RegNow

CS428/CS429 - Spring 2014

Samuel Gegg (gegg2), Rishi Girish (rgirish2), Kurtis Houdek (houdek2), Perry Huang (huang185), Nicholas Nordeen (nordeen1), Michal Szymulanski (mszymu2), Ethan Timm (timm3)

Part 1: Project Description

Regnow is a one stop tool for **class organization**, **recommendation** and **registration**.

Class organization and recommendation is an essential to any student going about choosing classes. Our application makes it easier for students to filter classes based on professor ratings, credit hours and average GPA for the course. Additionally, students can plan their schedule and view how their weekly schedule would look like. All this data is scraped from the university course database as well as third-party sites RateMyProfessors.com and Koofers and stored in our private database.

Automated class registration system that constantly checks class availability on the university registration system and it registers students who are interested in taking those courses. Students authenticate themselves using the university credentials and enter the classes that they want to register. This systems maintains a queue, for each class, of students who want to register for it. Upon availability, it pops people of the queue and automatically register them for the classes and sends them a confirmation message about it. Additionally, if users do not want to share their university credentials with us, Regnow also has a notification system that can inform the user of class availability in real time via email and sms.

Part 2: Development Process

We followed XP without stress on pair programming but following its other principles of continuous process, test-driven development, and collaborative code ownership. We followed XP's philosophy of continuous process, writing quickly and always going back to improve and refactor code as we went along. As we worked we made sure to refactor our code to make it more readable and maintainable so that we could more easily make changes as necessary. We also changed our requirements for each iteration wherever it seemed fit. There were portions of our project that were developed in an iteration that we did not initially expect. We also followed through with XP's guidelines regarding test-driven development. We began each part of our project by writing tests for the planned use cases, considering all the potential edge cases in which our code may fail, and only when that was complete did we write our implementation, keeping those potential issues in mind. However, we chose not to always follow XP's guidelines regarding pair programming. We noticed that it was sometimes logistically difficult to schedule meeting times for multiple people. It was more practical to allow team members to work mostly independently as long as they followed XP's guidelines regarding collective code ownership, made their work available and up-to-date on our repository, and reached out to other team members for assistance whenever more collaborative development was necessary.

Part 3: Requirements and Specifications

Requirements:

- To use the automated registration system, user must be a current student at the University of Illinois at Urbana-Champaign.
- Technology stack:
    - Python 2.7.4, Flask, BeautifulSoup, PyMongo
    - Ruby, Sinatra, Mechanize
    - AngularJS
    - Bootstrap
    - HTML/CSS/JavaScript
    - MongoDB
    - Twilio API

User Stories:

- **Iteration 1**
    - Write Proposal
    - Decide project framework and specifications.
    - Choose the technologies we are going to use.
    - Form Team
- **Iteration 2**
    - Implement Enterprise registration clone to test automated registration system
    - Set-up the web framework and database servers. Implemented a base for our test suites.
    - Read from database and design an algorithm to randomly chose a netid.
    - Build a Web crawler to gather data about classes from the University website.
    - Build a low-fidelity prototype of the front end to develop a better idea of functionality.
    - Design a database (MongoDB) to store data from RateMyProfessor
    - Build a Web crawler to gather data from RateMyProfessor.
- **Iteration 3**
    - Registration test system shows whether classes are closed of available.
    - Registration test system has custom classes configurable for different scenarios
    - Build RegNow course availability notification system
    - Build periodic course availability query system
    - Build registration automation system
    - Build queue management system for course registration
    - Design and implement user database, including tools to edit user data/profiles.
    - Design and implement user registration and profile pages.
    - Design and implement home, professor, and class pages.
    - Format data from RateMyProfessor and Koofers for database

- Build a web-crawler to gather data from Koofers
- Integrate data from my.illinois.edu with the database
- Collect course data from my.illinois.edu

