Technical Report

SISCalendar

Prepared by:
Michael Caputo
Zach Masiello
Ethan Mick
Shawn Thompson

Organization:
SIS.io

Sponsored by:
ITS Enterprise Web Applications team
Kim Sowers, Rick Myles, Lisa Hupf, Xiuli Gong

Faculty Coach:
Sam Malachowsky

Project Overview

Information & Technology Services (ITS) updated the Student Information Service (SIS) this past year, which is the system that allows students to access their class schedule and choose their classes. The new system offers many improvement over the old system, but it was still missing some crucial functionality. Student Government sent out a survey asking students what functionality they would like to see added. The biggest requests were; the ability to export the class schedule to Google Calendar/iCal, and the ability to view class locations on the RIT map.

Team SIS.io added this desired functionality to the SIS system through a satellite service, named SISCalendar. This service was implemented as a responsive web application, allowing devices of all screen size to easily access the functionality. The service uses the campus' central authentication system to ensure users only access their own information and utilizes ITS' SIS Gateway in order to provide the data.

This service can be used by students on all of the RIT campuses worldwide. For this project, we considered a student to be any person taking classes at RIT. If the student is taking classes on RIT's Rochester campus the service will allow the student to view their classes through the RIT campus map; however this will not be enabled if the student does not have classes on that campus.

SIS.io also expanded upon the original requirements and added additional functionality to SISCalendar. The primary features are the ability for a student to download their exams to an iCal file, and to import both their classes and exams into their RIT Google account calendar. Additionally we also account for different campuses' holiday calendars, and time zone differences between the campuses.

Basic Requirements

The original functional requirements for the project were as follows:

- Authenticate the user using the campus' central authentication system, Shibboleth
- Get the user's current schedule from the ITS developed SIS Gateway
- Display the schedule to the user with the location linked on the RIT Maps
- Download the calendar to an iCal file to allow importing into a calendaring application
- The website must be responsive and work on mobile

After completing these requirements in the first semester, we worked with the sponsor to determine the requirements for R2, which were as follows:

- Use the SIS Gateway to get the Holiday Calendars and account for holiday's with class schedules
- Account for withdrawn classes and don't display/download them.
- Include Exam information with the class information. This would appear in users calendar applications after the import.
- Import the students schedule to their RIT Google Calendar. This would integrate with the Google Calendar API and not involve any file downloading.

Constraints

Our team was limited in the technologies we could use because ITS is supporting our application upon completion. This meant that all technologies chosen took into account what ITS would like to support. Whenever we had a good case for choosing a different technology, that they may not have supported before, we brought it to them and explained why it was a good choice.

Our meeting times were also limited with our sponsor, and our own team meetings, because we had other classes, projects, and meetings we had to be at and ITS has their own meetings to work around. Luckily ITS was able to meet during the scheduled time for Senior project. So we were all able to meet every week at the same day and time throughout both semesters.
During the implementation phase we were limited in our user testing during the first semester. ITS had our development server behind a firewall so we couldn't have other users use our application on their own machines. But during the second semester we got the firewall opened so that it would be easier for others to access the application.

Development Process

During the first semester, we used the evolutionary delivery methodology. This process was proposed and explained to the sponsor and they agreed with the iterative approach that allowed for quick feedback. Evolutionary delivery is a methodology that blends evolutionary prototyping and staged delivery. In evolutionary prototyping the most prominent parts of the program are built first and then shown to the customer. The customer provides feedback, and then the next prototype is worked on. This loop continues until the prototype is essentially "good enough," and is released as the final software. Staged delivery is similar to evolutionary prototyping, but doesn't rely on customer feedback nearly as much. Rather, the approach incrementally builds the software and delivers it to the customer until finished.

Evolutionary delivery blends the two together, allowing for a lot of flexibility in how much feedback is incorporated into the next version. This was fantastic for SIS.io, because the requirements for version 1 were open ended and ambiguous. Without knowing exactly how big the scope was, evolutionary delivery helped us focus down the scope and learn the technology at the same time. Version 2 and 3 were in the second semester and had their own challenges associated with them, which we dealt with later (see below). Having feedback quickly is critical in delivering a high quality product which satisfies the sponsor's requirements. By using the evolutionary delivery methodology for building SISCalendar, we were able to incorporate feedback as necessary.

We used JIRA to track our tasks, and used Toggl to track time worked on tasks. There are numerous documents that were maintained, but for the scope of version 1 we kept these documents small. These documents included requirements, design, and test documents. These documents were also updated throughout the project and delivered at the end.

The process did not outline how communication with the sponsor should occur, and so we would communicate progress during our weekly meetings or over email.

Roles

Roles were created and assigned at the beginning of the first semester. They were adhered to for the entire year.

- Project Manager - Ethan Mick

The project manager is in charge of ensuring smooth communication with the client team and productivity within the team. If any issue arise, the PM needs to deal with them efficiently and effectively. The PM is also in charge of managing risks, tasks, and timing.

- Requirements Manager - Shawn Thompson

The requirements manager is the master of all things relating to requirements. As the project progresses, changes may need to be made, or requirements will need to be managed. The requirements manager handles all of these requests and ensures the project can stay in scope. The requirements manager will also scribe the meetings.

- Architect / Scrum Master - Mike Caputo

The architect is the master of design and technical overview. He understands the technology stack, knows the positives and negatives about the stack and architecture, and fully understands how the system works and interacts with other systems.

The scrum master is responsible for ensuring that a scrum team lives and works by the values of scrum. This includes making sure that scrum meetings (daily scrums, sprint planning, sprint reviews, planning poker, etc) meetings remain on topic. In addition, the scrum master owns the scrum process and helps the team perform at the highest level possible.

- Test Lead - Zach Masiello

The test lead manages all aspects of testing. He understands how the tests are put together, goes through test coverage, ensures complicated parts of the system are well tested, and can put together reports of testing. He works closely with the architect to ensure the system is testable, and also works closely with the requirements manager to ensure the acceptance tests are met.

Scrum

For the second semester we approached our sponsor and told them we were switching our methodology to Scrum. Scrum is an iterative and incremental agile framework that utilizes a flexible development strategy to ensure functionality that the customers cares about is delivered on time and to the customer's satisfaction. The lifecycle of Scrum is broken into chunks called "Sprints". SIS.io utilized one-week sprints to ensure that work was not put off until the last minute and thus wasting one or more weeks of development time.

