Final Report

Team Member:
Stephanie Lam
Hannah Ifekoya
Aaron Xu
Neel Singh
Yuvanesh Rajamani
Hao Zhu

Date:
4/25/2025

1. Summary

This project addressed the needs for a more efficient way to assign TAs and graders for classes every semester for the CS department. Since the legacy implementation was simply just a python script, the method requires the admin to input all data manually from student and professor which is time consuming and inflexible.

The APP mainly addresses data collection from student and professor, automatic assignment of these data, reassignment when applicants drop out, blacklist, and UI for add, delete, edit, importing and exporting CSV files. The App is designed with Ruby on Rails, and hosted on Heroku.

2. User Stories

Below is the breakdown of the user stories that were successfully implemented based on sprints with description, assignments and points given. All user stories are implemented based on the sponsor's need and guidelines.

Sprint 1 Stories:

Feature	Story	Story Points	Assigned To
UI overhaul	Unify CSS styling for all pages and add general improvement to be user friendly	3	Hao
Login Filters	Implement view base on roles	3	Aaron
Better Search	Add advanced search options to enable search with multiple input	2	Hao

	fields		
Assignment and Reassignment fix	Work on the Assignment and professor preferences	3	Yuvanesh Stephanie
Automated Email System	Add automated emails that let student navigate to our app to accept or reject positions	5	Neel Yuvanesh Stephanie Hannah

Sprint 2 Stories:

Feature	**Story**	**Story Points**	**Assigned To**
Assign TA to classes automatically	Implement the TA algorithm	3	Hao
Login via NetID/google	Implement the TAMU NetID SSO authentication for users to login to the website/portal	5	Aaron
Reassign TA	Allow the algorithm to rerun after assignments for any dropout/changes	3	Hannah Hao
Manually assign TA	Admin can manually assign	3	Yuvanesh Stephanie
Create Separate Page Views for users	Separate page views for Admin, Applicants and Faculty	3	Stephanie Neel

Sprint 3 Stories:

Feature	**Story**	**Story Points**	**Assigned To**
Import data from CSV to database	Import semester class schedules	3	Hannah Neel
Login via NetID	Implement the TAMU NetID SSO authentication for users to login to the website/portal	5	Aaron
Export data from database to CSV	Export the hiree to HR	3	Hannah Stephanie

Faculty recommendation system	Allow faculty to recommend students for TA/grader roles	3	Neel
Blacklist student	Allow faculty to blacklist students with unethical behavior	3	Stephanie

Sprint 4 Stories:

Feature	**Story**	**Story Points**	**Assigned To**
Creation of website/portal	Set up website/portal framework for faculty and students	4	Hao Stephanie
Login via NetID	Implement the TAMU NetID SSO authentication for users to login to the website/portal	5	Aaron Hannah
Application for students to be TAs or graders	Develop functionality for students to apply to available classes as TAs or graders	3	Yuvanesh
Faculty recommendation system	Allow faculty to recommend students for TA/grader roles	3	Neel

Below is a detailed description of all user stories that were planned and the available lo-fi UI mockups/storyboards. Some that did not get implemented are also included here with explanations.

Feature: Prioritizing PhD students
As the organizer, they want to make sure PhD students are prioritized when assigning TA positions so that all vacant positions are assigned to PhD students first.

Feature: Prioritize PhD student
As an organizer
So that I can assign all vacant position to PhD student first
I want to make sure PhD students are felt prioritized

Status: Not implemented
This feature is not implemented due to the requirement being partially satisfied with the original code base. The original python script already put PHD students at the highest priority but does not guarantee that a PHD student will get the position. Some of the weights were adjusted in the original code, but not comprehensive enough to summarize the work as an entire User story.

Feature: Blacklist student
As the organizer, they want to punish the behaviour of students who promised to be a TA for the position but backed out and changed their minds at the last minute, so that they can secretly penalize the student who performs unethical behavior. The number of points given for this user story was 3 points and this user story is implemented fully.