- **Iteration 4**
  - Design and implement user registration and profile pages
  - Regnow management of user and class objects
  - Users can be added to dynamic queue for registration
  - Deployed registration test website
  - Redesign the web layout of the application and redesigned the front end. Basic form designs and content.
  - Set up the web server to host the web-page and set up the domain name and connect with the hosting server.
  - Write GUI manual testing manual.
  - Add RateMyProfessor and Koofers data to database
  - Build a part of the dashboard that will show weekly schedule
- **Iteration 5**
  - Add initial thread pooling support.
  - Use parameterized testing.
  - Implement backend routine logic.
  - Implemented the user registration functions and user credential validation functions.
  - Extended the gui testing manual for new cases.
  - Display course/professor information stored in database
  - Enable course registration (including test environment)
  - Allow user profile creation and login
  - Add option that will add class from a search result to the weekly schedule
  - Build a search that will include searching criteria such as: course type (English, Computer Science, etc.), credit hours, course time, course term, on site or online class.
- **Iteration 6**
  - Add classes to the calendar in the profile page, and the class table page.
  - Create way to filter classes in class table.
  - Create a quick way to search for a class
  - Add option that will add class from a search result to the weekly schedule
  - Write backend query functionality for date & time of classes
  - Set up the user profile page and add RegNow automated registration link
  - Registration system automatically registers students for classes from waiting lists when spots open in those classes
  - System supports classes that must register with multiple CRNs concurrently
  - REST API to register for classes or sign up for waiting lists

Use Cases:

Table 3.1 Actor-goal List

Actor	Task-level Goal	Priority
Student	Set system to automatically register for a class when space opens	1
Student	Set system to receive course availability by email	2
Registration System	Monitor Class Availability	1
Registration System	Register students for courses with wait lists	1
Student	Set system to automatically switch course section	2
Registration System	Automatically register students from wait list	1
Registration System	Automatically notify students on course availability	1
Student	Set system to stop receiving course availability by email	2
Student	Search for Classes	2
Student	User Sign Up	3
Student	Edit User's Registration Request	4
Database	Crawl the Web for Data	1

Table 3.2 User case briefs 1

Actor	Goal	Brief
Student	Select Classes for Registration	Enter criteria for desired classes, which are searched for in the data base. Then select classes. Log in to the system or sign up and send classes along with login information to Registration System.
Student	Search for Classes	Search for classes in the database according to the criteria provided by the Student.
Student	User Sign Up	Collect and insert into the database Students login information.
Student	Edit User's Registration Request	Make changes to the Student's request for registration and send new registration request to the Registration System.

Registration System	Register Classes for Student	Check which Student's registration request is highest in the queue and periodically attempt to register request highest in the queue.
Database	Crawl the Web for Data	Crawl UIUC and RateMyProfessor websites in order to populate the database with the most recent class and instructor information.

Casual Use Case

Select Classes for Registration

Student will enter search criteria that will help her find desired classes in the database. The Database will check the search criteria against the database and return results to the Student. Then the student will make choices on classes and add them to his schedule. Once the Student is done with selecting classes she will log in to the system or sign up in order to provide UIUC NetId and Enterprise Password. Then the registration request is send to the Registration System to add the request to a registration queue and eventually be registered.

Full Dressed Use Case

Select Classes for Registration

Primary Actor: Student
Goal In Context: Select classes needed by the student and request registration of the student for the classes.
Scope: Organization – option available to all students across the campus for class search and registration.
Level: User level.
Pre-condition: none.
Successful Guarantee: a student will be added to the queue for registration and eventually registered for classes.
Minimal Guarantee: a student will find that his requirement collide with each other and the registration request cannot be submitted.
Trigger: a student wanting to register for a class, which viability fluctuates with other students dropping from and adding to their schedule.

Main Success Scenario

1. Student enters search criteria and starts the search.
2. Student begins to select classes from the search result and add them to her schedule.
3. Student logs into the system.
4. Student sends the registration request to the Registration System.

Extensions:

1a:

1. The search returns empty result from the database.
2. Student must refine search criteria and search again.
3. Go to step 2.

2a:

1. Student cannot find classes that fit her schedule or don't overlap.
2. Go back to step 1.

3a:

1. Student is not registered in the system and must register.
2. After registration student logs into the system.

Use Case Briefs:

Table 3.3 User case briefs 2

Actor	Goal	Brief
Student	1. Set system to automatically register for a class when space opens	Student will provide his login credentials to our system. He will then select which class he wants to be registered for when space opens, and the section and time.
Student	2. Set system to receive course availability by email	Student will be given the option to receive email notifications when student specified courses become available.