Our implementation of Scrum began each week on Monday. The SIS.io team met each Monday for a weekly reflection and sprint planning meeting. At this meeting, the team reviewed what went well during the last sprint, what problems were encountered in the last sprint, and what could be done better in the upcoming sprint. After this reflection, the team discussed any necessary changes to the methodology that are required based on the reflection feedback.

Once any new changes to the methodology were settled, the team began with the sprint planning portion of the meeting. This part of the weekly meeting involved looking at the backlog list in Trello and determining which high priority tasks would be pulled into the sprint that week. Tasks were pulled into a sprint based on their priority as determined from the following criteria:

1. The priority of the task to the customer
2. The amount of time the task is expected to take
3. The amount of work that has already been put into the task

After discussing the above criteria, if there were any ambiguities of what the task involved then the task was postponed until clarification from the customer was obtained. The number of tasks pulled into a sprint varied week to week based on the velocity (the number of story points the team completed in the last sprint). The amount of story points in a sprint should did not exceed the amount of story points completed in the last sprint. This was sometimes tweaked if an exceptional circumstance happened during the last sprint.

Each Tuesday, the SIS.io team had a customer meeting. During this meeting, we provided a demo of any new functionality to SISCalendar that was completed during the previous sprint. SIS.io also updated the customer on everything that had been completed in the previous sprint, what things were not completed in the previous sprint, any ambiguities in the backlog, and what was planned for the upcoming sprint. Customer feedback throughout this meeting was noted in the meeting notes posted to the SIS.io website and were added to the SIS.io team's backlog as feedback to discuss at the next sprint planning meeting.

Another important aspect of the scrum methodology is the daily standup. The daily standup is a meeting between all developers in SIS.io each day to discuss what each person accomplished that day, what that person expects to accomplish tomorrow, and any blockers that the person encountered during the course of their work. Due to the schedules of the SIS.io team, daily standups were held every day (with the exception of Monday and Tuesday - the sprint planning and customer meetings were counted as daily stand ups for each of those days respectively) at 9pm. The standups were held using Fastchat each night.

We choose to switch to scrum because it reflected our new needs better. After completing one semester of work on SISCalendar we had a strong idea of what the requirements were. Also, we had finished all the core requirements our sponsor had asked. This meant for the second semester we could focus on adding additional value. However, the end goal was to ship the product to production, which meant at any point we had to have a potentially shippable product. Scrum was a great framework which gave us all these benefits. We all had experience with using scrum, so the transition was very easy for our team.

Communication was further enhanced because of our use of Trello. We had a public Trello board which all members of SIS.io and our sponsors had access too. This meant that tasks could be assigned to either side of the team and progress could be noted without having a lot of email threads. A second Trello board was also created for the development team, and faculty coach, which allowed us to have a private and fast way

to communicate. The switch to scrum was a great change in our methodology and allowed us to have good communication with our sponsor and track progress easily.

Project Schedule: Planned and Actual

SIS.io came up with a rough schedule in the beginning of the semester. It had semester 1 planned out in some detail, but semester 2 was left purposefully vague. We didn't feel confident at the beginning of semester 1 to plan out the entire year. We told our sponsor that when we returned from winter break we would be able to plan out the second semester. Our goal was to outline the main requirements in semester 1, and build a prototype in the first part of the semester. Using the prototype we would solidify the core requirements of the product, and then finish that by the end of the semester. We planned to be able to push the code to production at the end of of the semester.

The following is our schedule for semester 1:

09-24-13
- Project Plan (Draft)

10-01-13
- Use case document (Draft)
- Set up Dev environment with GitHub

10-08-13
- Query about GitHub ITS account
- Incorporate feedback for Use Cases
- Project Plan final Review with ITS
- SRS (Draft)
- Technical approach review
- Schedule following week's meeting

10-17-13
- Incorporate feedback for SRS
- Use Case Document (Final)
- Project Plan (Final)
- Design Document & Wireframes (Draft)
- Feedback on designs

10-22-13
- SRS (Final)
- Incorporate feedback for Design Document & Wireframes
- Begin Development Work (authentication, non-API work)

10-29-13
- Design Document & Wireframes (Final)
- Continue Development (authentication, non-API work)
- Begin work on Test Plan

11-5-13
- Test Plan (Draft)
- Development Meeting w/Lisa
- API access for eServices information and RIT Maps Application

11-12-13

- Review if we need access for test server.
- Customer demo

11-15-13
- Request Test Environment

11-19-13
- Test Plan (Final)

11-26-13
- No meeting, Thanksgiving week

12-3-13
- Setup test server
- R1 Deployed to test
- Request Production Server

12-10-13
- 4:00pm - Interim presentation
- Final meeting of the Semester

12-13-13
- R1 testing completed
- R1 code frozen and ready for production

Most, but not all, of these milestones were met. At the end of the semester, our code was in good shape, but not ready to be pushed to production. After talking to our sponsor, getting the production server ready in such a small time would probably not work. Instead of pushing to production, we readied our code to be pushed to the test environment.

Also, our test plan ended up being incomplete at this point because of the difficulties of testing in this environment. We had to request test accounts to sign up for classes, and with the various refreshes and environment it was hard to create a solid plan for testing that could be followed. Most of the testing was done in an ad hoc manner, testing the 80%.

For the second semester, we originally had a version 2 and 3 planned. However, since we changed our methodology to scrum it made more sense to just have versions be iterations. Finished tasks were pushed to our dev and test environments, and the app was always in a potentially shippable state.

The schedule for semester 2:

1-28-14 (Sprint 0)
- First Meeting of the 2nd Semester
- Plan out scrum

2-4-14 (Sprint 0)
- Meeting with Lisa
- Explore possible automation
- Ask about Database

- Setup Dev/Test Environment

2-10-14 (Sprint 1)
- Dynamically Add End Dates to Classes
- Include Professor Contact information

2-17-14 (Sprint 2)
- Ethan is not going to be here
- Cache Holiday Calendar Data
- Pick the Term

2-24-14 (Sprint 3)
- Help Desk Document
- Support Document
- Cache Class Location
- Instructions for iCal Handling
- Hide Withdrawn Classes
- Add Schedule directly to Google Calendar

3-3-14 (Sprint 4)
- Add Schedule directly to Google Calendar
- Instructions for iCal Handling →
- As an unauthorized, but authenticated, user, when I log in I should see an information page.