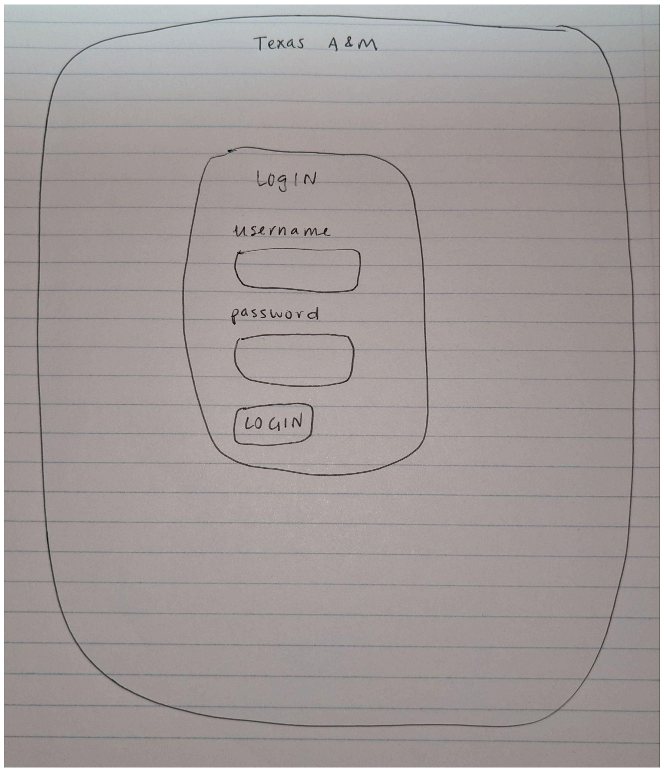
Feature: Blacklist student
As an organizer
So that I can secretly penalize student application
I want to punish behaviour of student promise to TA for a position but change in the last minute

Status: Implemented

Feature: Login via NetID
As a user for the application, they want to be able to track and use the functionality in a secure environment, so that they can login and check any of their progress. The number of points given for this user story was a total of 8 points and this story is implemented fully.

Feature: Login via NetID
As an user
So that I can login to save and to check my progress
I want to track and use the functionality securely

Login via NetID

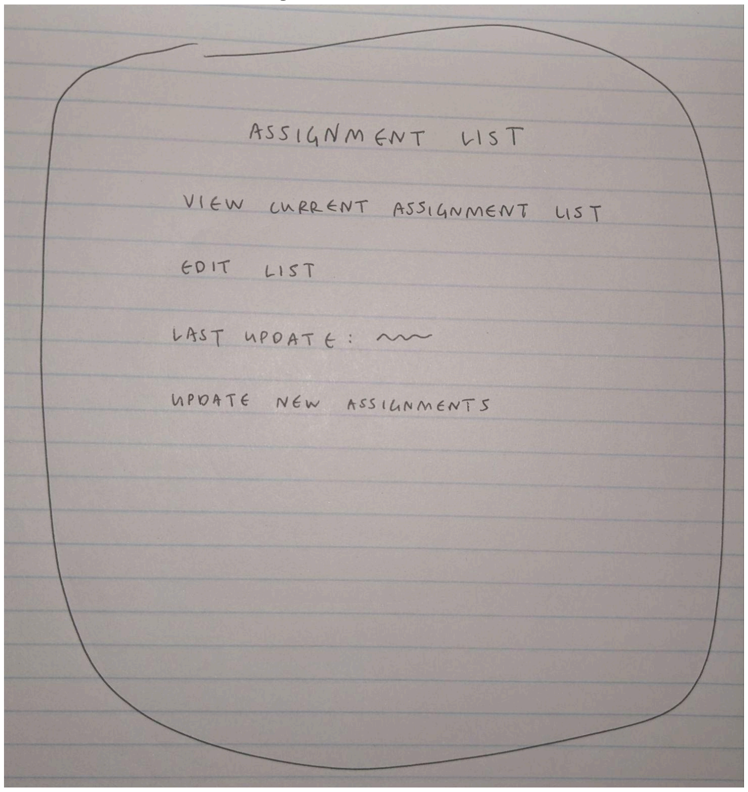


Status: Implemented
The requirement of this user story got changed halfway through the semester due to unresponsiveness of the department IT. Since there is no access to the TAMU NetID API, Google login was used as an alternative.

Feature: Assign TA to classes automatically
An organizer does not have to manually assign TA to class. This process will be done automatically with a press of a button where all the command line will be run in the app in the background allowing. The number of points given for this user story was 3 points and it is fully implemented.