Fully Dressed Use Cases:

Use case 1:

Primary Actor: Student

Goal in Context: Have system automatically register courses for students when spots are available

Scope: Front end user interactions

Level: user goal

Stakeholders and their interests:

Student/user - uses tool to register for class
RegNow - must handle user credentials securely, help students, and gain popularity with students
University - has history of being against automatic registration
Project owners - this functionality is a major feature the project centers around

Pre-condition: registration system must be able to assess whether class is open, and then must be able to automatically register students

Successful Guarantee: Users can join a waitlist and when the class opens they will be registered for it by the system automatically

Minimal Guarantee: When a student selects a class to register for, if the class opens, the system will automatically make an attempt to register for the class

Trigger: A user decides he wishes to join a class and that class is full

Main Success Scenario: A user registers on the RegNow waitlist for a class. There are no other students on the waitlist, but the class is full. A spot opens in the class. RegNow automatically registers the student for the class.

Alternative Scenarios: Many students register on the RegNow waitlist. No spots open in the class. None of the RegNow users are able to register for the class that semester.

Use case 2:
Primary Actor: Student
Goal in Context: Have student be notified immediately by email when system detects that courses become available
Scope: Front end user interactions
Level: user goal
Stakeholders and their interests:

- Student/user - uses email notifications to help register for class
- RegNow - wants to help students and gain popularity with students
- University - has history of being against automatic registration
- Project owners - this functionality is a major feature the project centers around

Pre-condition: A specific course needs to be open for register and the registration system needs to detect that it is
Successful Guarantee: An email will be sent to a student immediately when the system detects that the desired course is available to register.
Minimal Guarantee: Same as successful guarantee.
Trigger: A student adds a course to be monitored by RegNow system and opts for email notifications.
Main Success Scenario: A student adds a course to be monitored by the RegNow system and opts for email notification. The student starts starts receiving email notifications whenever the system detects that the course is open. The student is able to register for the course shortly after receiving and reading one of the email notifications.
Alternative Scenarios: A student adds a course to be monitored by the RegNow system and opts for email notification. The student starts starts receiving email notifications whenever the system detects that the course is open. The student receives and reads the emails, but never is able to register for the course successfully. The specific section closes by the time the student logs online and attempts to register for the course.

Part 4: Architecture and Design

Top - Down layout of our Application

Initial interaction of any user to our application is through our webpage. Users visit https://www.illiniregnow.com. From here, the users have to go to “Course Information” or use the search bar to find the courses they are interested in. Choosing the courses they want to register for, they go under “Register Now” and input their netid and AD password. Using theses credentials, our system automatically registers the user for that class if it’s available or, adds the user to the queue of that class to be registered later when a spot opens up.

Our applications top layer is the web interface. This is all that interacts with the user. Underneath it are top very separate applications, namely, automated registration system and the course recommendation and planning system. These independent applications do not interact with each other but share the common database. The lowest level of our system is the database(MongoDB) that stores all the required information used by our application.

Frontend

For the front end we used AngularJS to create the website in a MVC design. The directory static/partials/ contains all the views for the website. The directory static/js/controllers contains the models for the views. The directory static/js/directories, and static/js/services contains the controllers. Some angular controllers contain variables linked to forms in that view, and functions that are called when certain buttons are clicked, for instance in the class table view if the "Filter Classes" button is clicked it calls a function `submitFilter()` that will then call a function `postFilter(obj)` which sends the http post request to the python flask server. The angular controllers (models) don't communicate with the server directly, the directory or services are the ones that do that.

AngularJS doesn't reload any of the pages when navigating through the website. Index.html and main.html are the only pages that will be accessed; when navigating through the website, the URL will change but the browser doesn't reload. App.js sees when the URL changes and then it loads the new view html from the partials directory and will also load new javascript code, the model (called the controller in angularjs), depending on which is specified. When the URL changes, the new model and view are pasted into the `<div ng-view></div>` tag in main.html, this helps with redundancy. Since index.html has the navbar, then the new views that are loaded won't need to have the navbar html in it, because the site always stays in index.html, and the views are pasted into the `<div ng-view></div>` tag.

