CSCE 606 Spring 2020

Project Report

Team: Cucumber Junction
Project: ZLP Lecture Scheduler

05/05/20

Table of Contents

Summary	3
Team roles	3
Important Links	4
User stories	4
Summary for each iteration	14
Customer meetings	15
BDD/TDD process	17
Configuration management approach	17
Issues with Heroku	17
Issues with other Tools	18
Tools/GEMs used	18

I. Summary

The Zachry Leadership Program (ZLP) is a 5-semester certificate program which selects 32 undergraduate sophomore engineering students each year to participate. These students must take part in each semester consecutively with the same cohort they were admitted. During each semester, every student must attend a 2-hour lecture period once a week with their cohort members. In the past, choosing a lecture time that works with all students varying schedules has been very difficult and time consuming. Thus, the aim of this project is to create an application that could analyze all students tentative schedules before registration and select a time for the lecture period that works with the schedules.

The application has a secure login for both administrators and students, the main stakeholders. The administrator does a mass uploading of student information which ensures access is allowed only for pre-approved student users. Students are allowed to choose courses for their schedules from a set of confirmed TAMU courses. Administrators can choose an active semester and define a time range for students to add schedules for it. After the students add their schedules, the admin runs the algorithm to find the most suitable schedule.

II. Team roles

Sl.No.	Role	Name	Email
1	Product Owner	Lauren Haylock	laurenrhaylock@tamu.edu
2	Scrum Master	Abhishek Deb	abhishek_1014@tamu.edu
3	Backend developer	Matthew Fitzpatrick	matthew.fitzpatrick@tamu.edu
4	Backend developer	Arman Rezaee	arezaee@tamu.edu
5	Backend developer	Parsa Dastjerdi	parsadastjerdi@tamu.edu

The team roles remained consistent during the project and no change was made to any of them.

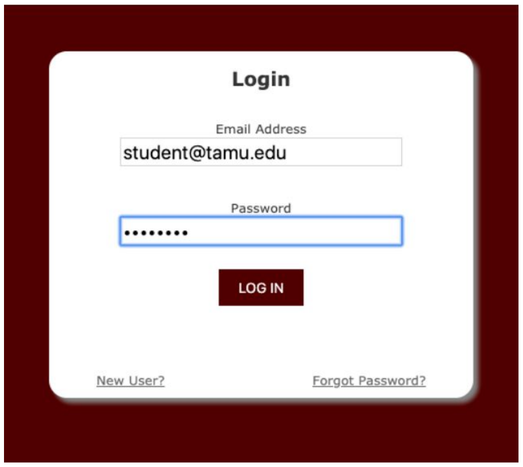
III. Important Links

GitHub: https://github.com/zlp-scheduler/zlp-scheduler
Pivotal Tracker: https://www.pivotaltracker.com/n/projects/2435699
Heroku deployment: https://tranquil-escarpment-17896.herokuapp.com/
Demo: https://youtu.be/Fuu57ewseGk

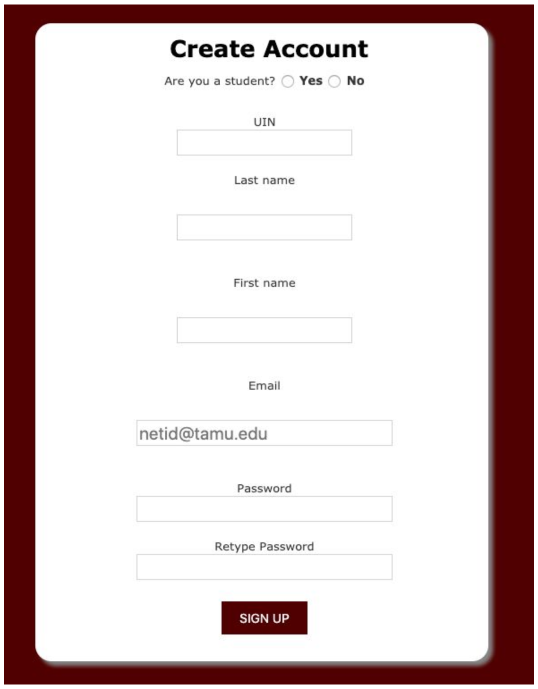
IV. User stories

1. (Points: 1) As a user, who is not registered, when I navigate to the root URI, I can click a "Register New User" button/link which takes me to a page where I can configure my user details.
    a. Required Information
        i. UIN
        ii. E-mail (MUST end in "@tamu.edu")
        iii. Password + Password Confirmation
        iv. Account Type (Student, Administrator)
    b. Optional Information
        i. First Name/Last Name

The implementation of this user story is completed. When the user goes to the root URL, there is a *New User?* Button which redirects to the registration page. The registration page screenshot is given in figure 1 (b).



(a) Root Url containing the New User link

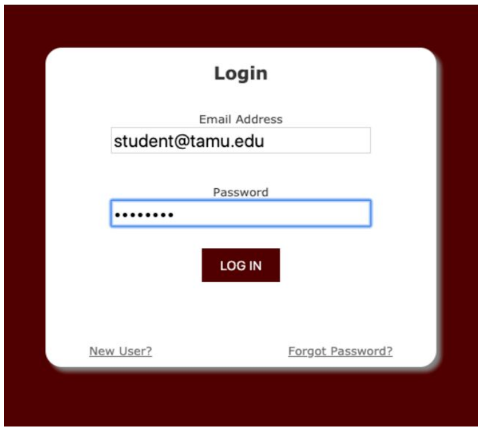


(b) Registration form

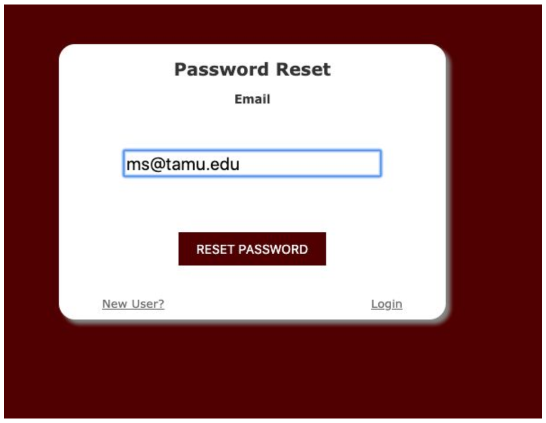
Figure 1: Sign up Feature

2. (Points: 2) As a user, who is registered but is not logged in, when I get to the "Login" page, there is a forgot/reset password button that will allow me to reset my password somehow.

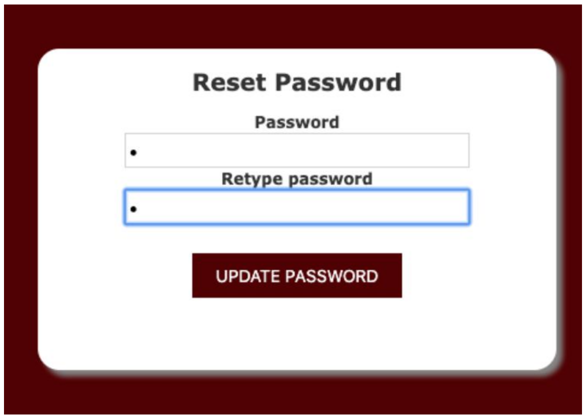
   The implementation of this user story is completed. When the user goes to the root URL, there is a *Forgot Password* Button which redirects to the Forgot Password page where the user enters his/her email to get a password reset link which then takes him to the Reset password page. The screenshots in figures 2(b) and 2(c) illustrate this.



