Capstone Matching Final Report

I. Team Roles

1. Zachary Laguna (Scrum Master)
2. Hunter Zacha (Product Owner)
3. Aakashdeep Sil
4. Yash Patil
5. Shripad Kulkarni
6. Sathvik Kote Revanasiddappa

II. Project Summary

The main goal of our project was to consolidate the Industry Capstone Matching process into a single application. The main parts of the process, student information submission, student matching with projects, and professor project assignment, used to be split up into multiple pieces or done manually. Our application unified all process aspects and reduced the manual work required for professors. Currently, students can submit their personal information, professors can view that information and scores generated using their "match data," and the admin can manage all parts of the process. The current implementation consists of 3 main parts: Admin View, Professor View, and Student View/Student Form. Each of the different views represents the project's primary stakeholders and has other functionalities associated with them.

Role/View Descriptions:
The Student Form consists of a form to enter the information needed for the Industry Capstone Matching process: UIN, gender, course, work authorization, willingness to sign contracts, nationality, and ethnicity. Students can also submit their resumes and rank the projects from the current semester based on how much they prefer them. Some information is prefilled from Google OAuth data, but the rest is manually entered or done through dropdowns. The administrator can lock the form based on the dates they set for opening and closing. Once students submit the form, they are scored based on when they submit it relative to when the form opened, how well their resume matches the project description (if they submitted a resume), and their preference for the project (1 being the highest).

The Professor View allows professors to view students' results for a given project (they can also add a filter to filter for a specific course) and download those results to a CSV. They can also view details of all the projects and filter by the semester, and they can submit their preferences for this semester's projects.

The Admin View allows admins to do everything a professor can do AND manage professors and admins (approve professors who submitted the registration form, promote professors to admin, delete professors, or add a new professor manually). They can view all submitted information from the student form and delete students. They can manage projects (add new projects, delete projects, edit the projects, and add restrictions/preferences to the projects), change the weightings of the different parts of the total score, manage courses (add course, delete course, and assign a course to a professor), view the results of the professor preferences form and give a project to a course/professor, manage the student form (set open/close dates and change min/max number of projects students have to rank), and create a dump of the database.

To see a detailed list of project requirements, kindly refer to our Scope Agreement document that was drafted as a collaboration between our team and our client, wherein a detailed list of requirements has been delineated: https://drive.google.com/file/d/1pcWTJtTVr_ZuIECvecUY5uqiPzbjpuwR/view?usp=sharing

III. User Stories

Iteration	Story Name	Assignee	Points	Description	Other Notes
1	BE: Database Initialization and Planning	Zachary	3	Create initial DB schema given project constraints	
1	BE: Store Project Information from Form	Sathvik	3	Add project info using filled in fields	
1	BE: Store Professor Information from Form	Hunter	3	Add new professor using filled in fields	
1	BE: Store Student	Shripad	3	Add new student using filled in	

	Information			fields	
1	FE: Create a Home Page	Aakash	3	Create landing page with redirects to login and prof registration	
1	FE: Professor Registration Form	Hunter	3	Create page for profs to register	
1	FE: Form to add projects	Sathvik	3	Create page for profs/admin to add new projects	
1	FE: Create a student form	Shripad	3	Create page for students to submit info	
1	FE: Create OAuth Page	Aakash	3	Add redirect page to additional login page	This was later refactored to be a single login button on the home page
1	FE: Professor Landing Page	Yash	3	Create a landing page to navigate to professor functionalities	
1	FE: Create Dark Mode	Yash	3	Create a dark mode for the entire application	This didn't function as intended, so we ended up scrapping it
1	Create Postgres Instance	Zachary	3	Create instance on AWS using the new schema	
2	Draft scope agreement	All	3 (x 6) = 18	Draft agreement with customer based on exact customer wants and needs for each type of user	
2	FE: Admin Landing	Hunter Zacha	1	Create a landing page to navigate to admin functionalities	
2	BE: Admin Landing	Hunter Zacha	1	Navigate to existing admin functionality routes	
2	FE: Manage Profs (CRUD on profs and admins)	Hunter Zacha	3	Make a page where admin can see all professors in a table, the courses they teach, etc. and perform CRUD operations	
2	BE: Manage Profs (CRUD on profs and admins)	Hunter Zacha	3	Handle CRUD operations - admin deletes, updates whether or not profs are approved/admin, and admin	