3-10-14 (Sprint 5)
- As a user, I should see holiday schedule information on my calendar.
- Add Schedule directly to Google Calendar
- As an Administrator, I would like to see Google Analytics for the application.
- As a user, I want to see my final exams.

3-17-14 (Sprint 6)
- As a user, I should see holiday schedule information on my calendar.
- Add Schedule directly to Google Calendar
- As an Administrator, I would like to see Google Analytics for the application.
- As a user, I want to see my final exams.
- Bug Fixes

3-23-14 (Sprint 7)

- R2 Complete
- As a user, I should see holiday schedule information on my calendar.
- Add Schedule directly to Google Calendar
- As an Administrator, I would like to see Google Analytics for the application.
- As a user, I want to see my final exams.
- Bug Fixes

3-31-14
- Spring Break

4-7-14 (Sprint 8)
- Bug Fixes
- Code Walkthroughs (Prior to, technical documentation given 1 week)
- Update code from code review.
- Senior Project Poster

4-14-14 (Sprint 9)
- Bug Fixes
- Code Walkthroughs
- Testing and User Testing
- Senior Project Technical Report

4-21-14 (Sprint 10)
- Bug Fixes
- Code Walkthroughs
- Testing and User Testing
- Senior Project Technical Report
- 4/24 - Meeting to review data and application.
    - 1-2, building 1.

4-28-14 (Sprint 11)
- Code Freeze
- Only Critical Bug fixes and refactoring
- All documents finalized
- Senior Project Technical Report

5-5-14 (Sprint 12)
- Go /No Go?
- Push to Production
- Only Critical Bug fixes

- Senior Project Artifacts

5-12-14
- Senior Project Artifacts
- Senior Project Presentation

5-19-14
- Final Review

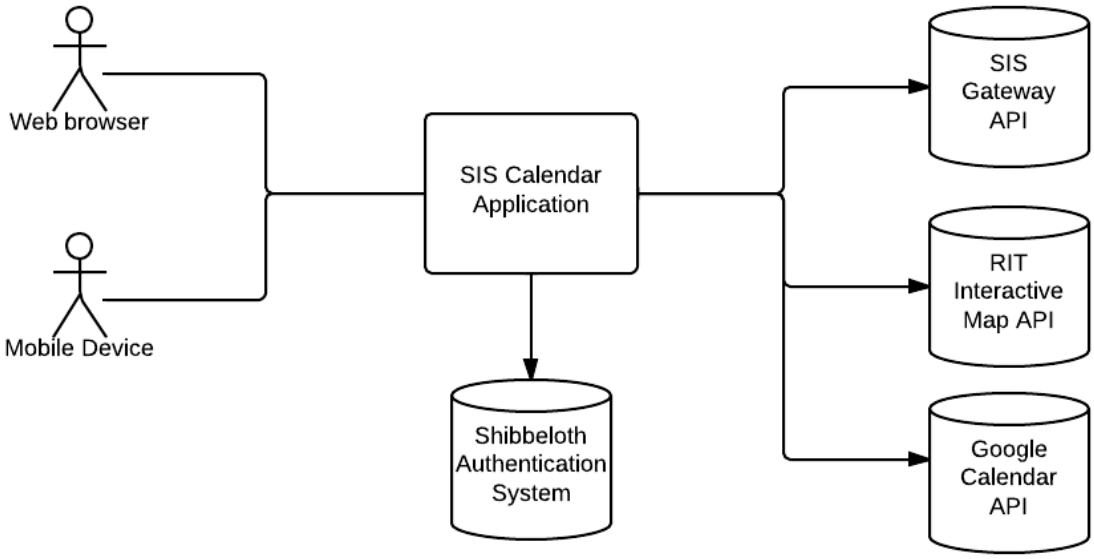
The great thing about using scrum is that each week our schedule was updated based on the velocities of previous sprints. This meant that our sponsor was always informed of our progress and how long things would take to complete.

At the beginning of the semester we informed our sponsor that we would need at least two sprints before we could accurately gauge our velocity and create the schedule for the semester. While skeptical, they allowed us until sprint 2 to create the schedule. This allowed us to be very confident in our ability to execute and meet our schedule.

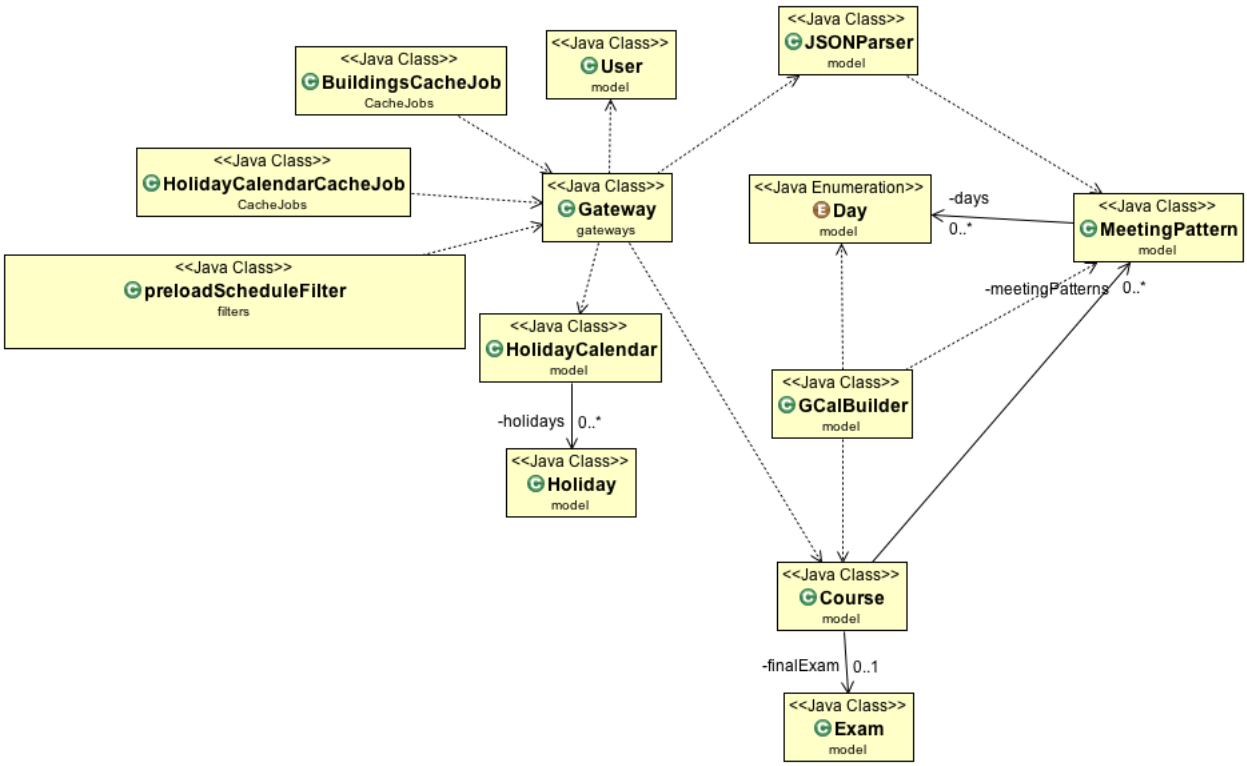
The biggest milestone of the semester was in sprint 12, the “Go/No Go” decision. This was when the sponsor black box tested SISCalendar and decided if it was ready to production. The final result was a small tweak that should be changed, but other than that it was good to go. We made the change and then were able to push the system to production.