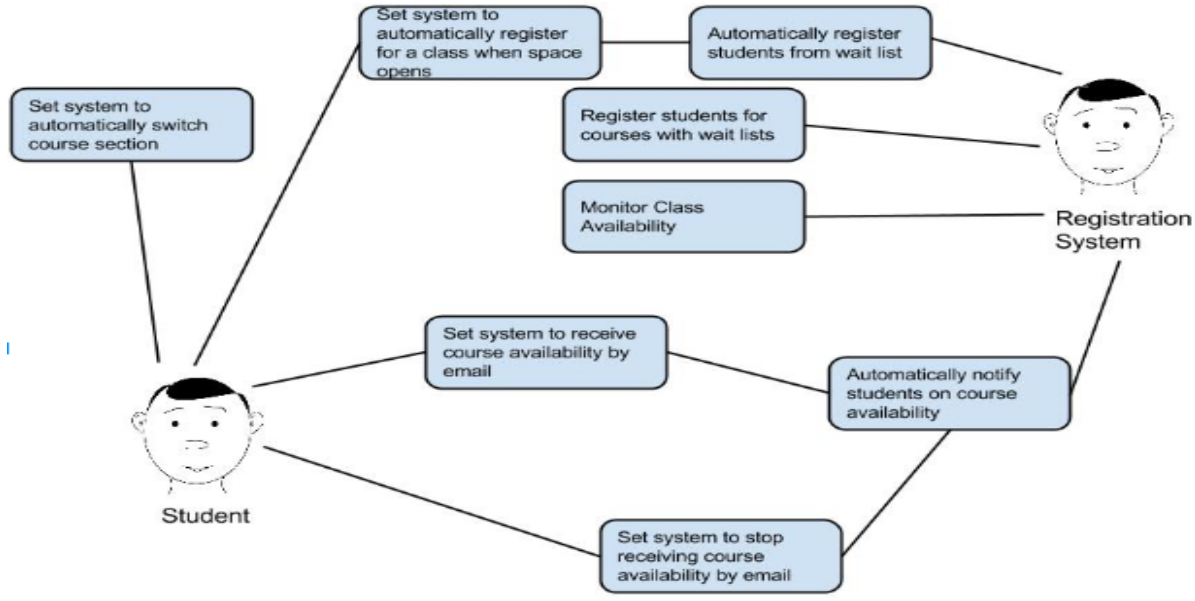

Figure 4.1 Use case diagram

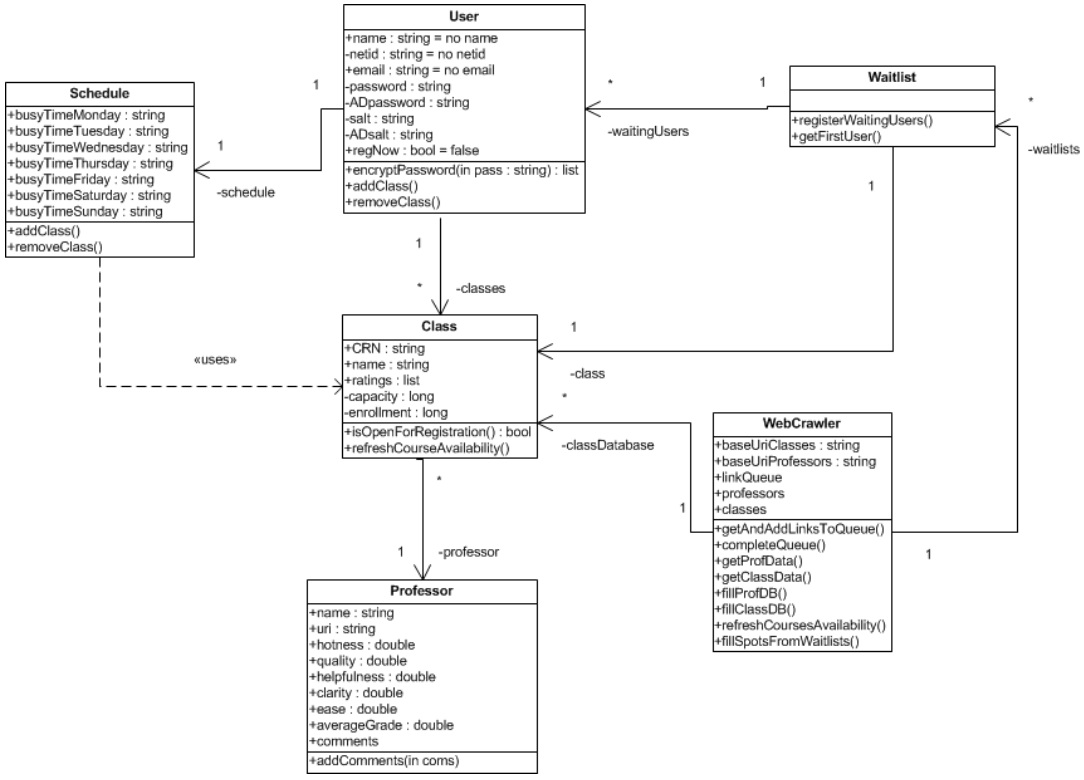

Figure 4.2 UML class diagram for registration system

Web crawler to gather data about classes from the University website

`classTable/src/scraper/MyIllinoisXMLRequest.py`

`classTable/src/scraper/TestXMLReader.py`

`classTable/src/scraper/XMLReader.py`

`classTable/src/connect/Monnect.py`

A library to access CISDOCS and collect class data available for a given school year and semester. The crawler requests XML documents from CISDOCS and parses the XML documents into Python objects. Then the data from the course objects are inserted into the MongoDB database. Each course CRN is unique in the database. The script for updating courses runs periodically to keep available courses in sync with the university courses. The library is implemented using Python's built-in library ElementTree.

Course Calendar

`frontend/static/js/courses_calendar.js`

`frontend/static/css/courses_calendar.css`

The course calendar displays courses selected by user in a weekly calendar. The calendar library takes care of placing selected course in the calendar at the correct time with appropriate time length of the course.

The calendar is entirely implemented using HTML, CSS and JavaScript. It runs entirely in browser.

Web crawler to gather data about classes from the University website

`classTable/src/scraper/MyIllinoisXMLRequest.py`

`classTable/src/scraper/TestXMLReader.py`

`classTable/src/scraper/XMLReader.py`

`classTable/src/connect/Monnect.py`

A library to access CISDOCS and collect class data available for a given school year and semester. The crawler requests XML documents from CISDOCS and parses the XML documents into Python objects. Then the data from the course objects are inserted into the MongoDB database. Each course CRN is unique in the database. The script for updating courses runs periodically to keep available courses in sync with the university courses. The library is implemented using Python's built-in library ElementTree.

