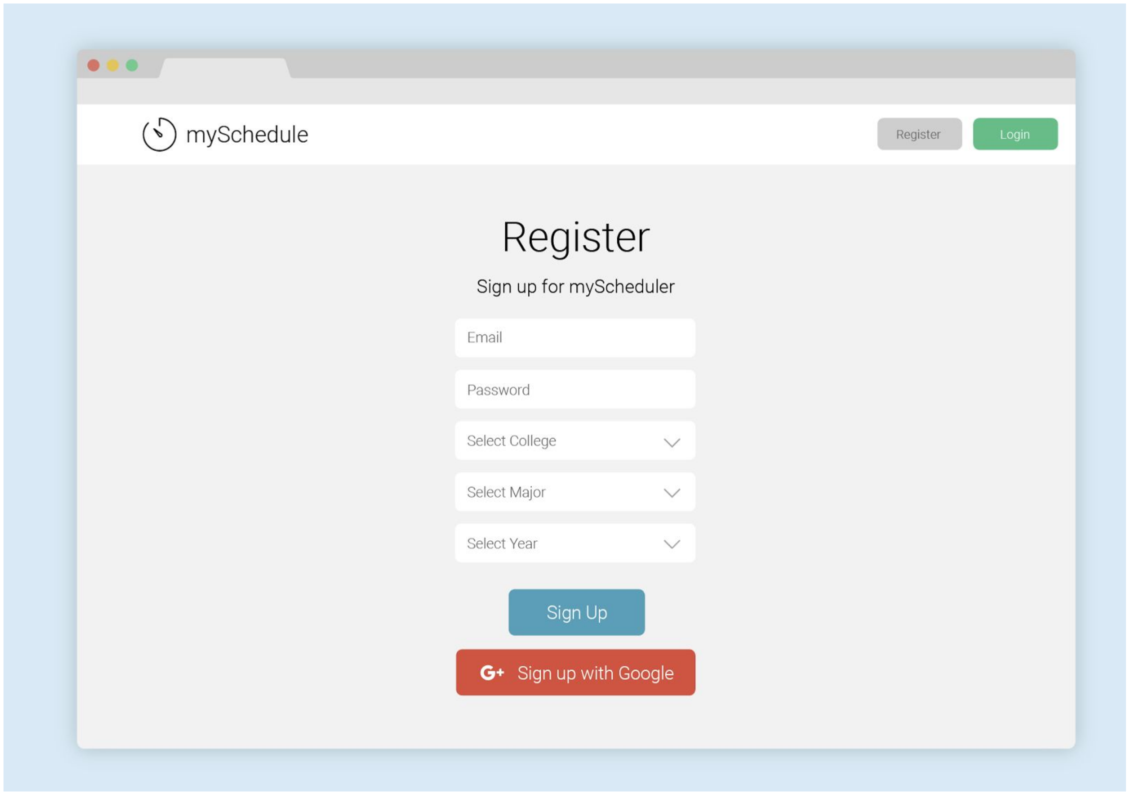
Class scheduling made easy.