System Design

The SISCalendar application interacts with several major components. These components are the Shibboleth Authentication System (used to authenticate and authorize a user before attempting to download a calendar), the SIS Gateway API (used to retrieve all calendar data for a given user), and the RIT Interactive Map API (used to generate a clickable URL to display a specific building on RIT’s campus). The application also interacts with Google’s Calendar API to allow students to automatically import their class schedule into Google calendar.



Below is a detailed diagram for the main SISCalendar Schedule subsystem. This is the system responsible for retrieving class and map information and manipulating that data to something usable by the user.



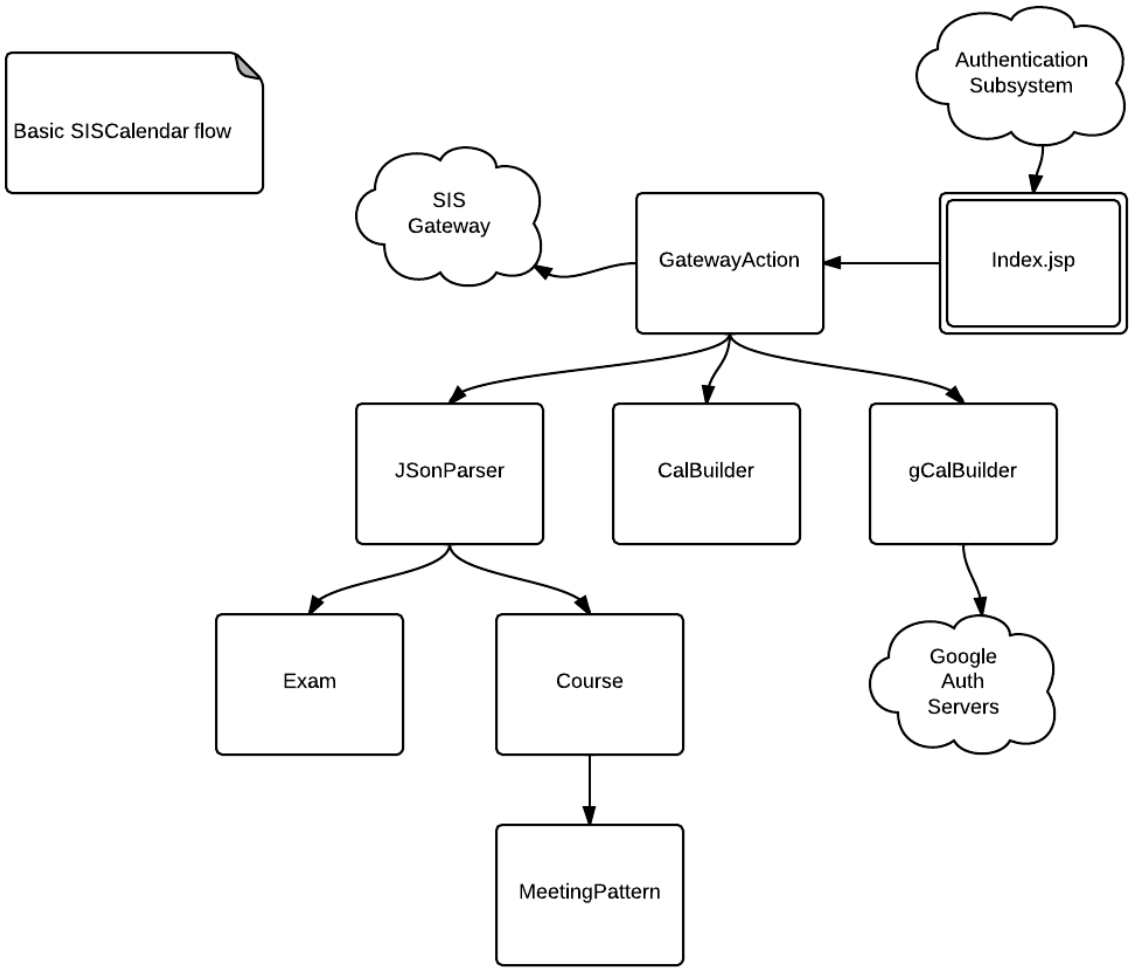
SISCalendar uses a Gateway class to interact with the external applications and web

services. The Gateway is responsible for downloading class schedule information, campus map and building information, and also for integrating with Google's calendar API.