(a) Root Url containing the Forgot Password link



(b) The Forgot Password page for entering email

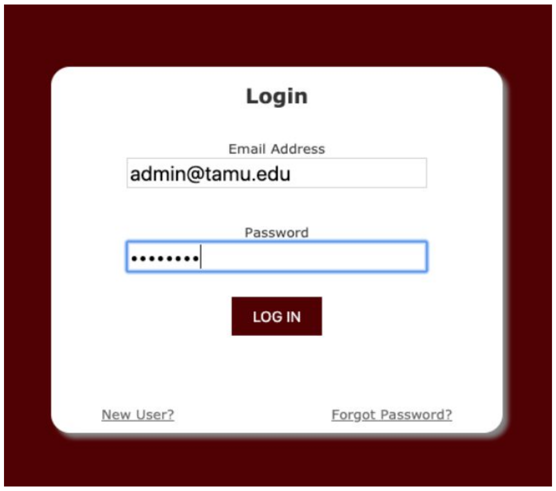


(c) Reset password page

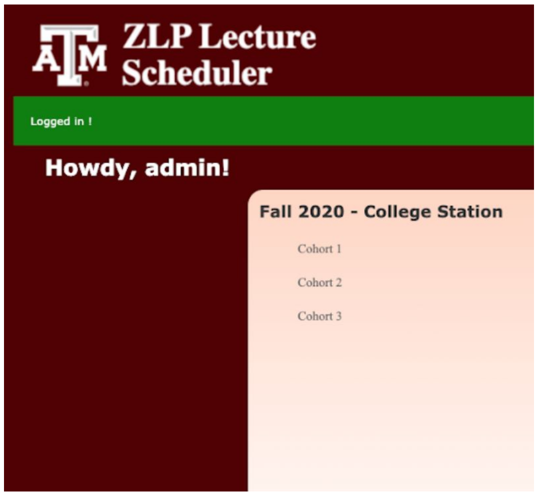
Figure 2: Forgot Password feature

3. (Points: 3) As a user, who is not logged in, when I navigate to the root URI, I will be presented with a log-in screen. When I log-in, the application will automatically know if I am considered an Admin or a Student and show the corresponding "Home" page.

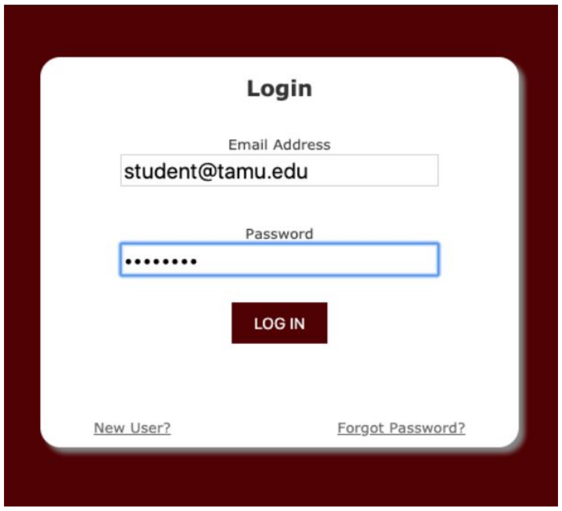
   The implementation of this user story is completed. When the user goes to the root URL, there is a login page which contains the *Log-In* screen, the *New User* registration link and the *Forgot Password* link. Further the backend of the login is set up in such a way that based on the credentials it identifies if the user is an admin or a student and redirects them accordingly. The screenshots in figure 3 illustrates this.



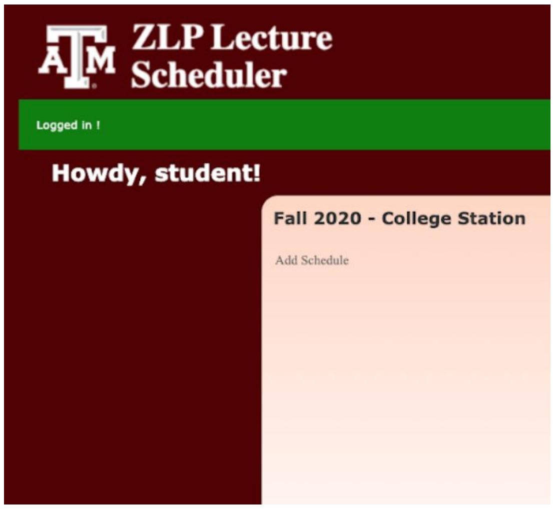
(a) *When admin logs in*



(b) *Redirects to the cohorts page*



(c) *When student logs in*



(d) *Redirects to the schedules page*

Figure 3: Login feature

Figure 4 gives a flow of user stories 1, 2 and 3. These three user stories didn't undergo any changes during the course of the project.