				can add a new prof	
2	BE: Implement Google OAuth	Aakashdeep Sil	5	Add Google OAuth to the login so a new user is created depending on which login they choose	This was later changed to a single login where only a new student is created and professors have to register or be created by admin
2	FE: Update Student Form	Shripad Kulkarni	3	Add additional fields to student form based on what client wanted	These were later turned into dropdowns during refactoring to minimize error
2	BE: Update Student Form	Shripad Kulkarni	2	Change model and handle additional info	Additional info was contract sign, nationality, ethnicity, etc.
2	BE: Professors/Admin can view projects	Sathvik Kote Revanasiddappa	4	Add a page for professors/admins to view project and filter by semester	
2	FE: Professors/Admin can view projects	Sathvik Kote Revanasiddappa	2	Add a page for professors/admins to view project and filter by semester	
2	Create migration of database for development	Zachary Laguna	3	Create a migration of new updated database to match the scope agreement	Introduction of scope agreement caused us to rework existing DB
2	FE: Upload resume	Zachary Laguna	3	Start functionality to allow users to upload resume	
2	BE: Upload resume and generate raw score based on that	Zachary Laguna	3	Start functionality to allow users to upload resume	
2	FE: Professors add sections	Yash Patil	3	Allow professors to add sections to their list of classes	This feature was later scrapped - instead only admin can assign professors to courses, unique combo of course #, section #, semester
2	BE: Professors add sections	Yash Patil	3	Allow professors to add sections to their list of classes	

3	FE: Admin can edit and delete projects	Sathvik Kote Revanasiddappa	3	Add edit and delete operations to projects (CRUD projects is complete)	
3	BE: Admin can edit and delete projects	Sathvik Kote Revanasiddappa	4	Add edit and delete operations to projects (CRUD projects is complete)	
3	FE: Admin can change weights of scoring factors	Hunter Zacha	2	Allow admin to change the weights of the factors used to score students matches with the project (time submitted, resume, preference)	
3	BE: Admin can change weights of scoring factors	Hunter Zacha	3	Allow admin to change the weights of the factors used to score students matches with the project (time submitted, resume, preference)	
3	BE: OAuth redirect to Admin, Student, and Prof Landing Page (depending on role in DB)	Aakashdeep Sil	2	Make it so Google OAuth redirects a user based on the role attributed to them in the DB	
3	FE: Admin can lock student form	Aakashdeep Sil	3	Make a lock/unlock button to close and open the student form and not allow students to view the form unless it is open	We instead changed it to allow admin to set the open and close dates so they don't have to do this manually
3	BE: Admin can lock student form	Aakashdeep Sil	4	Make a lock/unlock button to close and open the student form and not allow students to view the form unless it is open	
3	Table for Viewing and Ranking Projects	Shripad Kulkarni	7	Add a table for the student form to display project info and add fields to have students rank them	This was later refactored - Hunter made it into a table where you can see all project info, links, and rank them instead of ranking/viewing them 1 at a time
3	Resume file size and score	Zachary Laguna	4	Fix resume cookie overflow issue and generate a score for the resume	We ended up not using the fix for cookie overflow because we didn't want to store the

					resume files in the DB, instead did this client side
3	FE: Profs can mark their preferred projects for admin to see	Yash Patil	3	Make page for profs to mark projects that they want for their courses	This was later changed completely in iteration 5 - made it more similar to the refactored student form
3	BE: Profs can mark their preferred projects for admin to see	Yash Patil	3	Make page for profs to mark projects that they want for their courses	
4	Refactor Prof Registration, Change Weights, and Prof Management Pages	Hunter Zacha	3	Refactor pages created	
4	Create Student Management Page (delete and filter students based on course)	Hunter Zacha	2	Create a page to view the information students submit to the form, filter the students based on the course they are in, and delete students (update not needed)	
4	Refactor Oauth	Aakashdeep Sil	2	Fix Oauth to work with new DB	
4	Create config page (min/max projects and open and close date)	Aakashdeep Sil	2	Create a page to manage configuration - where admin can set the dates when the form opens and closes	Didn't fully complete implementation of min/max because student form was still being worked on
4	Refactor Projects Page (add, remove, view/filter)	Sathvik Kote Revanasiddappa	3	Refactor all parts of the projects page	
4	Add Sponsor Restrictions and Preferences to Projects Page	Sathvik Kote Revanasiddappa	3	Add aspect to projects page where restrictions can be assigned and preferences can also be assigned and given a bonus	
4	Refactor Student Form	Shripad Kulkarni	3	Refactor student form to work with new DB	
4	Add course selection to student form	Shripad Kulkarni	2	Add an option based on the existing courses to allow	

				students to select the course they are in	
4	Parse resume files on client side	Zachary Laguna	3	No longer need to store resumes, instead get the text client side using parse resume button	
4	Refactor resume database usage	Zachary Laguna	2	Added DB field to store parsed resume in student	
4	Refactor Professor Preference Page	Yash Patil	4	Refactor page for profs to submit preference	
5	Finalize student form	Hunter Zacha	3	Completely redo project viewing - show all projects and allow students to rank based on that. For course selector, only allow them to select courses from the current semester. Add dropdowns or radio buttons for different fields. Integrate form open/close and min/max number of projects	
5	Prefill student info into form	Hunter Zacha	1	If student already submitted the form, have their info prefilled	
5	Generate Raw Scores when form is submitted	Hunter Zacha + Zachary Laguna	2 (each) = 4	Hunter- generate raw score based on time submitted and preferences Zachary - generate raw score based on resume (integrate resume with student form final)	
5	Update config page to include min/max number updates	Aakashdeep Sil	5	Add the min/max field to the config page to allow admin to set the min/max number of projects	
5	Add dropdowns for restrictions and preferences	Sathvik Kote Revanasidda ppa	1	Change restriction and preferences to dropdowns so values have to match what is on student form	Reused dropdowns from student form and added them to existing preferences and restrictions
5	Generate results page	Sathvik Kote Revanasidda ppa	4	Make a page for admin to view student scores and download them as a CSV	
5	Manage courses page	Shripad	3	Make a page to add new	This is the updated