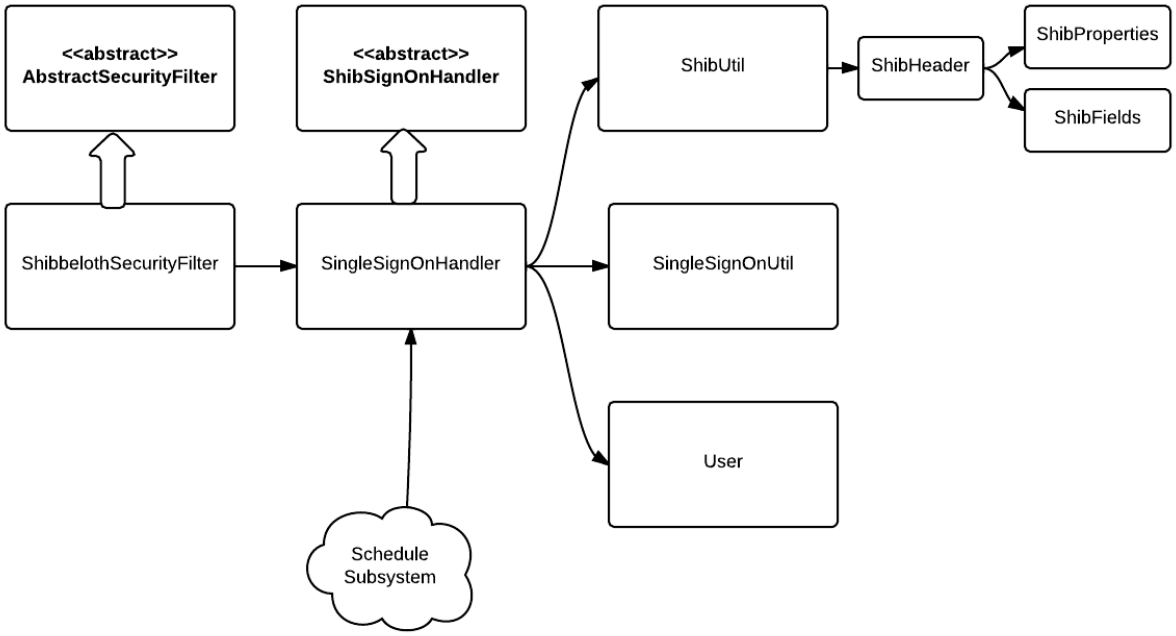
Once the Gateway has downloaded the necessary data, the data is formatted by the JSONParser. The parser is responsible for taking the JSON objects returned by the web services and creating the plain old Java objects (POJO) that the rest of the SISCalendar uses. SISCalendar uses a custom HolidayCalendar (used to store all of the known holiday information such as days off during the semester), a MeetingPattern class (containing information like days and times each class meets), a Course class (with information like professors and course title), and an exam class (similar to Courses, but only with the exam information for a specific course).

The preloadScheduleFilter is responsible for intercepting a clients request for the site and loading up all of the course and schedule information required to be displayed on the SISCalendar home page. This allows us to ensure that SISCalendar will always have information to display to a user before the users client finishes loading the site.

The next diagram helps to show the basic flow of events within the SISCalendar application as described above. Note that a user will initially enter from the Authentication subsystem, reaching Index.jsp once they are properly authenticated.



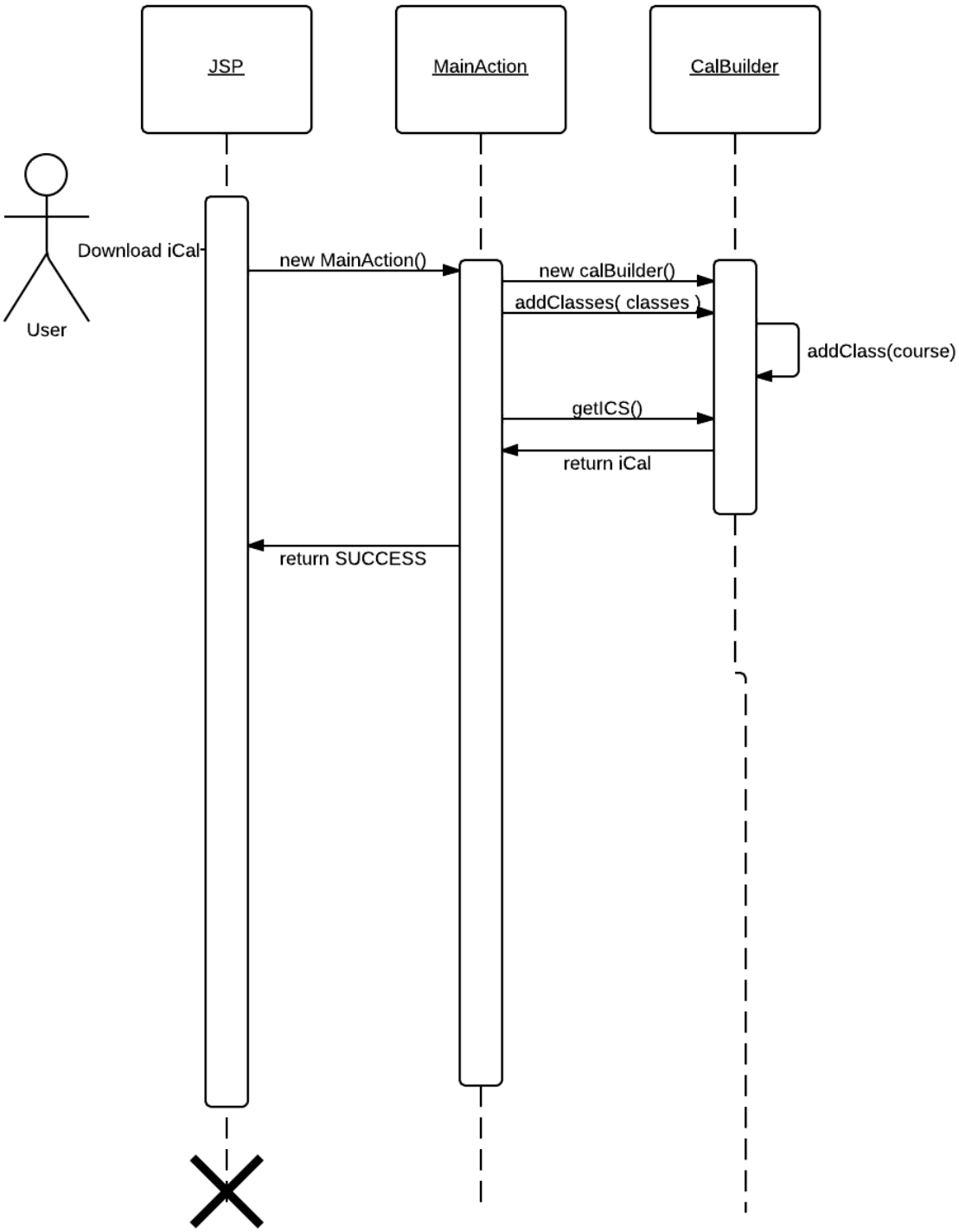
Initial class diagram of the authentication subsystem. This is based off of the authentication system put into place and currently used by the TeamBuilder project.