Feature: Assign TA to classes automatically
As an organizer
So that I do not have to manually do it
I want to assign graduate students to classes they would be most helpful in automatically

Assign TA to classes automatically



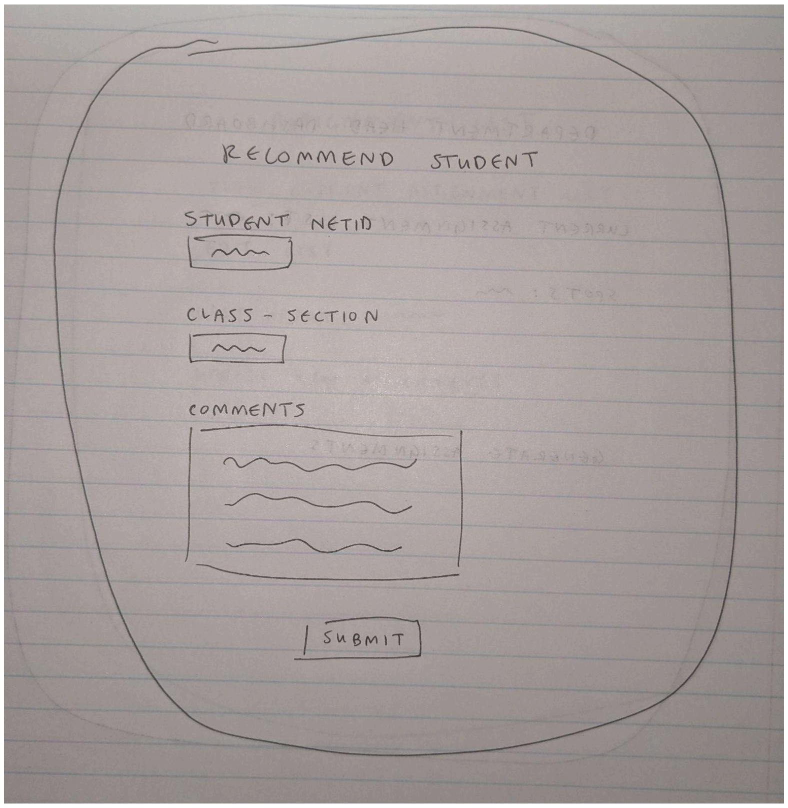
Status: Implemented

Feature: Recommend student
As a faculty, they want to be able to recommend specific students they want to be a TA or grader in their class, so that they can have a higher chance of working with the

student they want to work with. The number of points given for this user story was 3 points and it is fully implemented.

Feature: Recommend student
As a faculty
So that I get the student I recommended as the TA or grader
I want to be able to recommend specific students I want to be a TA or grader in my class