		Kulkarni		courses and assign/unassign a course to a professor	"sections" page since sections do not exist
5	View professor preferences page	Shripad Kulkarni	3	Make a page for admin to view the preferences of professors that submit the prof preference form	
5	Integrate resume upload w/ student form	Zachary Laguna	4	Move resume upload from dev test and integrate it with student form	
5	Manage database (DB dump button)	Yash Patil	4	Dump all contents of current database into SQL file so admin can use to overwrite current database if anything goes wrong	
5	Add role authentication to all pages	Yash Patil	3	Check the user's role and make sure that aligns with the page that they are on (non admins can't access admin pages)	

IV. Upcoming Stories/ Future Changes & Ideas

see meeting details from 11/27/2023 for explanation

Some of these will be started/completed by the current team after 12/1/2023 due to a customer request for a "beta" meeting on 12/6/2023 - ones with priority values were requested by the customer specifically on 11/27/2023 meeting

Story Name	**Priority (1 = most important)**	**Points**	**Description**	**Other Notes**
Variable restrictions and sponsor preferences	1	2	When they are filling out the information to add a project, the admin should be able to add > 2 sponsor restrictions and preferences.	This feature is already part of the "Edit" functionality; the customer requested that it be part of the "New Project" functionality as well.
Start of Semesters	1	1	Allow admin to change the start/end dates of each semester rather than being hardcoded.	Currently, Fall is set to start in August, Spring is set to start in January, and Summer is set to

				start in late May.
Change CSV Match Report to include more fields	1	3	On the CSV view results report, add a column for the professor name, course, each preference + restriction on top of the current fields	Currently, the CSV report has the student's name, individual scores, and total scores
Change View Results page to display more fields	1	3	Add columns for Professor Name and Course	Also, change column names to specify Sponsor Preference/Restriction rather than just Preference/Restriction
Create a CSV report for processor preferences and assignments	2	4	For each project, dump the professors that prefer the project, what they ranked it, their courses, and whether or not they are assigned the project for that course	
Change UI to have more contrast/distinction	3	3	Certain parts of UI don't have enough contrast, so it makes it hard to distinguish separation (notably on the cards displaying project info) and color code buttons to match with functionality instead of having all/most maroon	
Implement Dark Mode		2	Give an option on each page to change to dark mode and have it persist across all pages	This story has been started, but has issues so it is not in the final product. The team just didn't get around to finishing it.
Unify the UI		3	Certain UI elements differ across pages, specifically on the admin view - change the design on certain pages to match others.	Notably: table on score weights, table on managing courses, table on viewing professor project preferences
Professor course assignment		3	Allow professors to assign themselves courses that don't have professors assigned yet or unassign themselves a section.	This will likely be useful for the admin. If the professor does not use the professor registration form, the admin will not have to manually assign the professor a course and if a professor is not teaching a certain section

				anymore they can unassign themselves from the section rather than having the admin do it.

V. Iteration Descriptions

For each scrum iteration, summarize what was accomplished and points completed.

Iteration	Summary
0	No actual features were developed during this iteration. This iteration was spent planning, we developed our initial database diagram and came up with our initial set of user stories (many of which were later changed or removed based on feedback the customer gave). We came up with some small mockups that we used as the basis for our UI design going forward. We also deployed our application to AWS even though there wasn't any content on it.
1	This is the iteration where we started development, it mainly consisted of smaller, independent features such as professor registration, adding project information, beginning the student form, and the main landing page and professor landing page. We also added our initial schema to the application, so the models used during this iteration reflected the structure in our database.
2	Iteration 2 was one of our larger and more important iterations. One of the biggest parts of this iteration was drafting a scope agreement alongside the client - our team got together multiple times outside of scrum meetings to discuss what should/should not be included on the scope agreement. The main purpose of this was to prevent scope creep as the client suggested many features, but we wanted to narrow them down to the most important ones. Outside of the scope agreement, we implemented functionality to upload the resume and generate a score based on TF-IDF, view/filter projects, Google Oauth (although it didn't function in the deployed application because we didn't have a valid URL for our application), managing the professors, and changes to the student form.
3	Iteration 3 was a smaller iteration in terms of the breadth of features implemented. We started to refactor our code during this iteration because we had updated the UML to reflect the finalized scope agreement. Although some important features were implemented such as editing and deleting projects, changing the weights of the factors used to score projects, locking and unlocking the student form, creating a table for viewing and ranking the projects for the student form, and allowing professors to submit projects they prefer.
4	Iteration 4 was our largest iteration because it was almost 100% refactoring. This was a very difficult and long process for the team because we had already developed so much of the application by this point, but it needed to be done to integrate the different parts of the project. Before this, most of our project was disjoint where changes made in one area might not be reflected in the area where they should be due to our rapidly changing schema. As a result, our schema was finalized during this iteration and our existing features were refactored to reflect the new schema. We also added a few new features such as adding sponsor restrictions and preferences, creating a config page to adjust the date the form is closed/open and