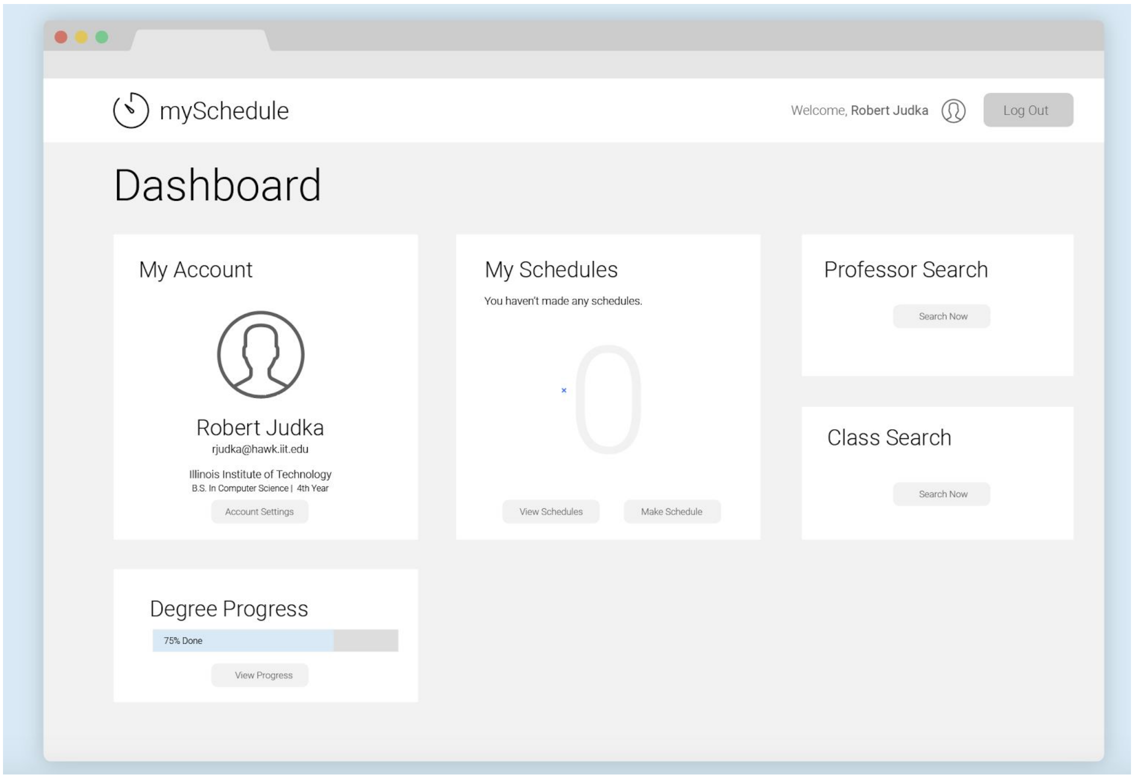


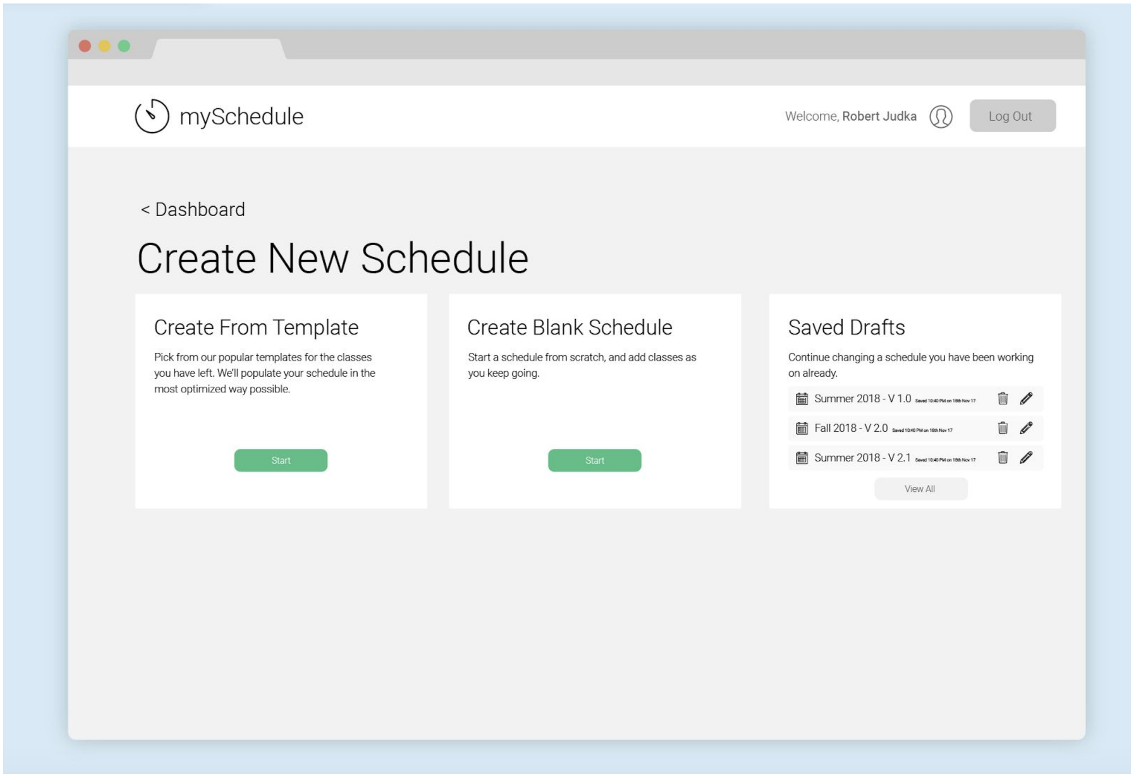
Login

Sign into myScheduler

Forgot Password?


mySchedule
Register
Login
Register
Sign up for myScheduler
Email
Password
Select College
Select Major
Select Year
Sign Up
Sign up with Google


mySchedule
Welcome, Robert Judka
Log 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

< Dashboard