Recommend Student

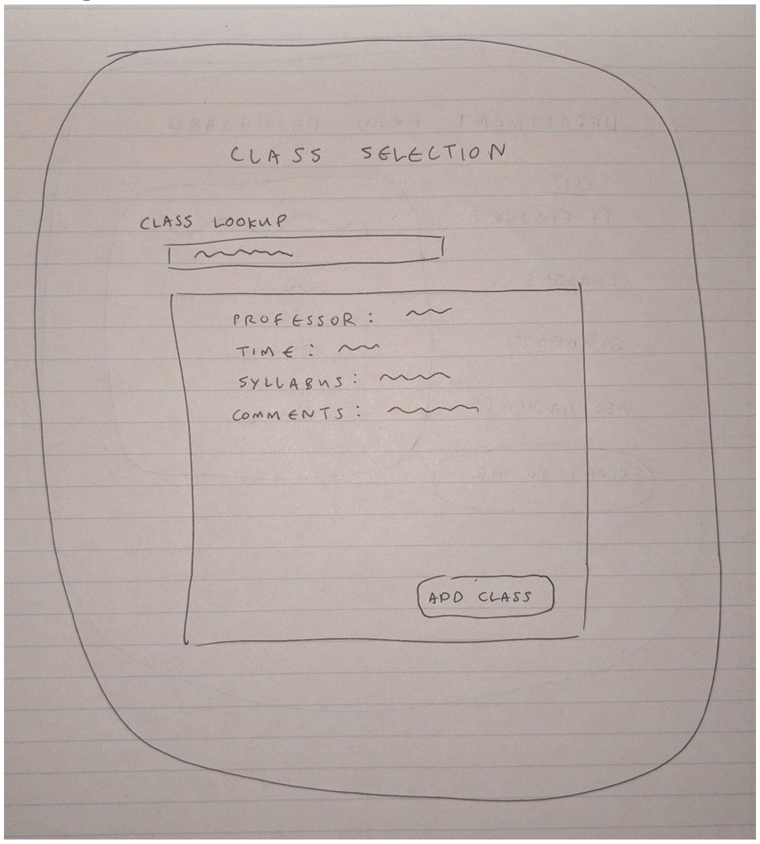


Status: Implemented

Feature: Applying to be TA or grader
As a graduate student, they want to be able to TA or grade for a class that I know the materials being taught, so that they can possibly be a TA or grader for the class they had chosen. The number of points given to this user story was 3 points and it is fully implemented.

Feature: Applying to be TA or grader
As a graduate student
So that I can TA or grade for a class that I had chosen
I want to be able to TA or grade for a class that I know the materials being taught

Applying to be a TA grader



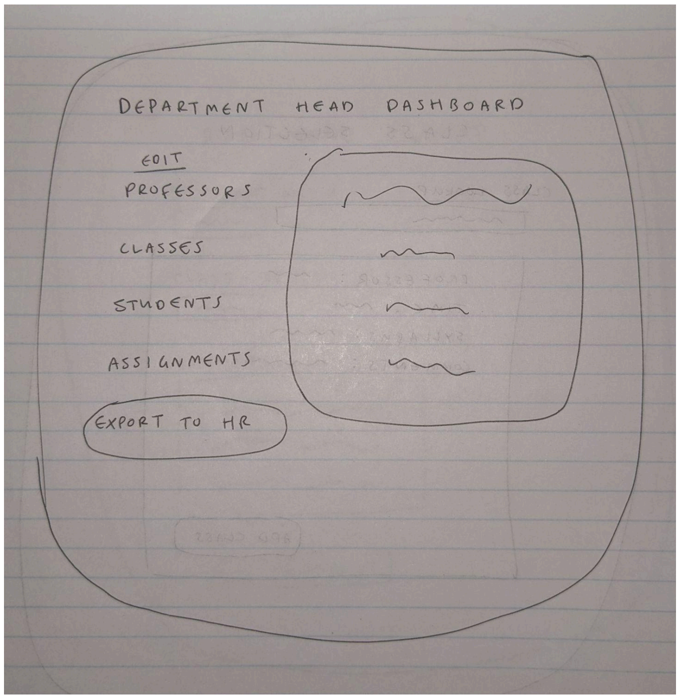
Status: Implemented

Feature: Exporting hirees to HR

As a department head, they want to be able to send all the potential hires to HR, so that they do not have to manually send each potential hire to HR. The number of points given to this user story was 3 points and it was fully implemented.

Feature: Exporting hirees to HR
As a department head
So that I do not have to manually send each potential hiree to HR
I want to be able to send it all at once to HR

Exporting hirees to HR



Status: Implemented

Feature: Import time schedule
As a faculty member, they want to be able to assign positions based on the student's availability, so that they can compare the student's schedule they want to recommend to the open positions.

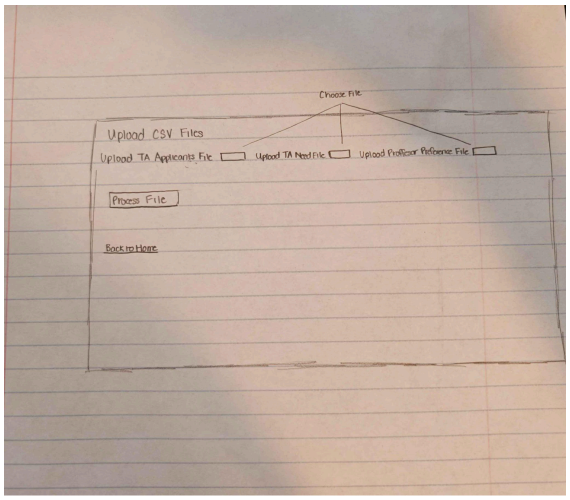
Feature: Import time schedule
As a faculty
So that I can compare my student's schedule to the open positions
I want to be able to assign position base on the student's availability

Status: Not Implemented
This feature is not implemented due to the same reason where department IT is unresponsive. However, this story is not on the priority list of the sponsor and was decided to be abandoned later.