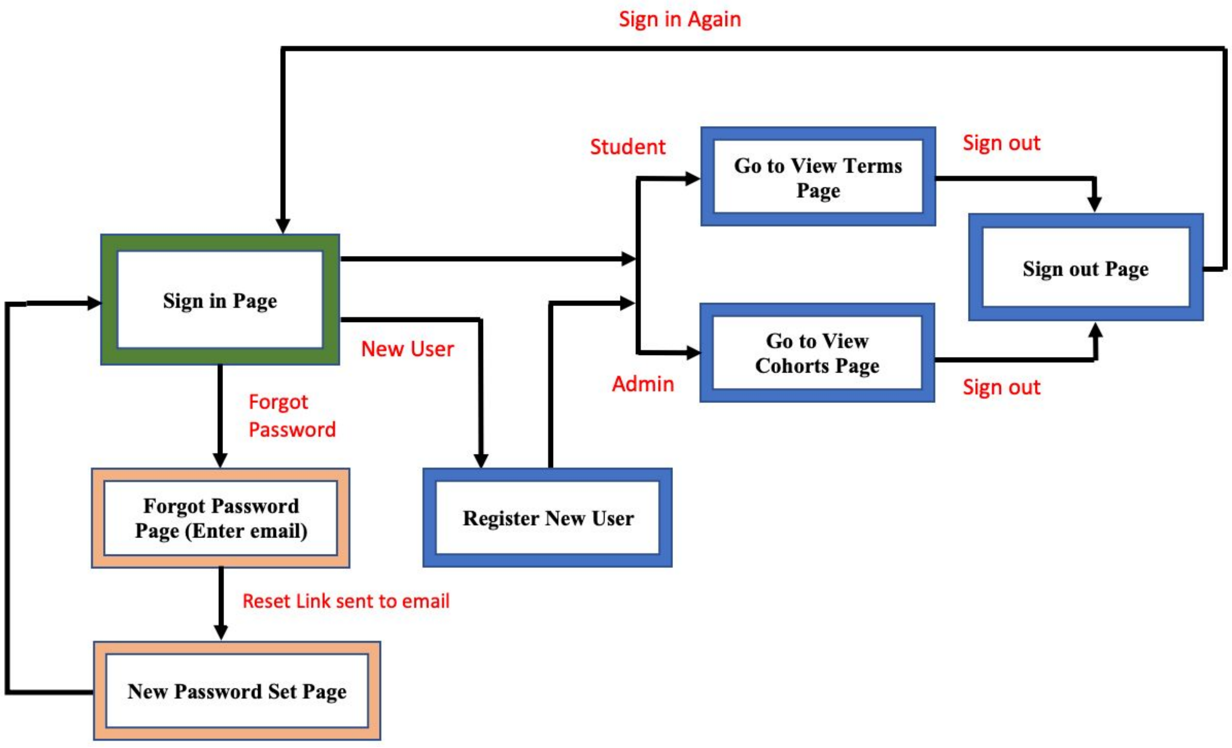
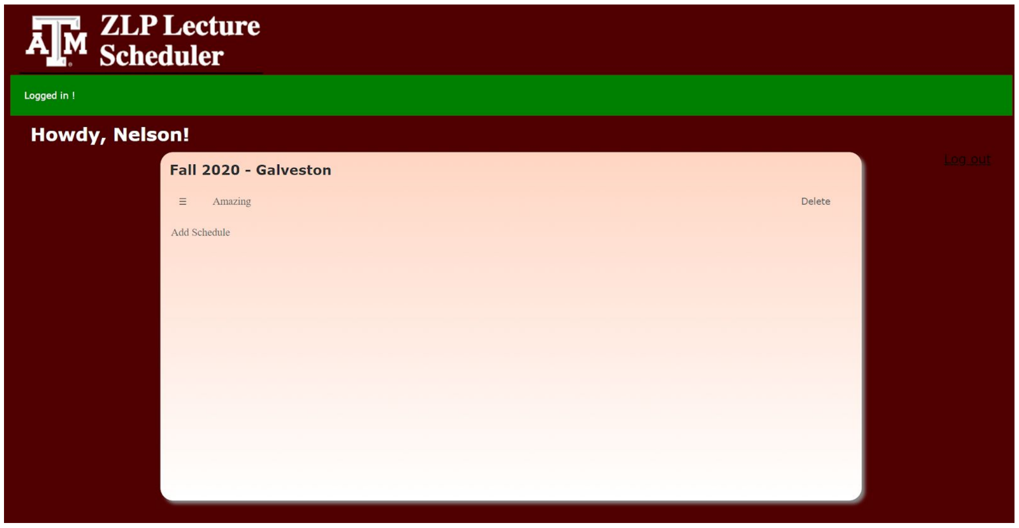


Figure 4: Flow of user stories 1, 2 and 3

4. (Points: 1) As a student, when I login, I am shown the "View Term" page, where I can see the active & open term and the schedules I have added to it. If there are no schedules, all I can see is the 'Add Schedule' button.



ZLP Lecture Scheduler

Logged in !

Howdy, Nelson!

Log out

Fall 2020 - Galveston

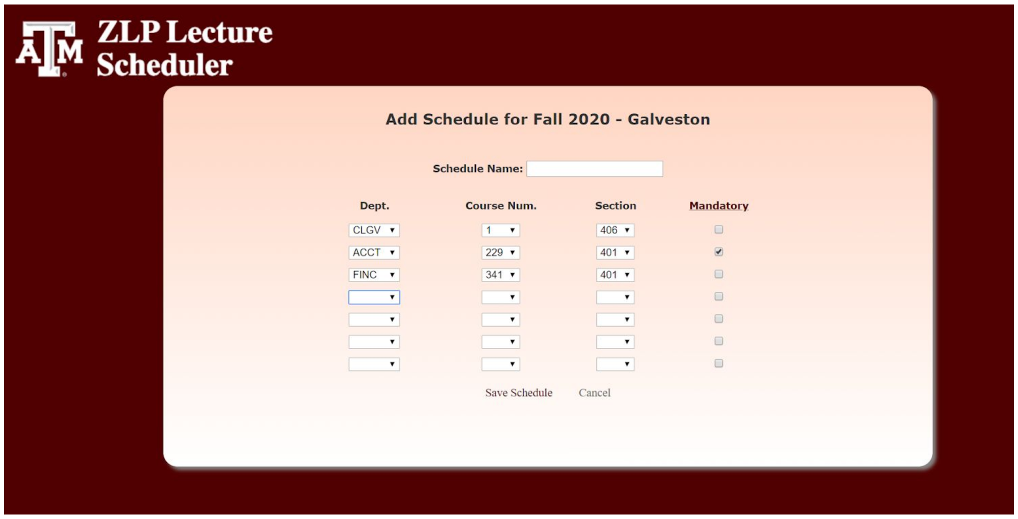
≡ Amazing Delete

Add Schedule

Figure 5: View Term page for student

Figure 5 is what the student is shown if they are currently in the time range that that administrator has defined as ‘open’ for students to add schedules. If not, they are redirected to a page that says “The director has not opened a semester for you to add schedules to. Please try again later.”

5. (Points: 3) Add Schedules
   i. As a student, when I click on the “Create New Schedule” link/button, I am taken to the “Create Schedule” page.
   ii. As a student, when I am on the “Create New Schedule” page, I can see a set of drop downs that I can use to build a schedule.
      1. TextBox for a “nickname” for the schedule
      2. Dropdown Inputs
         a. Department
         b. Course #
         c. Section
         d. Mandatory? (Checkbox that should indicate if it is required for graduation)
   iii. As a student, when I have finished creating a schedule, I can then hit a “Save Schedule” button that will take me back to the “View Schedules” Page where I can now see my existing schedules plus the one I just created.



Add Schedule for Fall 2020 - Galveston

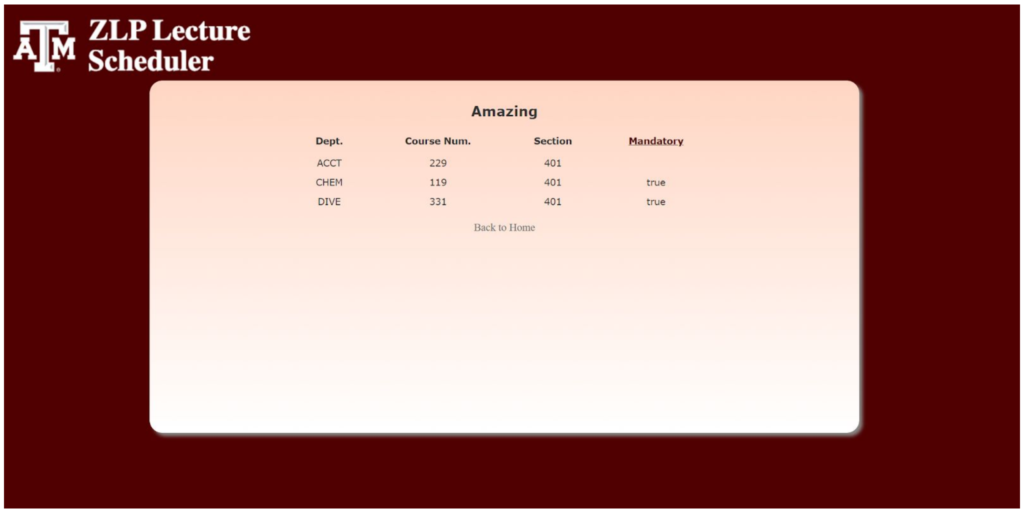
Schedule Name:

Dept.	Course Num.	Section	Mandatory
CLGV	1	406	☐
ACCT	229	401	✓
FINC	341	401	☐
			☐
			☐
			☐
			☐

Figure 6: Add Schedules page

On this page, the first drop-down for the department is filled with subjects that belong to the active term. Once the student chooses a subject, then the course number drop-down is populated with course numbers for the corresponding subject. Once the course number is chosen, then the section drop-down is populated with section numbers for that subject and course number. This data is scraped from the TAMU Course Search website the day before the active terms 'opening' date.

6. (Points: 2) As a student, when I select a specific schedule from the "View Term" page, I am taken to the "View Schedule" page.



Amazing

Dept.	Course Num.	Section	Mandatory
ACCT	229	401	
CHEM	119	401	true
DIVE	331	401	true

Figure 7: View Schedule Page

7. (Points: 1) As an administrator, once I successfully login, I want to see the "View Term" page, which shows the active term and a list of cohorts that are participating in that term.

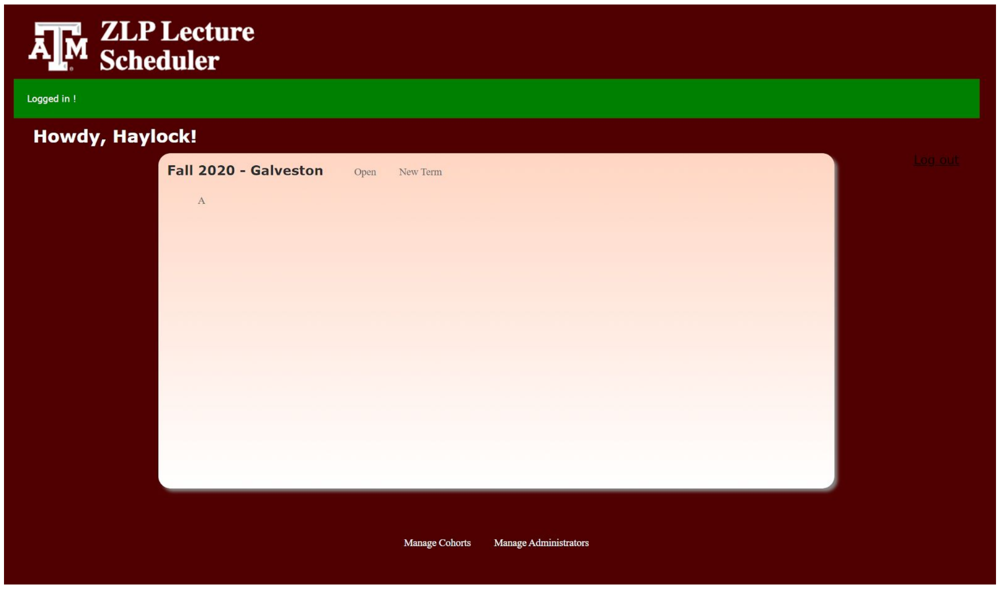
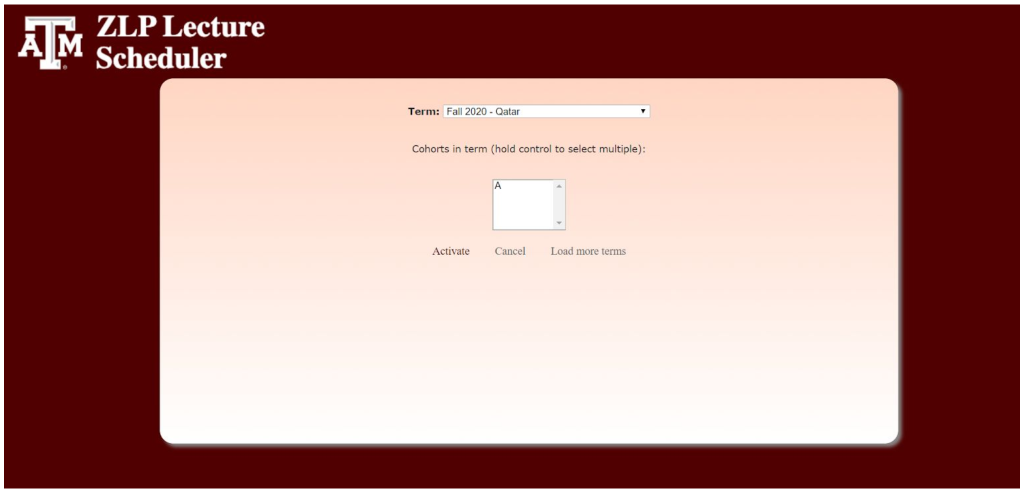


Figure 8: View Term Page for admin

8. (Points: 3) As an administrator, I want a "New Term" page, where I can choose a new term to activate and the cohorts that will participate in that term.



Term: Fall 2020 - Qatar

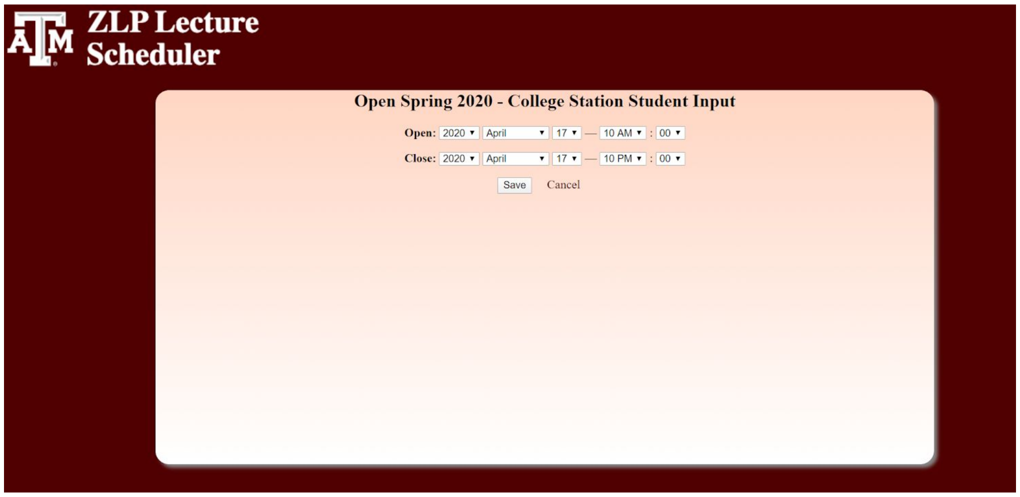
Cohorts in term (hold control to select multiple):

A

Figure 9: New Term page

The terms for the drop-down are scraped from the TAMU Course Search website when the admin clicks the 'Load more terms' button. Once an admin activates a term, it shows up on all dashboards, and the schedules from the previously active term are destroyed.