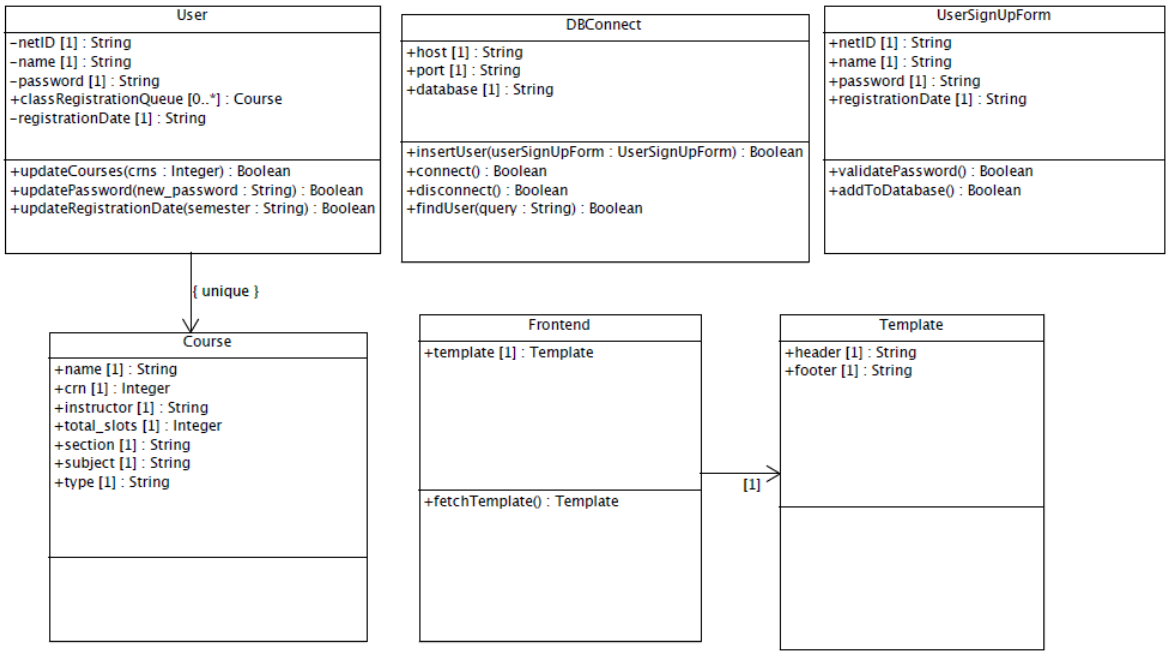

Figure 4.3 UML class diagram for course registration

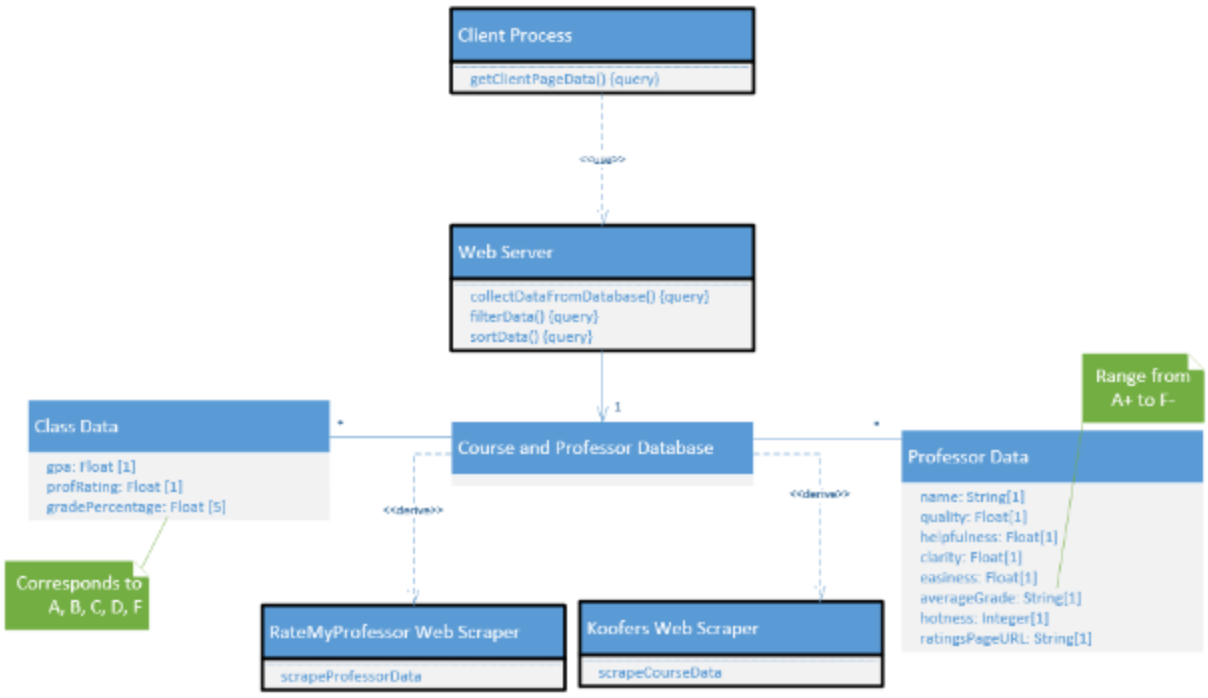

Figure 4.4 UML Class Diagram for getting course information

Registration System

`regnow/helpers/notification_manager.rb`

The notification system provides users with a text message or email-based notification when a desired course is available to register. The text messages are sent from using Twilio.com's API and the emails are sent from our server's SMTP server.