Feature: Reassign TA
As an organizer, they want to re-assign graduate students to classes automatically after the first round of applications are accepted/unaccepted, so that the organizer does not have to manually do it. The number of points that was given is a total of 6 points and it is fully implemented.

Feature: Reassign TA
As an organizer
So that I do not have to manually do it
I want to re-assign graduate students to classes automatically after the first round of applications are accepted/unaccepted

Reassign TA



Status: Implemented

Feature: Manually assign TA

As an organizer, they want to be able to manually assign a TA before and after the algorithm, so that they can have ultimate control over the TA assignments. The number of points given to this user story was 3 points and it is fully implemented.

Feature: Manually assign TA
As an organizer
So that I have ultimate control
I want to be able to manually assign TA before and after the Algorithm

Status: Implemented

Feature: Send out emails

As an organizer, they want to automatically send out emails to applicants about their application and to professors about the recommendation system, so that they do not have to manually do it. The number of points given to this user story was 5 points and it is fully implemented.

Feature: Send out email
As an organizer
So that I do not have to manually do it
I want to automatically send out emails to applicants about their application and to teachers about the recommendation system.

Status: Implemented

Feature: Acceptance or rejections

As an organizer, they want to be able to send offers to applicants and automatically process acceptance and rejection to positions, so that they do not have to manually do it. The number of points given to this user story was

Feature: Acceptance or rejections
As an organizer
So that I do not have to manually do it
I want to be able to send offers to applicants and automatically process acceptance and rejections to positions

Status: Implemented

3. Legacy Codes

Since the client requires a streamlined system for assigning TAs and graders each semester, addressing key needs such as easier student applications, faculty recommendations, offer tracking and automated data export for HR. The Legacy system consists of a standalone Python script using simple libraries Numpy, Scipy, and Pandas to parse CSV files and assign applicants based on calculated scores. However, the current system lacks flexibility for mid-process updates, mandatory PhD student hiring, and user-friendly interaction. Our goal is to improve this logic, add missing functionalities like adaptive assignments and priority handling for PhD students, and integrate these features into a web-based Ruby on Rails application with an intuitive UI.

To achieve this, we first focused on fully understanding the legacy Python code by reverse-engineering its data flow, scoring algorithm, and assignment process. We then began refactoring the system to support dynamic updates, allowing individual additions, removals, or reassignment of TAs without rerunning the entire algorithm. Additional functionality, such as prioritizing PhD applicants and enabling one-click HR exports, is being added to meet client requirements. Our approach involves stabilizing and improving the Python backend first, while planning for integration or potential migration of the logic into the Rails application to ensure seamless, user-friendly operation.

4. 5. 6. Team Role Accomplishment and Contribution

The project consisted of four sprints, each with its own distinct goals. Roles rotated each sprint to give every team member the opportunity to develop, and also gain experience as a product owner and scrum master. We also utilized pair programming throughout the project to improve collaboration and boost efficiency.

Sprint 1:

- The objective of Sprint 1 was to establish the basic components of the TA/grader assignment system into a deployable website/portal focusing on key functionalities. We aimed to get these features completed by Sprint 1:
    1. Develop the core website/portal framework for students and faculty to use
    2. Enable students to apply for TA/grader positions through an application system

Sprint Achievements Overview:

- Completed setting up the project framework, implementing the basic UI and CSS
- Developed the login page for users without the TAMU API authentication

- Created the application portal for students to fill out to apply for a TA or grader position

Team Role:

- Scrum Master: Stephanie Lam
- Product Owner: Hannah Ifekoya
- Developer 1: Aaron Xu
- Developer 2: Neel Singh
- Developer 3: Yuvanesh Rajamani
- Developer 4: Hao Zhu

Sprint 1 Table:

Scrum Master	Stephanie Lam	3 pts	**Pair Programmer** Worked on setting up project framework and implementing basic UI and CSS
Product Owner	Hannah Ifekoya	3 pts	**Pair Programmer** Worked on Integrating TAMU API authentication and developing login page for users
Developer 1	Neel Singh	3 pts	**Main Programmer** Worked on developing interface for faculty to recommend students and allow faculty to lookup students and display their info
Developer 2	Yuvanesh Rajamani	3 pts	**Main Programmer** Created the application portal
Developer 3	Aaron Xu	3 pts	**Main Programmer** Worked on Integrating TAMU API authentication and developing login page for users
Developer 4	Hao Zhu	3 pts	**Main Programmer** Worked on setting up project framework and implementing basic UI and CSS

Sprint 2:

Sprint Goal:
- The objective of Sprint 2 was to establish the database, allowing for the client to easily import and export the information needed:
- Develop the core website/portal framework for students and faculty to use
- Enable students to apply for TA/grader positions through an application system
- Allowing for admin to upload class data

Sprint Achievements Overview:
- Completed importing data from CSV to database
- Completed exporting data from database to CSV
- Created the faculty recommendation system

Team Role:
- Scrum Master: Hao Zhu
- Product Owner: Yuvanesh Rajamani
- Developer 1: Neel Singh
- Developer 2: Aaron Xu
- Developer 3: Stephanie Lam
- Developer 4: Hannah Ifekoya

Sprint 2 Table:

Scrum Master	Hao Zhu	3 pts	**Pair Programmer** -Worked on Blacklist
Product Owner	Yuvanesh Rajamani	3 pts	**Pair Programmer** -worked on CSV input
Developer 1	Neel Singh	3 pts	**Main Programmer** -worked on Recommendation system
Developer 2	Aaron Xu	3 pts	**Main Programmer** -worked on Tamu Login
Developer 3	Stephanie Lam	3 pts	**Main Programmer** -worked on Blacklist
Developer 4	Hannah Ifekoya	3 pts	**Main Programmer** -worked on CSV input

Sprint 3:

Sprint Goal:
- The objective of Sprint 3 was to implement the core hiring and TA/grader assignment features along with the login feature in a deployable website/portal:
    1. Assign TA to classes automatically
    2. Allow the algorithm to rerun after assignments for any dropouts/changes
    3. Allow the admin to manually assign TA/graders

4. Separate pages for Admin, Applicants, and Faculty

Sprint Achievements Overview:
- Completed implementation of the TA/grader algorithm including rerunning the algorithm for TA/grader assignments if needed for dropouts/changes
- Completed manual assignment functionality for the admin
- Created the separate page views for admin, applicants, and faculty

Team Role:
- Scrum Master: Aaron Xu
- Product Owner: Neel Singh
- Developer 1: Hao Zhu
- Developer 2: Yuvanesh Rajamani
- Developer 3: Stephanie Lam
- Developer 4: Hannah Ifekoya

Sprint 3 Table:

Scrum Master	Aaron Xu	3 pts	**Pair Programmer** -worked on login
Product Owner	Neel Singh	3 pts	**Pair Programmer** -worked on creating separate page views for users
Developer 1	Yuvanesh Rajamani	3 pts	**Main Programmer** -worked on manually assigning TA
Developer 2	Hao Zhu	3 pts	**Main Programmer** -worked on reassigning the TA algorithm
Developer 3	Stephanie Lam	3 pts	**Main Programmer** -worked on creating separate page views for users
Developer 4	Hannah Ifekoya	3 pts	**Main Programmer** -worked on reassigning the TA algorithm

Sprint 4:

Sprint Goal:
- The objective of Sprint 4 was to add additions to the app that would meet the client's standards and finish the implementation of the login feature
  1. Add advanced search options to enable search with multiple input fields

2. Add automated emails that let students navigate to the application to accept or reject positions
3. Work on assignment and reassignment fixes and professor preferences
4. Implement view based on roles of either student, faculty, or admin
5. Unify CSS styling for all pages and add general improvements to be more user friendly

Sprint Achievements Overview:

- Completed addition of advanced search options to enable search with multiple input fields
- Completed addition of automated emails that lets students navigate to the application to accept or reject positions
- Completed the fixing of assignment and reassignment and added professor preferences
- Completed the unification of CSS styling for all pages and added general improvements to be more user friendly
- Completed the login feature and the views are now based on either student, faculty, or admin