Create New Schedule

Create From Template

Pick from our popular templates for the classes you have left. We'll populate your schedule in the most optimized way possible.

Start

Create Blank Schedule

Start a schedule from scratch, and add classes as you keep going.

Start

Saved Drafts

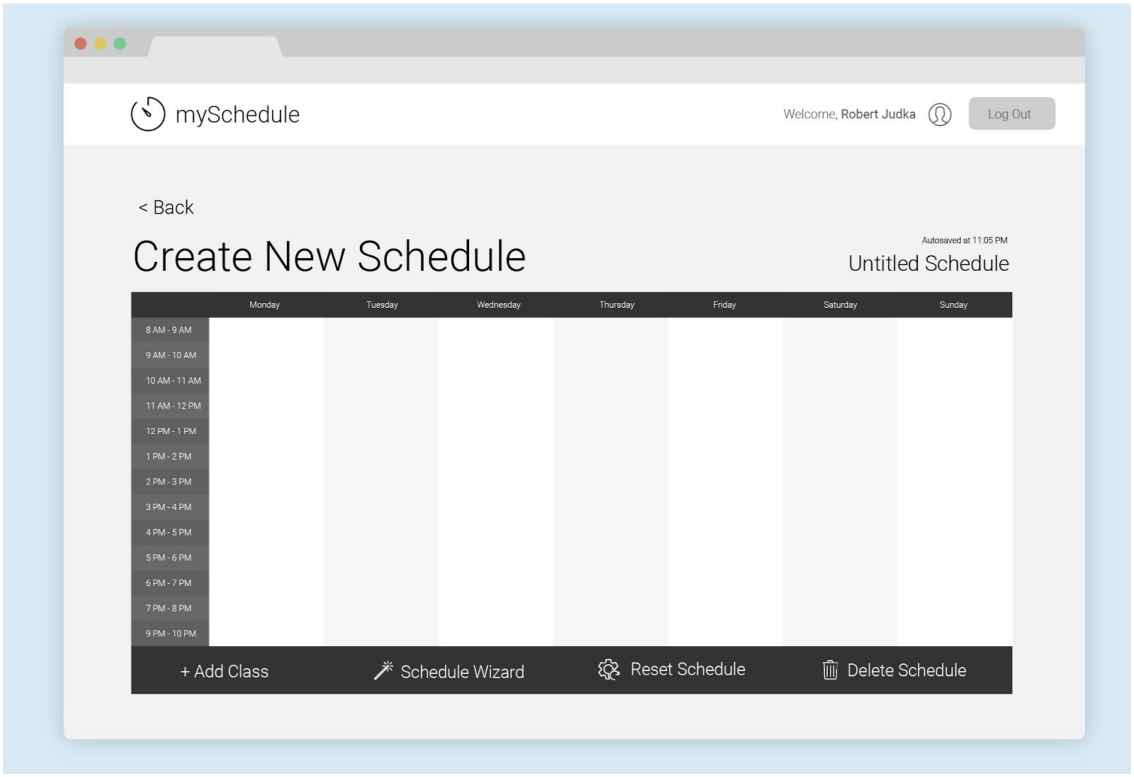
Continue changing a schedule you have been working on already.