	the min/max number of projects, parsing the resumes client side, and creating a page to manage the students.
5	Iteration 5 was focused on creating a cohesive product from the several constituent parts. We integrated our components namely the student form with the viewing results page. The integration of our code consisted of making sure that objects are added and removed from the database correctly as well as making sure all features are displayed correctly. During this time, we adhered to our scope agreement and worked to make sure that our items were completed. This iteration was primarily used to add the final features such as finalizing the student form, having student info prefilled in the form, generating raw scores using the student form, a page for admin/professors to view and download student scores, a button to create a dump of the DB to the admin, and adding role authentication on all pages.

VI. Team Member Point Breakdown

Team Member Name	I1	I2	I3	I4	I5	Total	% of Total	Expected "Fair Share"	% of "Fair Share"
Aakashdeep	6	8	9	4	5	32	16.41 %	16.67%	98.46%
Yash	6	9	6	4	7	32	16.41 %	16.67%	98.46%
Shripad	6	8	7	5	6	32	16.41 %	16.67%	98.46%
Hunter	6	11	5	5	6	33	16.92 %	16.67%	101.54%
Zachary	6	12	4	5	6	33	16.92 %	16.67%	101.54%
Sathvik	6	9	7	6	5	33	16.92 %	16.67%	101.54%

VII. Meeting Summaries

Iteration	Meeting Date	Summary
1	9/22/2023 (2:30 pm at PETR 108)	We reviewed our database UML diagram with the customer and received feedback on what data we should store and what we shouldn't, along with consolidating some of the tables (combining the Professor and Admin tables). The customer suggested a new feature where professors could test resume, skill match, and preference score weights to see how student match scores change as the weights change and be able to download CSV reports of these. They also suggested we use a dropdown to select a semester that is automatically generated (based on the date) to prevent user errors.
1	9/29/2023 (2:30 pm at PETR 108)	We walked the client through the parts we had completed for iteration 1. She was satisfied with what we had done overall and liked the simplicity of the UI. We reviewed the database again and were given a few suggestions for revisions, which are now reflected in our UML diagram. The client also suggested we come up with a scope agreement before the next meeting so that we can agree on a scope for the project rather than coming up with new features during each client meeting instead of focusing on the core idea of the project.
2	10/5/2023 at 9:30 am (on Zoom)	During this meeting with the customer we went over our first draft of the scope agreement with the client (that she suggested we make in the meeting before). She wanted us to send the scope agreement to her so she could verify that we are meeting the project requirements and prioritize the tasks on the agreement (from essential to optional).
2	10/13/2023 at 3:00 pm (on Zoom)	This meeting was to finalize the scope agreement. We reviewed how the client wanted the different features outlined in the scope agreement to look/behave to minimize errors when submitting information and make the application more straightforward.
3	10/27/2023 (2:30 pm at PETR 108)	During this meeting, we went over the finalized Scope Agreement that our client sent over and adjusted the wording and some of the features that we agreed to implement so that they could be more precise. We also showed our progress on the project and received feedback from the client: she liked the overall simplicity of the UI and functionality but wanted minor changes to the form (selecting multiple ethnicities, selecting visa status, etc).
4	11/3/2023 (2:30 pm at PETR 108).	During this meeting, we went over our revised schema (now that the scope agreement is completely finalized), and the client looked at the schema to ensure that it stored all the information that they would need (and had the correct relationships set up). We discussed how we would store the restrictions and preferences and how those would factor into a student's score (0 if they fit into a restriction and a bonus amount of points if they fit into a preference). We also discussed future plans for the current iteration (Iteration 4) and the