9. (Points: 2) As an administrator, I want an "open" page that allows me to open a period of time for the students to enter schedules for the active semester.



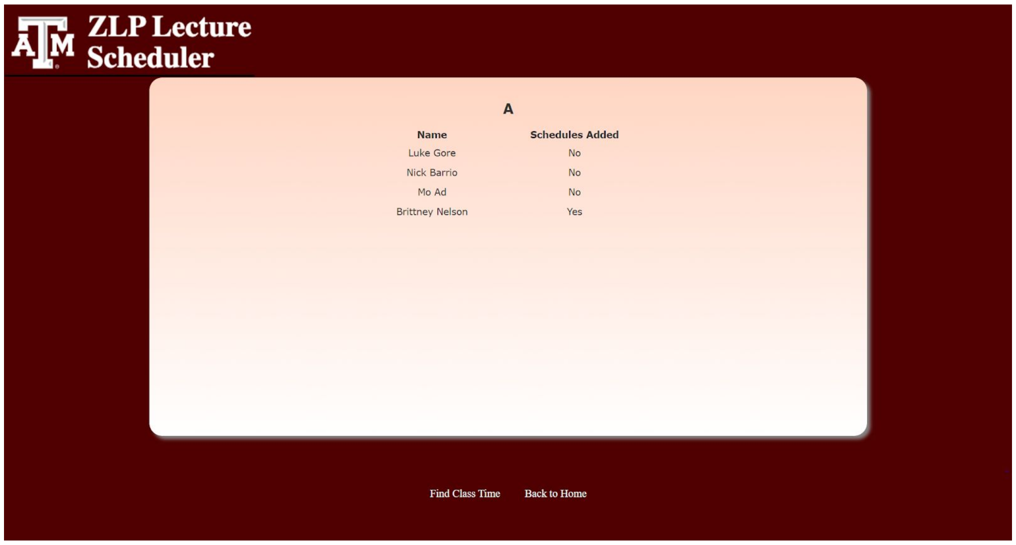
Open Spring 2020 - College Station Student Input

Open: 2020 April 17 — 10 AM : 00

Close: 2020 April 17 — 10 PM : 00

Figure 10: Open page

10. (Points: 2) As an administrator, when I click on a particular cohort in the active term, I want to view the list of students in that cohort and whether or not they have added schedules for the term.

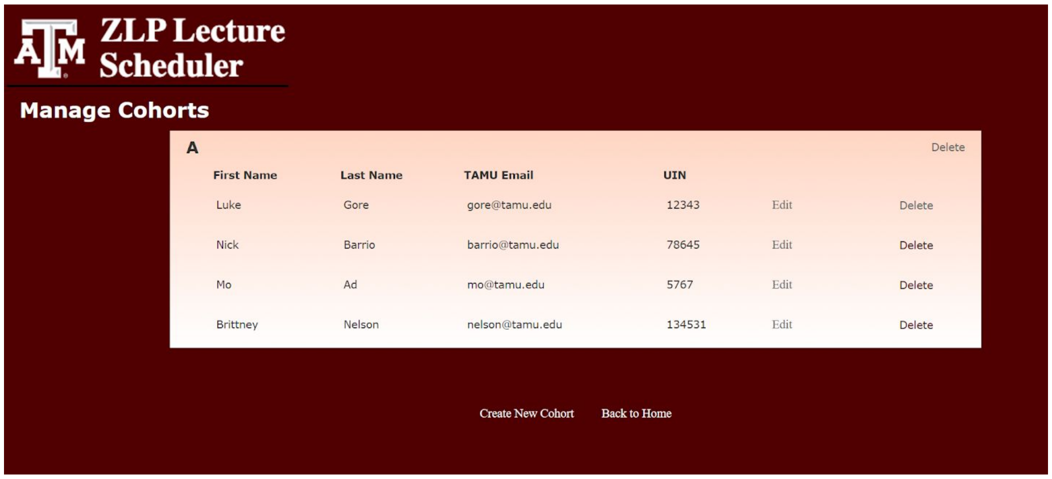


A

Name	Schedules Added
Luke Gore	No
Nick Barrio	No
Mo Ad	No
Brittney Nelson	Yes

Figure 11: View list of students and their status

11. (Points: 2) As an administrator, when I click on "Manage Cohorts," I am taken to the manage cohorts page where I can see a list of all cohorts, all students in each cohort, add/delete cohorts, and add/edit/delete students.

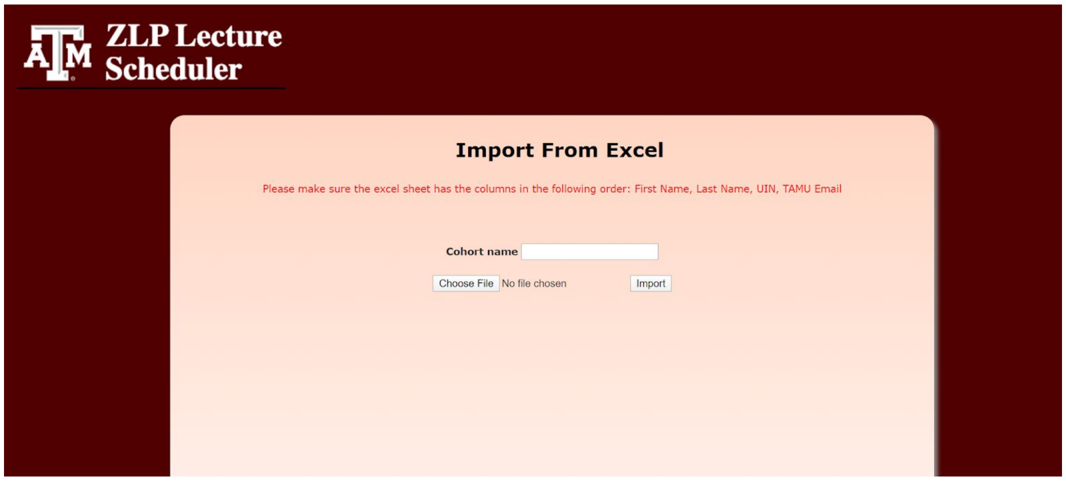


A

First Name	Last Name	TAMU Email	UIN		
Luke	Gore	gore@tamu.edu	12343	Edit	Delete
Nick	Barrio	barrio@tamu.edu	78645	Edit	Delete
Mo	Ad	mo@tamu.edu	5767	Edit	Delete
Brittney	Nelson	nelson@tamu.edu	134531	Edit	Delete

Figure 12: Manage Cohorts page

12. (Points: 3) As an administrator, when I click on "Add Cohort," I am taken to the add cohort page where I can upload an excel file with my students information.