Summer 2018 - V 1.0 Saved 10:40 PM on 18th Nov 17

Fall 2018 - V 2.0 Saved 10:40 PM on 18th Nov 17

Summer 2018 - V 2.1 Saved 10:40 PM on 18th Nov 17

View All



mySchedule

Welcome, **Robert Judka**

Log Out

< Back

Create New Schedule

Autosaved at 11:05 PM

Untitled Schedule

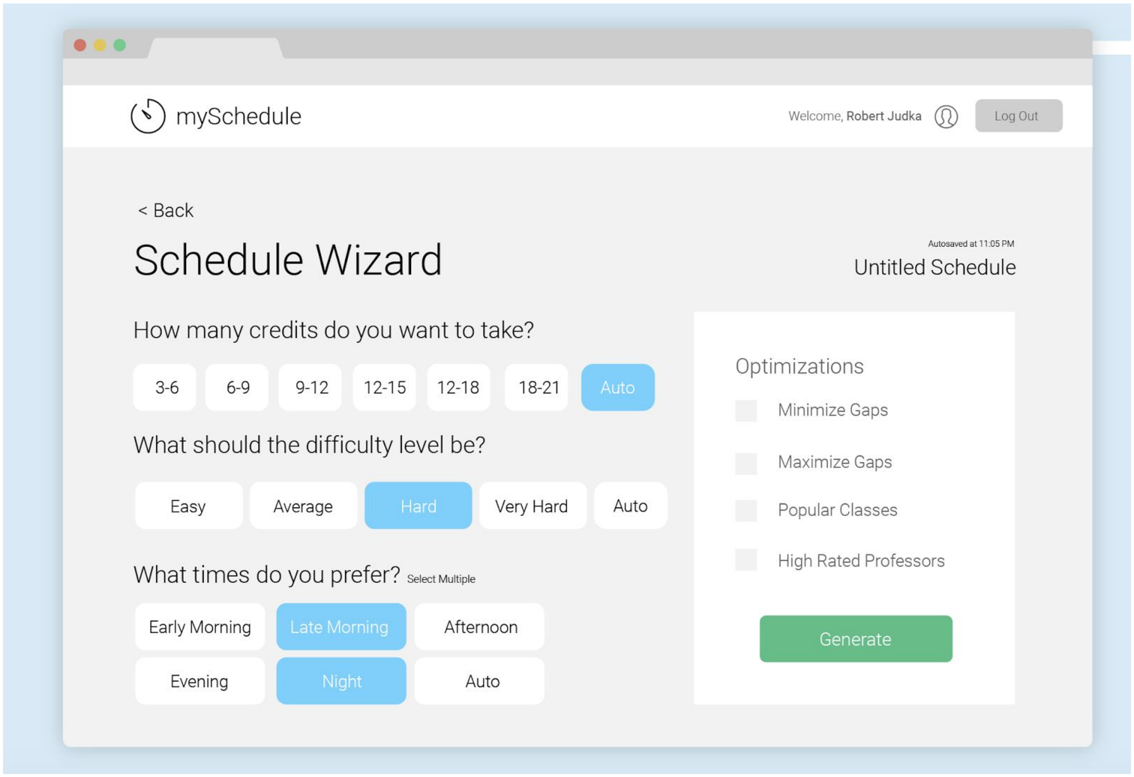
	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
8 AM - 9 AM							
9 AM - 10 AM							
10 AM - 11 AM							
11 AM - 12 PM							
12 PM - 1 PM							
1 PM - 2 PM							
2 PM - 3 PM							
3 PM - 4 PM							
4 PM - 5 PM							
5 PM - 6 PM							
6 PM - 7 PM							
7 PM - 8 PM							
9 PM - 10 PM							

+ Add Class

Schedule Wizard

Reset Schedule

Delete Schedule


mySchedule

Welcome, **Robert Judka** Log Out

< Back

Schedule Wizard

Autosaved at 11:05 PM

Untitled Schedule

How many credits do you want to take?