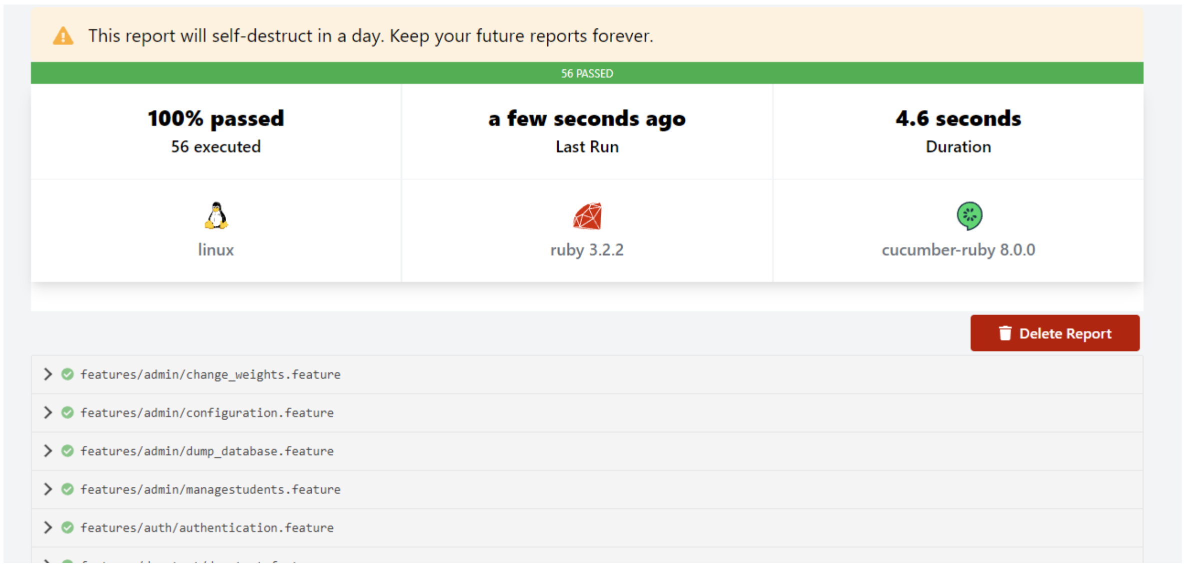
		last iteration. We let her know that our primary goal of this iteration was to refactor and that we would be adding the final features at the start of Iteration 5.
5	11/20/2023 (2:30pm at PETR 108)	During this meeting, we showed the client the features that we had implemented on our deployed application (revised OAuth, complete student form - with resume upload & generating scores, viewing results, etc.) - she approved of the features, and we let her know our plans for the remainder of the project, and we also scheduled an alpha showcase session for 11/29 to do a guided walkthrough of the completed project.
Post Iteration 5	11/27/2023 (2:00pm at PETR 108)	This was the “alpha showcase,” where we did a guided walkthrough with the client of all of the features of the application. We showed her the entire process from start to finish (student submitting the form, prof registering, and being approved). Then, she could view and interact with the data herself as an admin, going through each of the different admin functionalities with our guidance and explaining what everything does. After this, she gave us some suggestions about adjusting the features to better meet her needs. We set up a meeting for 12/6/2023 at her request to show any final adjustments we made based on her feedback - we let her know that some of the suggestions might not get finished this week or by our team because of the timeframe (and exams). Still, we are going to try to get most of her high-priority suggestions done before the 12/6/2023 meeting. Although, our project as it currently exists is fully implemented and in-line with the scope agreement.

VIII. BDD/TDD Process

Our BDD/TDD Process was relatively simple, once we had picked the stories that we wanted to implement for the iteration based on the stories that we had originally came up with or . We would begin by writing the Cucumber scenarios for each feature/story that was being implemented during that iteration - these would serve as a guideline for the features that we implemented and how they would behave given different user inputs (both good and bad paths). Rspec tests were usually written once we had an initial framework written for the controllers to verify that all initial variables that we needed for the feature had the correct values and we continued to write tests based on the different paths that we would have (ie invalid info entered, incomplete info entered, valid info entered, etc.) and wrote tests based on how they should behave. We would then begin implementing the features and continuing until we were able to pass all the tests and we confirmed the feature was complete. If we wanted to add onto an existing feature we would modify the existing tests to reflect how we wanted to change/update the feature and would repeat the process of developing until all tests pass.

A major downside of using Cucumber was that it did not interact well with getting information from the session info on Rails - which is the basis of many of our features (namely the student form) because it requires retrieving the user_id of the user that is logged in and uses that information on whatever page the user is on. Cucumber is not able to retrieve this information, so we needed to add a workaround to solve this issue so we could test some of our major features. The workaround we used is manually setting the variables that use the user's user_id to some test value that we create if we are running Cucumber tests - while it isn't necessarily an elegant solution, it allows us to test critical pieces of our application. Another downside posed by Cucumber is if we have fields in our application that are populated using JavaScript, then Cucumber is not able to retrieve those fields. This is an issue for fields that we populate dynamically, such as on the view results page where we allow the professor/admin to select a semester, and the project field will only show projects from that semester, so we had to make it so the default value for that field is all projects rather than just the ones for that semester.