ZLP Lecture Scheduler

Import From Excel

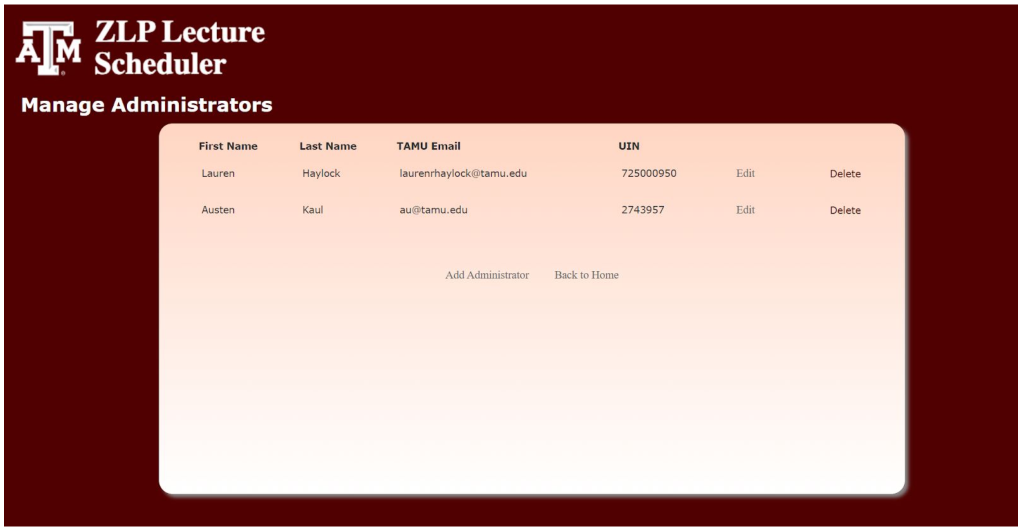
Please make sure the excel sheet has the columns in the following order: First Name, Last Name, UIN, TAMU Email

Cohort name

Choose File No file chosen Import

Figure 13: Add Cohorts page

13. (Points: 2) As an administrator, on the View Term page, I want a link to the Manage Administrators page, where I can add, delete, or edit other administrators of the system.


ZLP Lecture Scheduler

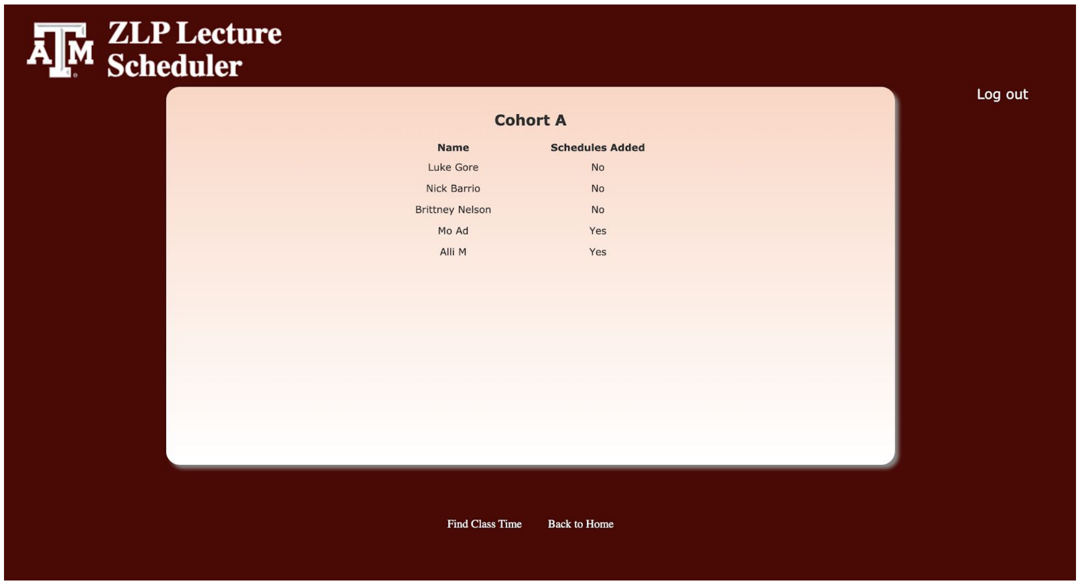
Manage Administrators

First Name	Last Name	TAMU Email	UIN		
Lauren	Haylock	laurenrhaylock@tamu.edu	725000950	Edit	Delete
Austen	Kaul	au@tamu.edu	2743957	Edit	Delete

Add Administrator Back to Home

Figure 14: Manage Administrators page

14. (Points: 3) As an administrator, who has the schedules of all students, I want to be presented with the optimum time slot for the class and students who have a conflict with that time. The admin can click on “Find Class Time” in each cohort to get an email with the optimal schedule. This user story can be broken down into three pieces listed below: converting schedules from the backend into a matrix format, running the optimization algorithm on the matrices, and passing the optimal time slot and schedule combination back to the administrator.



Cohort A

Name	Schedules Added
Luke Gore	No
Nick Barrio	No
Brittney Nelson	No
Mo Ad	Yes
Alli M	Yes

Figure 15: View list of students and their status

Above is the page that the administrator will be presented with that contains all student schedules for a specific cohort. This was also presented in user story 10. Once every student has completed his or her schedule, the administrator will be able to press "Find Class Time" to find the optimal class time. Figure 16 is a high-level diagram of the entire algorithm. The green indicates completed sections, and the red indicated incomplete sections. We have not been able to implement a method to mail the administrator the optimal time slot and optimal schedules yet, but it will be completed during the summer.

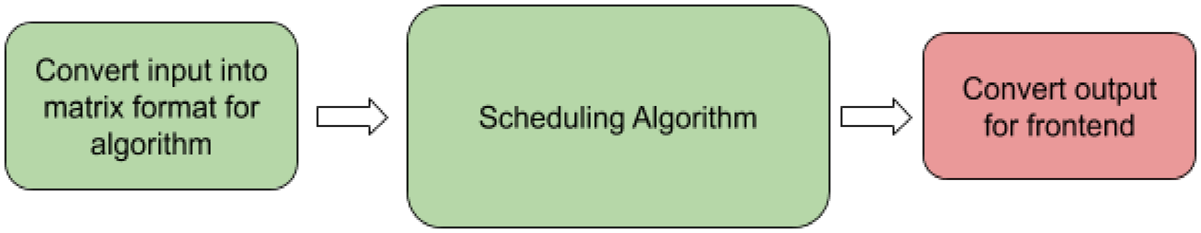

Convert input into matrix format for algorithm
Scheduling Algorithm
Convert output for frontend

Figure 16: Core Algorithm

Overall, this user story is incomplete due to the last step in the algorithm being incomplete. This user story as a whole is too large to be contained in one user story and had to be subdivided into three separate user stories.

15. (Points 2) As a scheduling algorithm, who is provided with a set of students who each have a set of schedules, I want to convert these schedules into a set of matrices that can be manipulated to perform optimization on. Each schedule will be represented by a matrix of size (days x number of time slots) where the number of time slots is the total

number of 5 minute intervals between 8:00am in the morning and 5:15pm in the afternoon.

The implementation for this story is completed. The algorithm will retrieve all students within the database and convert their schedules into a matrix format. This is done within the MatrixGenerator class in the file controllers/concerns/matrix_generator.rb. This step had to be redone after the User model was created, since we hadn't settled on the final version of the database.