`regnow/helpers/queue_manager.rb`

This is the queue management system that treats Array objects in our MongoDB server as queues. Our registration system will use this queue system as a waitlist for each course.

`regnow/helpers/registration.rb`

This is the main registration logic behind Regnow that processes our queue system and triggers automated registration actions and notifications. Our queue is routinely processed and shares the same database with ClassTable for course information.

`regnow/mock/main.rb`

This is a complete mock of the University's registration system that we used in our development process to verify our work without interacting with the official University system.

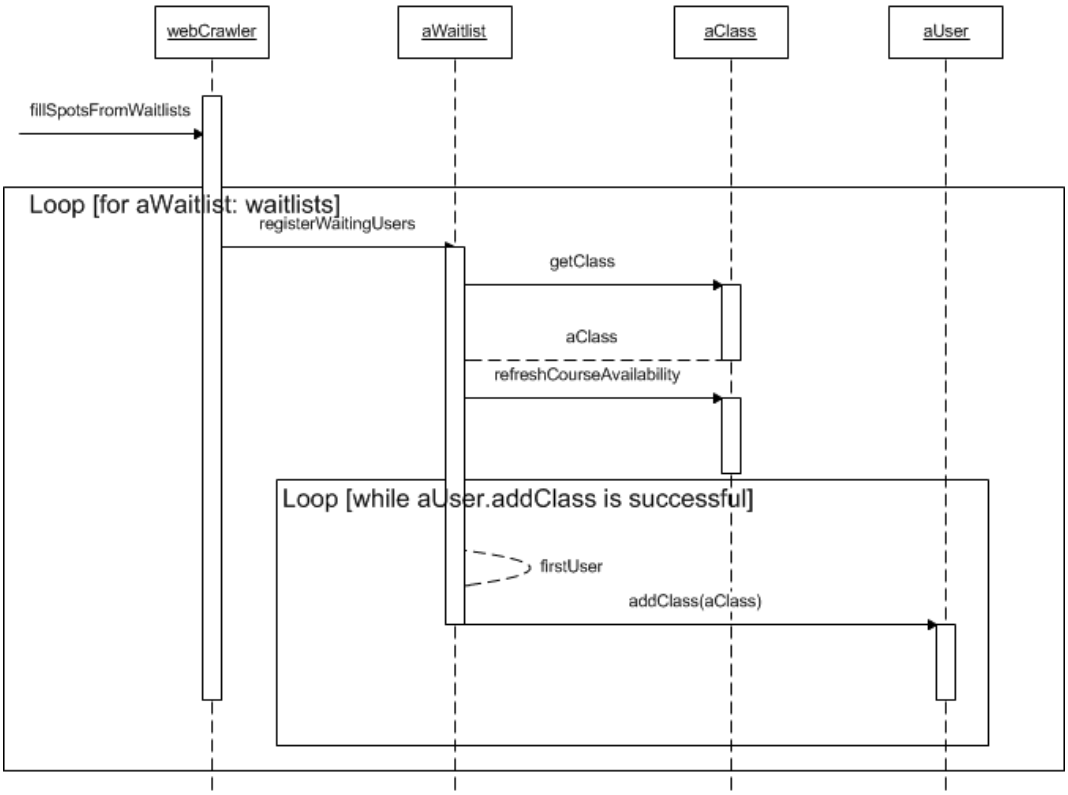

Figure 4.5 Sequence diagram for getting courses into database

Part 5: Future Plans

Next steps in the RegNow project are adding additional security to our password storing mechanisms, thorough testing with the real Illinois registration system, and then marketing and release to the public.

Critical parts of our project require us to store users' netID passwords in a form that can be converted to plain text. This is a requirement, as our entire registration service relies on the ability to log in to the Illinois Enterprise registration system as each user to add classes for them. Storing passwords in plain text or a format that can be unencrypted to plain text is a major security risk and must be treated carefully. Very few websites do this, and it seems the best way to approach it would be to a separate server that is locked down that stores the passwords and performs registration operations. This would make it easier to control the access an attacker could achieve, especially if there was no way for this machine to return any output, such at the users' passwords.