The following diagrams show the sequence of several common use cases. These cases are downloading an iCal file, logging in, and fetching a schedule.

Downloading an iCal

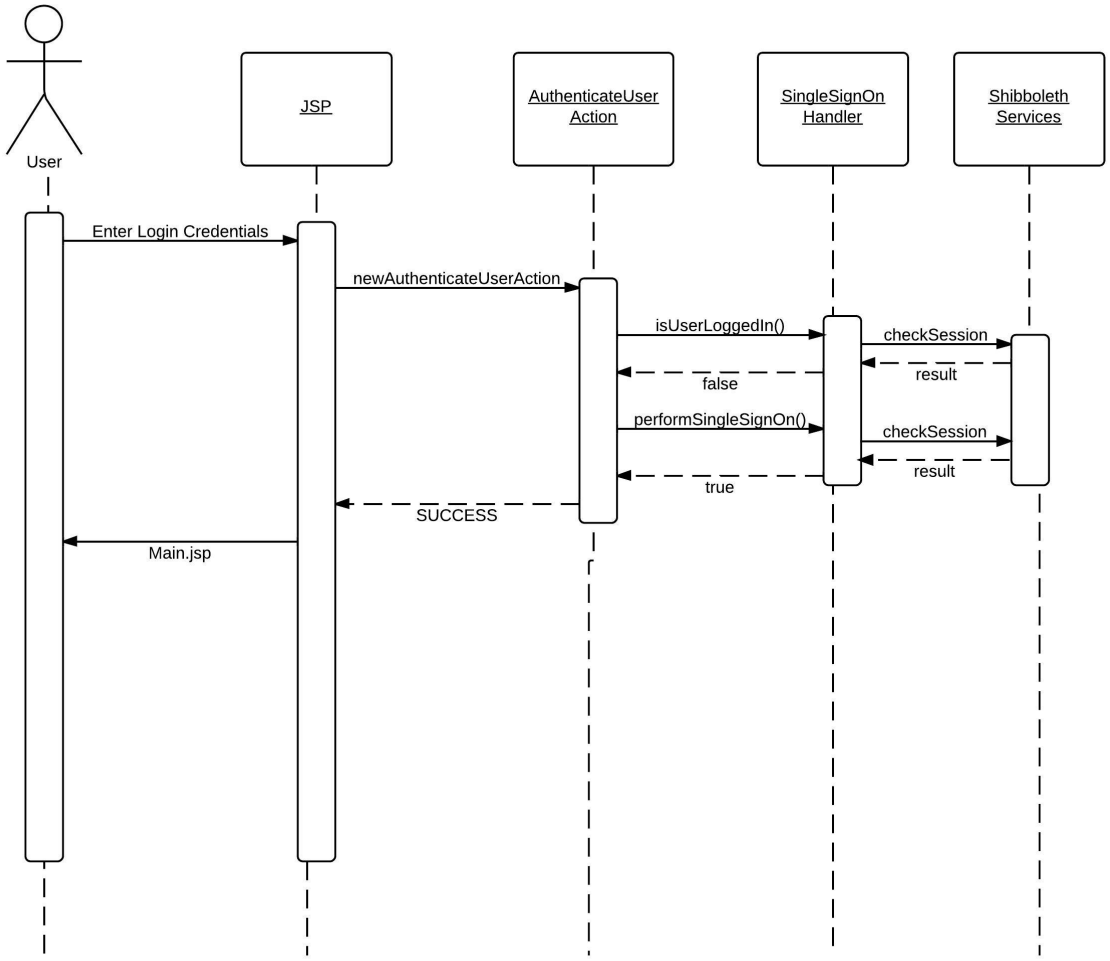
When a user clicks on the “Download Course” or “Download Exam” button in the iCal portion of the SISCalendar site, the following sequence of events are executed. First, the MainAction controller is created. The controller then creates a new CalBuilder object and passes the builder a list of classes to add to the iCal file. The CalBuilder adds each course to its file (including doing some error catching). Once built, the MainAction makes another request to the builder and is given the resulting iCal file. If the iCal was successfully created and returned as a download for the user, SUCCESS is returned.



Logging in

A user logs in using RIT's Shibboleth Authentication. In order to handle the authentication properly, we needed to implement several login handlers in SISCalendar.