16. (Points 2) As a scheduling algorithm, who is provided a set of matrices to perform optimization on, I would like to return a combination of schedules that provide the minimal amount of conflict in a 2 hour gap. Therefore, I will iterate through all combinations and for each combination I will find the least amount of conflict (minimization).

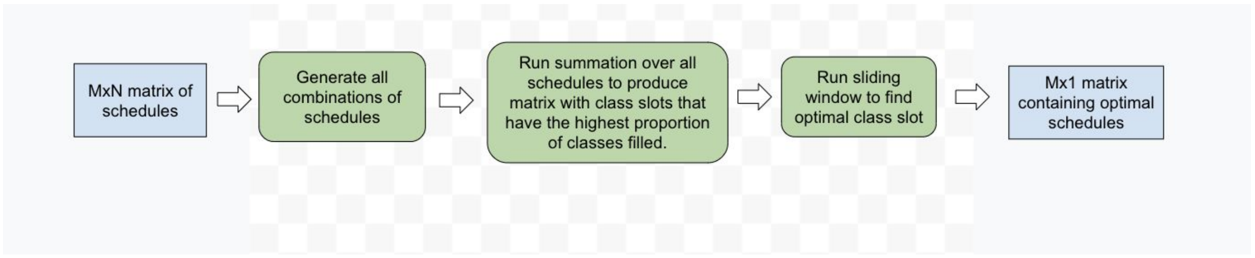

MxN matrix of schedules
Generate all combinations of schedules
Run summation over all schedules to produce matrix with class slots that have the highest proportion of classes filled.
Run sliding window to find optimal class slot
Mx1 matrix containing optimal schedules

Figure 17: Core Algorithm I/O

Figure 17 is a simple diagram of the scheduling algorithm's inputs/outputs and internal methods.

The implementation for this story is complete. The algorithm will take a 2D array of schedules, where each schedule is in matrix format, and run a brute-force optimization algorithm on the matrices. After it has computed the optimal time slot (least amount of conflicting schedules) it will return the optimal combination of schedules, the optimal time slot, and a 2D "heatmap" that consists of all the student's schedules. These can all be accessed through the Scheduler object with simple method calls like Scheduler.optimal_val, Scheduler.optimal_index, etc.

17. (Points 2) As a scheduling algorithm, who is provided with the optimal combination of schedules, optimal time slot, and a 2D heatmap of all conflicts of schedules, I will provide this information to the administrator through an email (or through a web interface).

This user story is incomplete. As of now, we are planning on running the algorithm on a separate background process and then sending the result to the admin through an email. While testing, we were able to compute the output within 4-5 seconds on a set of 16 students, with 3 schedules each. This still needs to be expanded to a minimum of 30 students as the algorithm's runtime is exponential with respect to the input, so it will grow rapidly as more students/schedules are added.

18. (Points: 1) As a scheduling algorithm, I would like to have a set of test data to use to ensure that the algorithm is working at all stages.

19. (Points: 1) As the backend, I should be able to issue a web request to the TAMU schedule website and get back the desired set of data for a list of possible Terms. (I.E. Spring 2020, Fall 2020, Spring 2021, Fall 2021) etc…

20. (Points: 1) As the backend, I should be able to issue a web request to the TAMU schedule website and get back the desired set of data for a list of possible subjects for a term. (I.E. Accounting, Engineering, Spacewalking, Basketweaving) etc…

21. (Points: 1) As the backend, I should be able to issue a web request to the TAMU schedule website and get back the desired set of data for a list of possible courses for a subject and term. (I.E. Accounting, Engineering, Spacewalking, Basketweaving) etc…

22. (Points: 2) As the backend, I should have Ruby Models to represent a Course, Term, and Subject along with database relations that tie them all together.

23. (Points: 2) As the backend, I should have the ability to parse the JSON received from the TAMU website into the Ruby Models created in story 21.

24. (Points: 3) As the backend, I should expose a comprehensive interface that allows other classes to easily import and save the scheduling data from TAMU into the ZLP-Scheduler database.

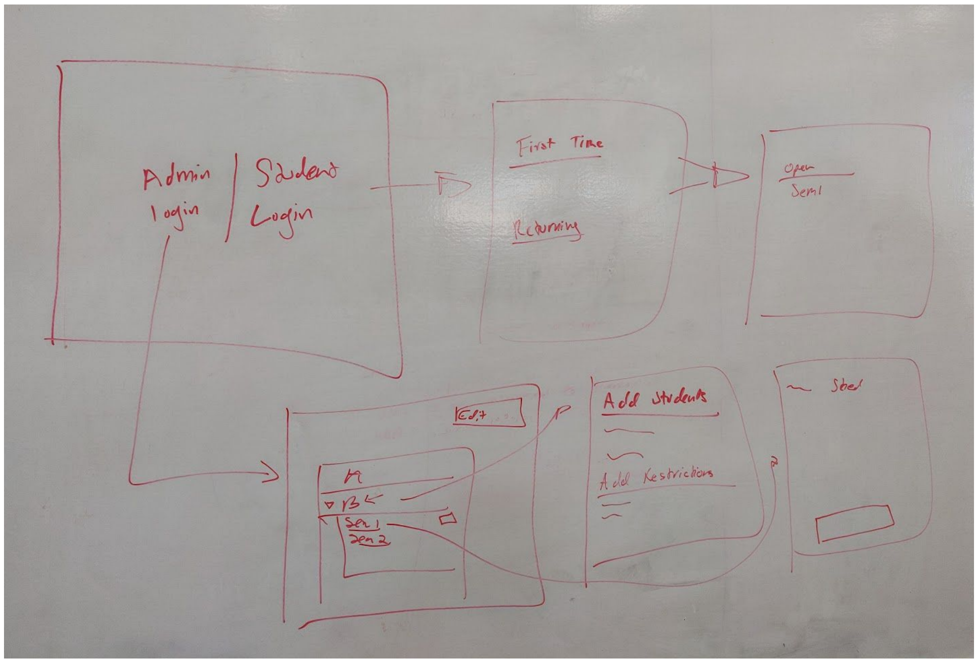
V. Summary for each iteration

Iteration Number	Summary
0	The team met with the customer and collected their requirements. After the meeting user stories were brainstormed and divided among the team members for ownership.
1	On the frontend, a design flow for the user authentication and authorization was performed. A basic html skeleton layout of the Create Schedule and View Schedule page for the students, View Term page for the admin was also designed. On the backend, parsing of class information from the TAMU course website was performed and the first half of the core scheduling algorithm was written.
2	On the frontend, the student user authentication was set up with features like Sign up, Sign in and Sign out. The student View Terms page was modified to only view the current term (as determined by the admin) and the schedules they have created for that term. The admin View Term page was modified to only view the current term (as determined by the admin) and the cohorts included in that term. The New Term page for the admin was designed so that when the administrator feels they are done with a particular semester, they can create a new semester by wiping any data from the past semester. A Manage Cohorts page and Open Term page were also created. On the backend, parsing from the html course page was completed and models were created to store this parsed data. Part of the algorithm was implemented and was tested on a small sample data set. This was essentially the first two steps in the algorithm's pipeline, which only required one more method to be implemented to function properly.
3	On the frontend, a forgot password feature was added to the authentication. User authorization was implemented using Devise gem. It didn't work as intended because authentication was set up in such a way that it created a problem for the Devise gem, so we had to redo the authentication from scratch. On the New Term page the admin can choose a term available to become active, rendering all other terms inactive. The active term name is then displayed at the top of the View Term page for admins. It is also displayed for the students if the current time is in the range of 'open' times. The term table was also updated with an open date and close date column. On the Open Term page, the admin can select a range of times to 'open' this term for students to add their schedules to it and on the student side the controllers were modified to only direct a student to the View Term page if they are currently within the 'open' time range for the active term. If not, they're redirected to the 'closed' page. On the backend, parsing had to be rewritten as the old Compassxe page was retired and replaced with a brand new format. The implementation of the core algorithm was