Next we would need to test thoroughly with the actual University of Illinois registration system. Our testing has been with a clone of the system that behaves identically on the front end, but we have not done extensive testing against the real system. We need to test against the real system to ensure its servers can handle the increased load we will be placing upon it, as well as check to make sure the system does not notice our system and block us, perhaps classifying us as malicious or breaking rules.

To release to the public we would need to make users aware of our service and what it does, and then convince them that their data is safe with us. We could use online UIUC related forums to publicize our website, and have a description of how it works on our front page. The hardest part would be earning the trust of UIUC students, since they will have to disclose their netID passwords to us. To do this we will probably just have to wait for a few early adopters to try our software, then rely on them spreading the work that our software is trustworthy and useful.

Personal Reflections:

Rishi:

RegNow was a unique experience that allowed me to work closely on a significant project as part of a big team. I started on this project working on the automated registration system backend. Most of contribution was testing out the ResQue and Redis queue management system and integrate it with our project. Unfortunately, we decide to not implement the queueing using those two applications. Moving on, I significantly worked on the frontend design and layout of the project. Integrating it with BootStrap and unifying both ends of the project under the single UI was major part of my work in iteration 5 and 6. I strongly think RegNow could be a vital tool for students when they are looking to choose and register for classes. As the university is expanding, more now than ever, classes are becoming harder to register for. This tool can be a huge help to students moving forward.

Kurtis:

While the Registration System functions perfectly as currently planned, there is very little feedback when registration fails about why the attempt failed. It could be because the user is already registered for the class, because they are restricted from registering, or they are not in that major. This feedback could be helpful for a user who is confused why his registration request failed.

Perry:

I worked on Regnow and hope that our work can someday be used publicly with the University's registration system. Some departments, including the Computer Science department, are growing at a tremendous rate and students are often finding difficulty while registering for courses. There is also no waitlist system that is available for every class, so users with custom automated registration systems will dominate those without. I think that many students will find our Regnow system very useful. The University has a history of standing against the use of automated registration systems and this forced us to create our own mock registration system to test our features.

Sam:

I greatly enjoyed working on this project. Getting to apply my knowledge and skills in support of a larger product was a great experience I hadn't gotten before. I found more than anything else I learned a lot about communication and working with a team. At the beginning our team had its issues with communication: it took some time before everyone was on the same page with regards to what needed doing and how to accomplish it. As time went on our collaboration skills improved and we were able to work efficiently as a cohesive team. I certainly hope our product will see use: I think it has a lot of potential and with some further work and polish could make a great supplement to or even replace existing systems

within the university. If we decide to continue building we need to use the summer registration period as an opportunity to test on the live system and work further on our user interface, as while the functionality is mostly there it can be a bit hard to utilize for a new user.

Michal:

Just like in every CS class with a term project I learned a little bit more about organization of a small team of developers. As the leader of this group I did not learn anything new because I don't have any leverage to motivate people except spamming everyone with text messages and emails about work due. In general the project came out quite well, most of its features work but they must be polished. The setup of the whole project is over complicated due to use of Ruby and Python. I still don't see any good reason for using two different scripting languages. I see a good future for this project, it can successfully replace the school registration system.

Ethan:

Working on RegNow with the team was good experience to get under my belt. Early on, we collectively found the largest difficulty to be communication. A personal difficulty was attending the group meetings; circumstances with my infant daughter and the hospitalization of my fiance had my schedule and personal life all over. Even with these difficulties, being a part of RegNow's creation has been a rewarding pleasure; the project is a natural, but incompatible, complement to my past cs411 project, IlliniAdvisor, which designed personalized class schedules among other features designed to aid students in signing up for classes effectively and efficiently. I look forward to taking the valuable time management, communicative, and product design concepts learned from this project with me to my future endeavors.

Nick:

This course was a good experience for me. It taught me how important communication was for a project, and that meeting more often is important. In the beginning we did not communicate or meet up as often as we should have, and I slacked off because I didn't put the effort needed to stay in the loop. It took me a bit longer than it should have to get back on the right road, and I've learned that I need to be more proactive when working in a group like this. If we had communicated more, and had been more willing to try new things, there wouldn't have been as many conflicting things, for instance rewriting of code that people have already done, and there wouldn't have been two servers. Communicating more would have made combining functionality easier.