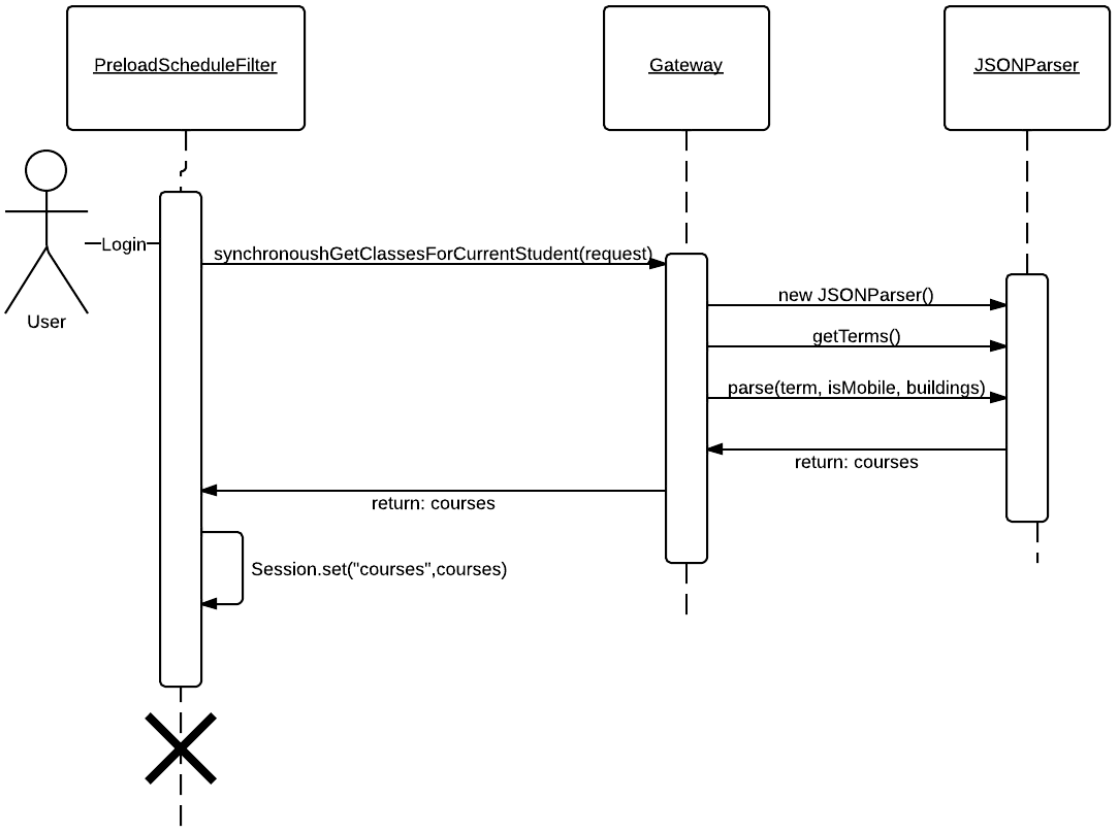
Once the user enters their login credentials we create a new AuthenticateUserAction controller. We check our SingleSignOnHandler to see if the user is already logged in (if they are, we can immediately download their calendar and bring them to the main page). We do this by checking the ShibbolethServices session. If false is returned, the AuthenticateUserAction controller calls the SingleSignOnHandler to perform a singleSignOn. This will attempt to authenticate with Shibboleth using the entered credentials and then check to ensure the user has been stored in Shibb's session. If the user is successfully authenticated and added to the session, SUCCESS is returned.



Fetching a schedule

As soon as a user is successfully authenticated, SISCalendar performs a call to fetch any existing schedules that the user has. This is done via the PreloadScheduleFilter - a Struts construct that is executed before index.jsp is rendered. The filter makes a call to the Gateway class to synchronousGetClassesForCurrentStudent(request), passing the current HTTPRequest object. The Gateway then instantiates the parser and gets the current displayable school terms (SISCalendar will only show the current term, the next term when available, and 2 previous terms). Once the Gateway has all

of the data it needs, it passes the information needed to the JSONParser to have a list of SISCalendar's Course objects created. Once the courses are returned to the preloadScheduleFilter, the filter adds the courses to the current session so that the jsp pages are able to access them when rendering.



A number of alternative designs were considered for SISCalendar. We considered omitting using Java at all in favor of javascript/php solutions. We eventually decided to go with Struts because ITS already had the infrastructure set up to handle Struts applications. This includes having deployment scripts created, servers running Tomcat configured for Struts applications, and plenty of applications already solving issues such as Shibboleth authentication that we knew SISCalendar would eventually need to solve. Using Struts also allowed us to perform a turnover to our sponsors without having to train them on a completely new technology stack. This made it easier for both the SIS.io team and the ITS team.

We decided to use the Struts 2 framework for our development for a number of other reasons as well. Struts 2 is the most up-to-date (stable) version of the Struts framework currently released by Apache (updated as of 09-21-13). This means that problems stemming from the framework itself were minimal. Struts is also (as stated above)

one of the approved technologies given to the student team as it is currently already supported by ITS in the Team Builder project. The Team Builder project had already solved the problem of Shibboleth authentication in the Struts 2 framework, which allowed the SIS.io team to reuse code shortening the expected (and actual) schedule and avoiding unforseen challenges caused by Shibboleth authentication with new technologies.

Struts 2 also provides a Model-View-Controller paradigm to applications developed in it. This allowed the SIS.io team to create a clean, reusable, and easily extendable application within the confines of the schedule.

Apache Ant was used in the creation of .war file builds for development, testing, and production. Ant is one of the ITS approved libraries for creating java builds for deployment, as well as being the library used by the Team Builder application. SIS.io was able to utilize Team Builders Ant scripts as examples when constructing our own, making the process as a whole easier.

SIS.io also used a private GitHub git repository to store our code. A master branch was stored on the account provided by RIT ITS, and each SIS.io developer was able to maintain their own local copy and feature branches.

For rendering and interacting with the client side, SISCalendar utilizes JQuery and normalize.css. JQuery allows the manipulation of data and UI elements easily and efficiently on the client side. Normalize.css is used to ensure SISCalendar has cross-browser consistency when the UI is rendered.

Process and Product Metrics

Over the two semesters, we employed the following metrics:

Comment Density: The number of CLOC (Comment lines of code) / LOC (Lines of code). This will be used to indicate how well the source code is documented.

Lines of code = 3643

Lines of comments = 949

Density = 21%

Not terrible, and a lot of the code we had we got from other sources and did not comment.

Cyclomatic Complexity: The number of linearly independent paths through a functions source code. This will be used to indicate the complexity of each of the functions within the project.

We tried to keep this low, although some parts of the code became a lot of nested if statements due to how the parsing library we used worked. This metric was not routinely collected.

Defect Density: The number of defects per line of code. This metric will be used to determine how error-prone our project is.

This was calculated to be 0.0036336. Over the course of the project we had a relatively small number of reported defects (defects that another team member caught and reported), but a large number of lines of code in javascript and java.

Test line coverage: The percentage of total lines of production code covered by automated tests at any dimension (unit, integration, graphical, etc).

We did not keep our unit tests up to date because our model was very transient and small. Our integration tests and acceptance tests were used to see if the system was working, so calculating code coverage never became something we did on a regular basis.

Hours of effort: The total number of hours, doing any work for senior project, per developer.

We used Toggl for time tracking. For semester 1, we worked 454 h 42 min. For semester 2, we worked, 409 h 32 min.

Google Analytics: Generate detailed statistics about the web application's traffic and traffic sources.