Team Role:

- Scrum Master: Hannah Ifekoya
- Product Owner: Stephanie Lam
- Developer 1: Neel Singh
- Developer 2: Hao Zhu
- Developer 3: Aaron Xu
- Developer 4: Yuvanesh Rajamani

Sprint 4 Table:

Scrum Master	Hannah Ifekoya	3 pts	**Pair Programmer** -worked on adding automating emails that lets students navigate to our app to accept or reject positions
Product Owner	Stephanie Lam	3 pts	**Pair Programmer** -worked on adding automating emails that lets students navigate to our app to accept or reject positions
Developer 1	Neel Singh	3 pts	**Main Programmer** -worked on adding automating emails that lets students navigate to our app to accept or reject positions
Developer 2	Hao Zhu	3 pts	**Main Programmer** -worked on UI overhaul and added advanced search options

Developer 3	Aaron Xu	3 pts	Main Programmer -worked on implementing view based on role
Developer 4	Yuvanesh Rajamani	3 pts	Main Programmer -worked on adding automating emails that lets students navigate to our app to accept or reject positions

7. Customer Meetings

Customer Meeting Dates: 1/30/2025, 2/19/2025, 3/6/2025, 3/27/2025

1/30/2025: Team introduction with the customer, customer talked about what the client wants from the project and the problems he had with the legacy project especially with not being able to automatically assign TA/Grader position to the applicants, it was really hard for him to do the reassignment if someone rejects the offer or dropped in between. Received preferences from the client for the assignment such as PHD students are preferred by default, applicant should have a good GPA, if the student has taken the class before it is preferred, recommendation by the professor, if the student is an international student they should have at least level one english. These preferences helped us navigate the assignment process.

2/19/2025: Reconfirmation with the client of what he wanted and gave him a status update with what has been completed. Also told the client how the IT department was not responding or being helpful about the integration of TAMU SSO login feature and told him we would keep him updated on the status. The team and the client discussed what the basic structure looks like after sprint 1 and had some feedback over there regarding the structure and appearance of the website.

3/6/2025: Reconfirmation with the client of the current status of the application, telling him that the login issue was still persisting due to the IT Department. The client wanted a feature that enables them to send the selected applicant's data directly to the TAMU HR department for further on boarding process.

3/27/2025: The team showed the client the current application status and received feedback from the client on what he wanted. The client gave suggestions including wanting a form drop-down for subject fields, confirmation for full deletions, some styling changes, and for the team to keep in mind to make it as fool-proof as possible for the users.

8. BDD and TDD

Our TDD and BDD were implemented using Rspec and Cucumber respectively. Our main implementations of these tests were to develop the controller and then test it. We had some issues where tests would work fine but we would have to adjust the controller and model which would completely mess the tests up. Our app also has google authentication built in which made testing difficult as many of our models were associated with a logged in user or the controller required a logged in user. BDD testing of our javascript also proved to be difficult as javascript can be finicky and vary from browser to browser.

9. Configuration Management

For configuration management, the project followed a Git based branching strategy and a protected main branch. Each new feature, bug fix, or configuration change was developed on a separate branch, which was merged into main through pull requests after code review and successful test runs. We also made use of the .env file managed by dorenv-rails. The production configuration is handled through server-managed env variables.

During the project, we performed a few spikes over the Google Oauth integration and Ransack search. The project maintained around 4 active branches at a time for the different user stories. However, we do have 35 branches for testing and merging purposes. We have 4 releases managed by tagging commits on the main branch.

10. Heroku

This project was chosen to be deployed on Heroku as we felt that it was the most suitable for a Ruby on Rails application. Rails works seamlessly with PostgreSQL which is mainly used on Heroku. Heroku also has a ton of supported tools for Rails so it was the most optimal choice especially in a situation where our client might not be well versed in DevOps. A lot of the difficulties that arose with Heroku came from simply not knowing how to use it at the start, but it's very easy to get up to speed as Heroku is very simple and follows a lot of rails conventions. The main issue we were facing in our deployment was resolving an issue where our python algorithm was not working due to python not being installed on Heroku. This was resolved after adding the python buildpack for heroku. Deployment was simple after that as all you needed to do was push the main branch to Heroku and run the rails migrations in Heroku.