	completed and some other auxiliary methods were implemented, for example, all possible combinations of schedules were now able to be iterated through a brute force method. A method was implemented that returns an array of students for a given conflict in a set of schedules. The code to translate the data from the format provided in the database to the format required by the algorithm was half-way completed.
4	On the frontend, user authentication including features like Sign up, Sign In, Sign out, Forgot password and authorizations for student and admin were implemented successfully from scratch. No gems were used for any of them. So, upon login, the admins are redirected to the View Terms page for the admin and students are redirected to the View Terms page of the students. Several functionalities were added to the students and admin pages. On the New Term page, an admin could select which term to activate and which cohorts to include in that term. They could also click the “load more terms” button if they do not see the term they wish to activate. On the View Term page, a student could see the active term and a list of schedules they had added for that term. From here, they could delete their existing schedules, add a new one, or click on one of the schedules to view it. On the View Schedule page, a student could see a list of courses they added for a particular schedule and on the Add Schedule page, a student could use the provided drop-downs to choose courses they wanted on this schedule. On the View Students with Schedules page, once an administrator clicks on a cohort from the view term page, they are taken to the View Cohort Semester page, which displays a list of students and if that student had added schedules for that term yet. On the Add Cohort page, an admin could upload an excel file with students' information to create their accounts, which they could later claim in the new user login. On the Manage Cohorts page, the admin could view a list of current cohorts and the students that belong to it. They could also edit or delete students, or delete an entire cohort at once. Finally, on the Manage Administrators page, the admin could view a list of all admins, edit/delete them, or add a new one. On the backend, we got the schedule web scraper fully operational and imported all the classes into our database. Dummy data was created for three students with three schedules, a piece which was used to test the pipeline of the algorithm. There were issues with using Postgres for local development, so we were not able to finish the user story to convert class schedules into matrices, but the user story was completed by the next Monday afternoon.

VI. Customer meetings

1. Meeting 1 - 02/21/2020, 12:00 pm - 1:00 pm, ZACH 425
   In the first customer meeting, all team members and customers became acquainted. The customers detailed their current problem with lecture scheduling to the team, and some preliminary UI sketches and user stories were created.



2. Meeting 2 - 03/23/2020, 11:30 am - 12:30 pm, Zoom meeting
   Due to spring break and the COVID-19 outbreak, this meeting was postponed from its original date to nearly 1 month after the first customer meeting. In this meeting, some initial UI's were shown to the customers as well as the initial design flow for the login page. The customers asked for some adjustments to the user stories and the UI's - mainly from viewing multiple semesters at a time and keeping a history of past semesters to only viewing one semester at a time and keeping no history of past semesters. They wanted to keep the program simple and stated they do not need any sort of history.

3. Meeting 3 - 04/03/2020, 11:30 am - 12:30 pm, Zoom meeting
   During this meeting, the new UI's and corresponding user stories with only one term on display were shown to the customers. They approved this change, and the rest of the user stories were left the same. At this point, the customers also made it clear they wanted a method to bulk upload cohorts of students, such as via excel sheet, so a new user story was created to fulfill that need. They were shown the working login for students, and

updated on the status of the TAMU Course Search website scraping and the scheduling algorithm. Per the teams request, the customer sent out a survey to their students to collect past semesters schedules to test the algorithm on.

4. Meeting 4 - 04/17/2020, 12:00 pm - 1:00 pm, Zoom meeting
In this meeting, the customers were shown the activate & open term features, which they approved of. The TAMU Course Search website changed between this meeting and the last, so the page scraping algorithm had to be rewritten, and they were updated on this setback. The customers were also shown the working ‘forgot password’ feature, which they approved of as well.

5. Meeting 5 - 05/1/2020, 3:30 pm - 4:30 pm, Zoom meeting
In this final meeting, the customers were shown the cohort management pages, the administrator management pages, the functioning add schedule page drop-downs, and the functioning login for pre-approved users (as uploaded by the admin). They were very pleased with how much progress was made, and their only concern about the application was the progress of the scheduling algorithm. They were provided with an update on the status of this algorithm and were told the team members would continue working on it until the final report was due. We will be giving them another update on the final status of the algorithm in the coming week.

VII. BDD/TDD process

During each iteration we utilize the user stories and UI sketches made with the customer to gain an accurate sense of what is needed. Then, we discuss how to best implement the user story with TDD in mind. We designed several Cucumber and Rspec tests to ensure we were meeting the standards set by our customer.

VIII. Configuration management approach

We used github for project management and Cloud9 as the primary development system. We had divided the work into four parts including login and user interaction on the frontend and getting schedules from TAMU and optimizing student schedules on the backend. Each user had the responsibility of pushing their own code to their own branch and after consulting with each other we merged the branches into master branch. In total we had 17 branches that after finishing each task that branch was merged into the master branch. For the scheduling algorithm, there was one spike branch for testing out algorithm implementations before writing them in the main scheduling branch. The rest of the branches were branched off by the functionality that was being implemented.

IX. Issues with Heroku

The only issues encountered when deploying to heroku were limited. Foremost, we needed to make sure the heroku client was installed and properly set up on the local host. From there, the next step was making sure Postgres was properly installed on the heroku host that we spun up. The only other issue that had to be solved was how to properly push a sub-directory of the git repository to heroku instead of the entire repository. This was solved with the `git subtree push --prefix` command which allows you to specify the sub directory to be used as the root directory when pushing to a remote.

X. Issues with other Tools

We had minor issues setting up Postgres on local machines. On MacOS, there was an issue where a postgres admin was created without a password, and therefore we couldn't perform any actions since no password existed. We had to move to Cloud9 instead of using the local machine for that specific instance. Otherwise, we had to create the proper user accounts for rails, and other than that we did not encounter anything with Cloud9 or Postgres. We had one or two merge conflicts we had to resolve with git but that's to be expected during project work. Nothing out of the ordinary.

XI. Tools/GEMs used

Ruby version: 2.6
Bundler version: 2.1.4
Pg: Gem file for database management
Roo: Gem file for importing .xlsx or .xls or .csv files
Turbolinks: It makes navigating your web application faster
Acts_as_list: It allows the sorting of a list