Cucumber Tests:



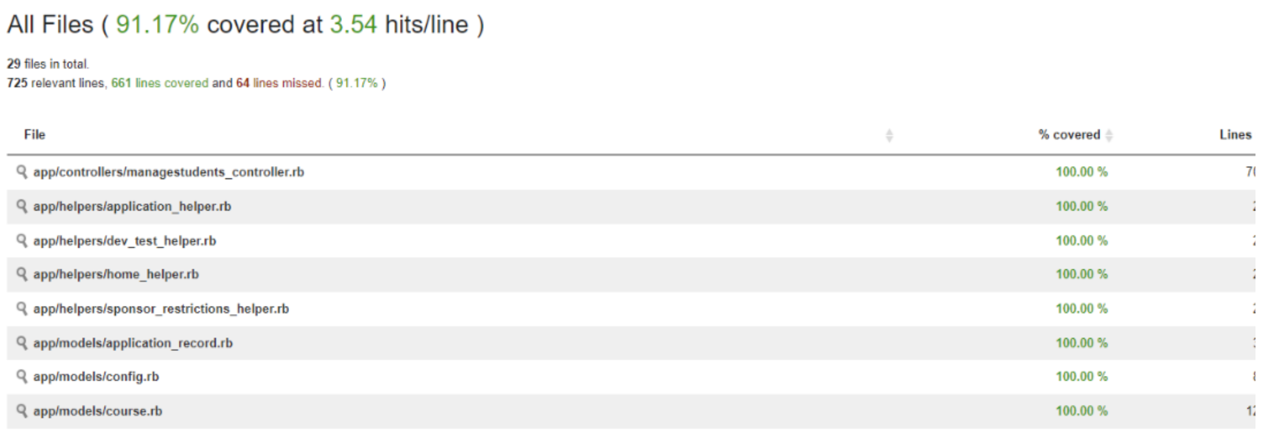
This report will self-destruct in a day. Keep your future reports forever.

56 PASSED

100% passed	a few seconds ago	4.6 seconds
56 executed	Last Run	Duration
linux	ruby 3.2.2	cucumber-ruby 8.0.0

```
> features/admin/change_weights.feature
> features/admin/configuration.feature
> features/admin/dump_database.feature
> features/admin/managestudents.feature
> features/auth/authentication.feature
```

Cucumber Test Coverage:

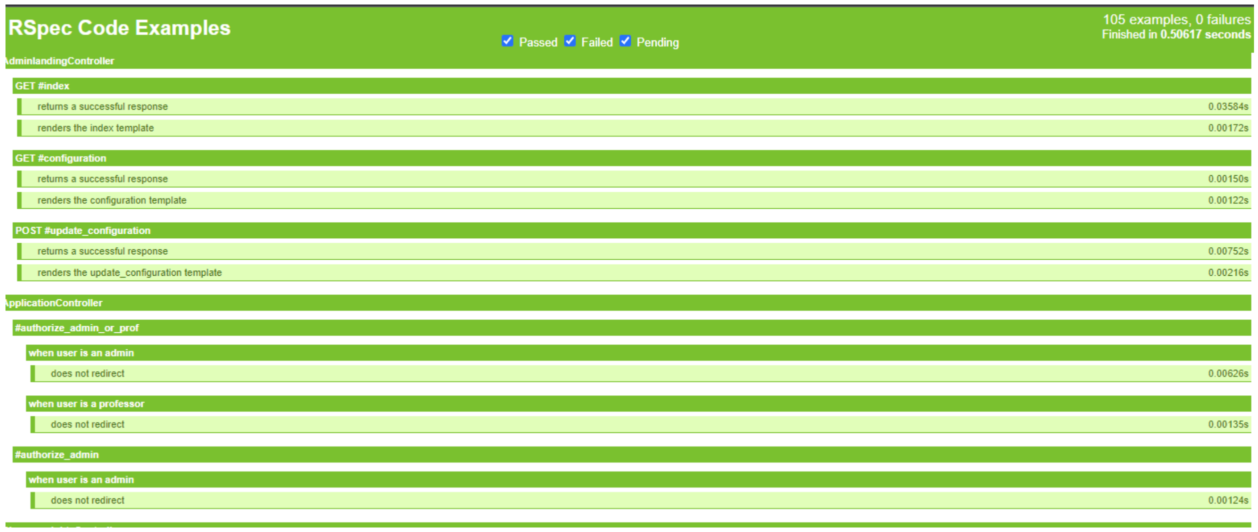


All Files (91.17% covered at 3.54 hits/line)

29 files in total.
725 relevant lines, 661 lines covered and 64 lines missed. (91.17%)

File	% covered	Lines
app/controllers/managestudents_controller.rb	100.00 %	70
app/helpers/application_helper.rb	100.00 %	2
app/helpers/dev_test_helper.rb	100.00 %	2
app/helpers/home_helper.rb	100.00 %	2
app/helpers/sponsor_restrictions_helper.rb	100.00 %	2
app/models/application_record.rb	100.00 %	3
app/models/config.rb	100.00 %	8
app/models/course.rb	100.00 %	12

Rspec Tests:



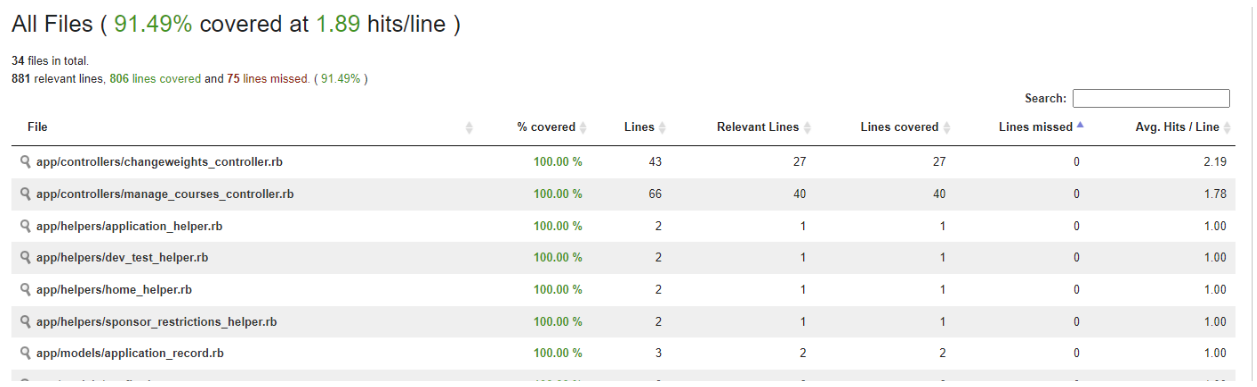
RSpec Code Examples

✓ Passed ✓ Failed ✓ Pending

105 examples, 0 failures
Finished in **0.50617 seconds**

```
dminlandingController
  GET #index
    returns a successful response                          0.03584s
    renders the index template                             0.00172s
  GET #configuration
    returns a successful response                          0.00150s
    renders the configuration template                     0.00122s
  POST #update_configuration
    returns a successful response                          0.00752s
    renders the update_configuration template              0.00216s
pplicationController
  #authorize_admin_or_prof
    when user is an admin
      does not redirect                                    0.00626s
    when user is a professor
      does not redirect                                    0.00135s
  #authorize_admin
    when user is an admin
      does not redirect                                    0.00124s
```

Rspec Test Coverage:



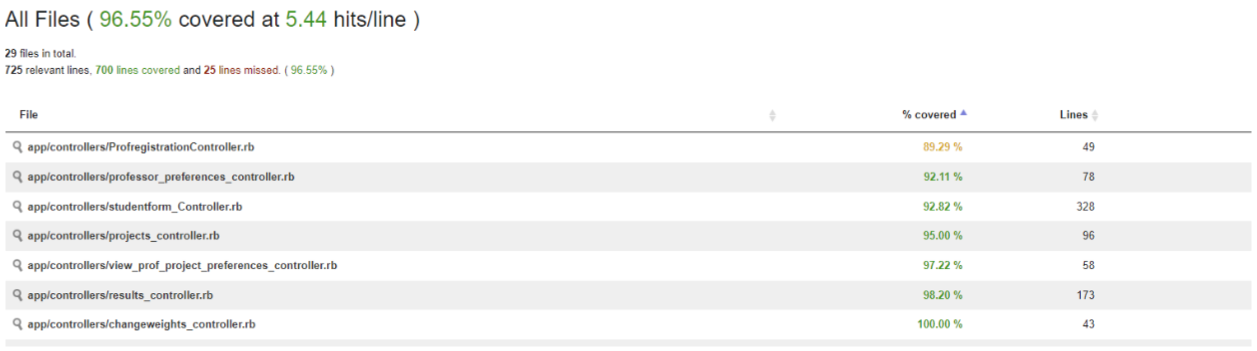
All Files (91.49% covered at 1.89 hits/line)

34 files in total.
881 relevant lines, 806 lines covered and 75 lines missed. (91.49%)

Search:

File	% covered	Lines	Relevant Lines	Lines covered	Lines missed	Avg. Hits / Line
app/controllers/changeweights_controller.rb	100.00 %	43	27	27	0	2.19
app/controllers/manage_courses_controller.rb	100.00 %	66	40	40	0	1.78
app/helpers/application_helper.rb	100.00 %	2	1	1	0	1.00
app/helpers/dev_test_helper.rb	100.00 %	2	1	1	0	1.00
app/helpers/home_helper.rb	100.00 %	2	1	1	0	1.00
app/helpers/sponsor_restrictions_helper.rb	100.00 %	2	1	1	0	1.00
app/models/application_record.rb	100.00 %	3	2	2	0	1.00

Total Coverage:



All Files (96.55% covered at 5.44 hits/line)

29 files in total.
725 relevant lines, 700 lines covered and 25 lines missed. (96.55%)

File	% covered	Lines
app/controllers/ProfregistrationController.rb	89.29 %	49
app/controllers/professor_preferences_controller.rb	92.11 %	78
app/controllers/studentform_Controller.rb	92.82 %	328
app/controllers/projects_controller.rb	95.00 %	96
app/controllers/view_prof_project_preferences_controller.rb	97.22 %	58
app/controllers/results_controller.rb	98.20 %	173
app/controllers/changeweights_controller.rb	100.00 %	43