Users: 332, pageviews: 782, Average session duration: 00:02:10.

Class Imports: 69, Final Imports: 46, Class Downloads: 40, Final Download: 21.

Jira tasks closed per individual: The number of tasks per week that a team member successfully closes.

Average tasks closed was 8, min was 2, max was 15.

Average time Jira task is open: The average amount of time an individual team member leaves open Jira tasks.

Average time was: 10 days 14 hours.

Number of late tasks: The number of tasks per week that a team member delivers “late” - where late is defined as Sunday night or later before our Tuesday meeting with ITS.

Average: 4.

This metric became skewed when our dev server went down for 2 weeks - all our tasks became late even if we had most of it finished. Some people did better than others, but for the most part we got the tasks done on time.

Trello tasks closed per individual: The number of tasks per sprint that a team member successfully closes.

Ethan: 35, Mike: 27, Zach: 22, Shawn: 37

Velocity: The amount of story points that were finished in each sprint.

Sprint 1: Estimated: 24, Actual: 19

Sprint 2: Estimated: 27, Actual: 19

Sprint 3: Estimated: 20, Actual: 22

Sprint 4: Estimated: 10, Actual: 9

Sprint 5: Estimated: 32, Actual: 28

Sprint 6: Estimated: 24, Actual: 20

Sprint 7: Estimated: 20, Actual: 15

Sprint 8: Estimated: 30, Actual: 25

Sprint 9: Estimated: 20, Actual: 22

Sprint 10: Estimated: 15, Actual: 16

Sprint 11: Estimated: 30, Actual: 34

Sprint 12: Estimated: 8, Actual: 7

Overall, we did not use the metrics as much as we should have, with the exception of velocity, which we calculated every week and used for sprint planning. Our metrics show that the project went decently, well, and we didn’t have any catastrophic failures. The metrics for our code base actually show a product with few defects, but could have used more tests to prove that.

Product State at Time of Delivery

The state of our product is that it is currently in production for students at all of RIT’s campus’ to use. We completed all of the original requirements for this project, and went above and beyond the sponsor’s original requirements. The original requirements

of a student being able to download their class schedule and location on RIT Maps was finished in the first semester. For the second semester we came up with a bunch of enhancements, and had the sponsor select the most important ones they would like. Of those, we implemented: downloading exams, honoring holiday calendars, and importing into Google Calendar. Additional features that we were not able to finish, but are areas for future enhancements are to have the application remember what you last downloaded and provide an ical file that's just the differences. Also to have the application automatically update a student's google calendar whenever a change is detected on their schedule. Overall the sponsor is very happy with the level of work we completed and the finished product we delivered to them.

Project Reflection

What went right?

We were able to manage our scope very well throughout both semesters. During the first semester when we were gathering our requirements, we found many features that were deemed as nice to have but not necessary. We wrote these features down but put all of them on the back burner for release 2. For release 1 we wanted to just have the core requirements for the project complete - downloading a schedule to iCal, and viewing classes on the RIT map. That way when we went to release 2 we would be able to have a successful project.

Because we were able to finish up the core requirements in R1 we decided to switch to Scrum because we knew we weren't going to be able to finish all of the features we had on our backlog and wanted to have a working build at the completion of each of our 1 week sprints. We think that choosing 1 week sprints instead of 2 was beneficial to the project because we found that sometimes we were waiting until the end of the sprint to finish our assigned tasks. The change to Scrum was also very beneficial to our team because of the flexibility it provided when building the non-required but nice to have features. No matter where a sprint ended, we knew we would have a completely functional product to ship.

Our sponsor worked with us closely to make the project successful. They were very responsive to our questions. Sometimes answering questions and debugging problems with us into the night. We also had a public trello board where they could update us on any progress or problems that occurred during the sprint. This, along with our teams private board, provided us and ITS with one place to go for any tasks and updates on the project.

Using Git and GitHub instead of SVN was also very beneficial to our team. All of our team members were more comfortable with Git than SVN which meant there was less learning involved and less chance of mistakes. Git also allowed us to very easily make branches for each feature being developed, ensuring a clean and efficient development process. By only allowing reviewed code to be merged into the Master branch we were also able to ensure that only correct and tested code was ever deployed to the production and test servers.

What went Wrong?

ITS was having a refresh of our development server. This was so that the development server's data could be a more accurate portrayal of the production data, which would allow us to test better because it would be closer to the production setup. This was going to cause our dev server to be down for 72 hours. Unfortunately, because of several issues that ITS had the dev server was down for about a week. Then we had a blizzard and RIT lost power. This caused our server along with many other production services that ITS supports to need immediate maintenance, putting our dev server lower on the priority list for ITS. So in all our dev server was down for about another week, or two weeks total. During this time we were unable to test our code with real data, which was part of our User Story completion criteria. Luckily we had some mock data that we could use to continue development, so features were still being implemented, but we were unable to mark any of our tasks as done for those two sprints because it wasn't tested against real data.

Testing was also difficult to do because of the nature of the SIS system. We had test accounts that ITS made us, but we had to create a schedule for those test accounts and then register for classes. But because these were test accounts, many times they didn't have any prerequisites for classes. This made it difficult to get into the types of classes we wanted to test. When the server refresh happened all of these test accounts got deleted, so we had to go through the process over again.

What would you do differently?

When we started scrum during the second semester we initially had trouble finding a good way to do our standups without being physically in the same location. Between work and class schedules amongst team members, there was no time to meet except for 9pm most days of the week. We started out saying we would use google hangouts, but nobody used this and so we decided to switch to another group texting app called GroupMe so that we would get notifications when one person did their standups. But the app caused problems for multiple users. It drained one of our team members phone battery in a couple hours and another team member didn't get any notifications from GroupMe. We decided to finally switch to a much better group chat application FastChat, which gave us all of the functionality we needed for our daily standups.

We should have spent more time setting up a good way to test our application. Because of the variety of different accounts and classes that a user could have it would have been beneficial to focus more on testing. That way we could have been more confident launching our application into the production environment. One of the issues was that ITS didn't have a good system setup for creating regression tests. We did set Selenium up, but didn't use it often enough because of the time required to setup good tests with it. If we could do it again, we would have added more time for setting up, running, and using Selenium tests in our schedule instead of leaving it up to each team member to create their own.