3-6 6-9 9-12 12-15 12-18 18-21 [✓ Auto]

What should the difficulty level be?

Easy Average [✓ Hard] Very Hard Auto

What times do you prefer? Select Multiple

Early Morning [✓ Late Morning] Afternoon

Evening [✓ Night] Auto

Optimizations

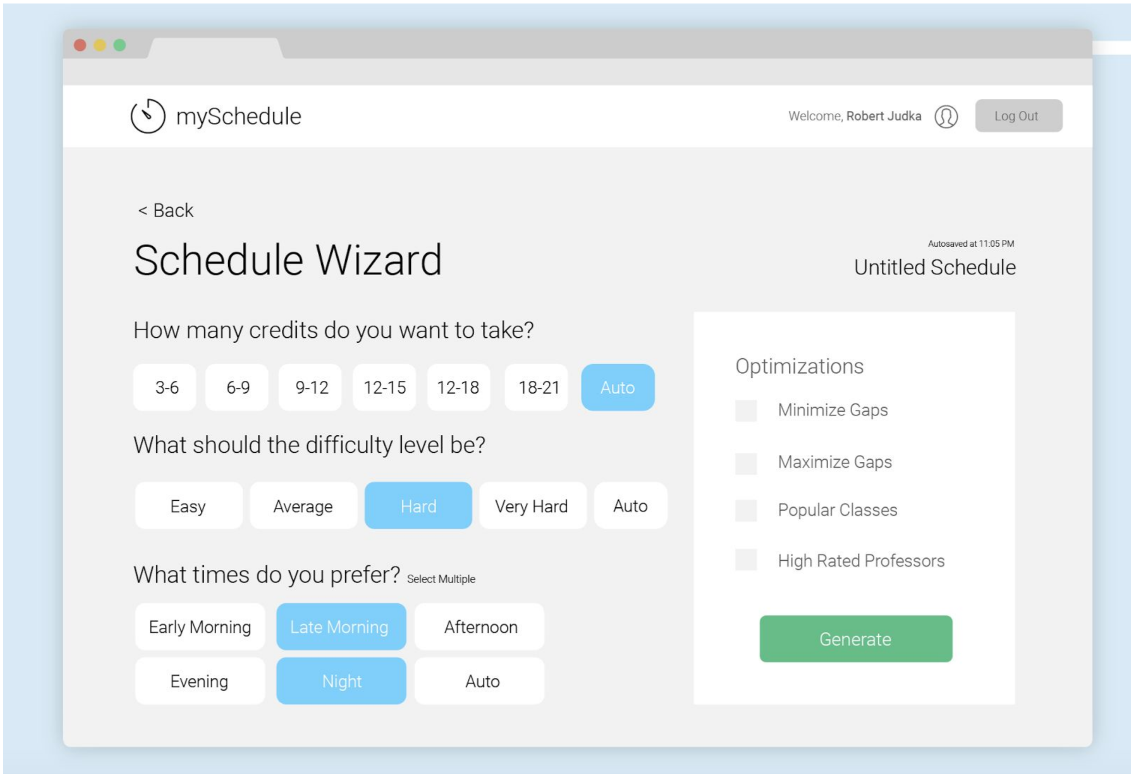
☐ Minimize Gaps

☐ Maximize Gaps

☐ Popular Classes

☐ High Rated Professors

Generate



mySchedule

Welcome, **Robert Judka** Log Out

< Back

Schedule Wizard

Autosaved at 11:05 PM

Untitled Schedule

How many credits do you want to take?

3-6 6-9 9-12 12-15 12-18 18-21 [✓ Auto]

What should the difficulty level be?

Easy Average [✓ Hard] Very Hard Auto

What times do you prefer? Select Multiple

Early Morning [✓ Late Morning] Afternoon

Evening [✓ Night] Auto

Optimizations

☐ Minimize Gaps

☐ Maximize Gaps

☐ Popular Classes

☐ High Rated Professors

Generate

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
```