11. Tools Description

This project used several key tools and libraries to support development, testing, and security. GitHub handles version control and collaboration through pull requests and workflows, though it requires disciplined use to avoid conflicts. SimpleCov provided test coverage reports, helping identify untested code areas, while CodeClimate offered maintainability insights, though it may produce occasional false positives.

For security, Brakeman was used to scan for Rails-specific vulnerabilities, providing fast static analysis, though manual review is still needed for full coverage. RuboCop, with the rubocop-rails-omakase configuration, enforced a consistent code style, though its strict defaults required tuning for team preferences.

Testing relied on RSpec, supported by Cucumber, Capybara, and Selenium for behavior-driven and system tests. While this stack offered thorough coverage, system tests could be slow or flaky without proper care. dotenv-rails managed environment variables securely in development, but production variables required external management.

Authentication was handled using Devise and OmniAuth Google OAuth2, simplifying secure login via Google but requiring careful credential setup. For pagination and search, Kaminari and Ransack were used, offering flexible features with some limitations in advanced or large-scale use cases. These tools together ensured the project remained secure, maintainable, and well-tested.

12. Code Pushed

All of our code is pushed to github and deployed on Heroku.

13. Deployment

Our repo contains everything that will be needed to run this app locally and on Heroku. The README contains detailed instructions on what dependencies are needed and how to deploy. The Gemfile contains all of the required gems for this rails project to work.

For Local setup:

1. Ensure Ruby and Python are installed
    a. Ruby version
        i. ruby "3.4.1" or higher
    b. Python Version and libraries
        i. Python 3.9
        ii. rich==13.3.2
        iii. scipy==1.7.1
        iv. pandas~=1.5.3
        v. numpy~=1.22.4
2. Configuration
    a. Run this to install all the right ruby dependencies
        i. bundle install
3. App Setup instructions:
    a. Create a google app
        i. https://console.cloud.google.com/
        ii. Create a new project
        iii. go to OAuth Consent Screen and fill it out
        iv. Go to Create OAuth client ID
        v. Fill out the app name information
        vi. Add http://127.0.0.1:3000/auth/google_oauth2/callback to the redirect uri and Create
        vii. Save Client id and Client secret credentials
    b. Create .env file in root directory of project
        i. add previously saved client id to GOOGLE_CLIENT_ID="CREDENTIALS”
        ii. add previously saved client secret GOOGLE_CLIENT_SECRET="CREDENTIALS"
        iii. add admin emails to .env ADMIN_EMAILS="admin@email1.com,Admin@email2.com"
4. Run rails migrations
    a. rails db:migrate
5. Run app locally
    a. rails server

For Heroku Deployment:

- Make sure to first do the Local setup as there important steps that must be done before deploying

1. Ensure Heroku CLI is installed
    a. https://devcenter.heroku.com/articles/heroku-cli
    b. Login to heroku
        i. heroku login
    c. Create app in Heroku
        i. heroku create your-app-name
    d. Push Project to Heroku
        i. git push heroku main
    e. Go to your app console on your heroku dashboard and add Heroku Postgres Essential 0 to add-ons. This is the lowest tier(should be enough for hundreds of applicants), Change based on your needs.
    f. Add env var to heroku
        i. heroku config:set GOOGLE_CLIENT_ID="google client id" GOOGLE_CLIENT_SECRET="google client secret"
        ii. heroku config:set ADMIN_EMAILS="admin@email1.com,Admin@email2.com"
    g. Add Python buildpack to heroku
        i. heroku buildpacks:add heroku/python
    h. Run migrations on Heroku
        i. heroku run rails db:migrate
    i. Go to OAuth portal and update the redirect uri
        i. Fill out with this https://(YOUR APP URL)/auth/google_oauth2/callback

14. Links

- **GitHub Repository:** **https://github.com/stephanielam3211/CSCE606_Project?tab=readme-ov-file**
- **Live Project:** **https://tamu-csce-ta-portal-0646b9b7b2a9.herokuapp.com/**

15. Presentation Links

- **Presentation and Demo:** **https://youtu.be/XvABZu6AQ_o**