IX. Configuration/Deployment Approach:

Our project was hosted on GitHub, and we used Git for version control. Our project had two important branches, the "main" branch, where stable releases were deployed, and the “dev” branch, where the ongoing development commits were added. We created a branch for each feature we worked on from the “dev” branch. Once the feature was complete, we created a PR to merge the feature branch to the dev branch. Other team members verified the PR through rspec/cucumber tests and manual testing and if it functioned as expected, we merged the feature branch to dev. At the end of each Iteration, we merged the dev branch into our main branch and deployed the application. As mentioned in the course requirements, we also created a new branch and tagged it at the end of each iteration. However, we didn’t necessarily follow this exact process all of the time.

For example, if we found a bug, made some UI changes, or added more tests to an existing feature to improve its coverage/test quality, then we made a new branch for it (or pushed directly to dev for smaller fixes). This led us to create many branches, making our repository cluttered and resulting in many small commits being made to our project. If we had to change anything going back, it would likely be this - we should have created a uniform method for naming and creating branches (i.e. naming a branch according to the corresponding story #) and keeping bug fixes on the branch that the feature is associated with.

We did a release for each iteration and will have a final release to include the final report along with some smaller changes to the code.

X. Issues With Deployment (Heroku + AWS) and Other Tools:

We initially deployed to AWS to allow us to develop using a pipeline to automatically check for any errors in our code before deploying to production. However, given the customer’s budget constraint, we had to migrate to Heroku for the final deployment. Another issue presented by AWS deployment was that we only had an IP address for our deployed application, and there was no domain name. Due to this, our Google OAuth was not working properly, and this was another one of the reasons we moved deployment to Heroku.

We used SQLite for our local development, but for production, we used PostgreSQL. We had to make some modifications to the database schema code so that it works with the production environment (some data types in Postgres do not exist in SQLite or exist in a different form).

Initially deployed to AWS using AWS code Pipeline and AWS EC2 instances.
One of the problems of pipeline deployment was that if an application that is broken is pushed to production, sometimes the AWS code pipeline script has a hard time dismantling that broken program. This may require manual intervention. This is why it is

imperative for us to check our code to make sure it runs before we deploy it. To resolve this, we added a build stage to test for build success.

One small problem that we had initially was the operating system for our deployment. There was a lack of built-in packages for Amazon Linux 2 This resulted in us having to manually install a lot of programs, such as git, in order for us to get the project running. This was remediated by using a Ubuntu instance instead.

Another issue was when we attempted to use AWS Cloud 9 to access the same AWS instance as our production deployment. the issue here stems from the fact that AWS uses Ruby as an installer for AWS Cloud 9. This version of Ruby conflicts with the version of Ruby that exists in our project. To remedy this, we had to use self-contained Ruby versions that were accessed by the exact path matched to the executable.

Once we moved our deployment over to Heroku, our issues with deployment were solved. However, we can likely make Heroku deployment easier by linking to our GitHub repository to automate our deployment process rather than simply pushing the changes to the deployed app through the command line. This would allow us to have CI/CD where changes made to the main branch will automatically be deployed and would allow us to automate testing before deployment using Github actions rather than having one of us manually run tests to ensure they pass.

XI. Describe Gems + Other Tools:

Name of the gems used	**Functionality**	**What we used for**
pg	Provides a PostgreSQL adapter for connecting Ruby applications to PostgreSQL databases.	This is used to connect to the production PostgreSQL database.
matrix	Provides a Ruby implementation of the Matrix class. The Matrix class can be used to represent and manipulate matrices, which are rectangular arrays of numbers.	This is used in resume parsing to store the feature in the matrix.
dotenv-rails	Manage environment variables in Rails projects by loading them from a .env file into the ENV (environment) of the Ruby process.	This is used to load the variables from the .env file containing the Google OAuth secret key and client ID.
omniauth	Simplifies the process of	This is used to implement the

	implementing third-party authentication (OAuth) in web applications.	functionality of authentication in our application.
omniauth-google-oauth2	An OmniAuth strategy for integrating Google OAuth 2.0 authentication into Ruby applications. OmniAuth is a flexible authentication system Rack applications, and it supports a wide range of providers, including Google.	This is used to implement Google OAuth to verify that the users are from Texas A&M University.
omniauth-rails_csrf_protection	Provides protection against cross-site request forged (CSRF) attacks for OmniAuth requests.	This is added to provide protection for our application against CSRF attacks.
SimpleCov	Helps developers understand how much of their code is covered by automated tests. It tracks which parts of the code are executed during test runs and generates detailed reports, highlighting areas that need more testing.	This is added in order to see what is the coverage percentage being achieved by the test cases that have been written.
capybara	Simulates interactions with web applications. It is commonly used in conjunction with testing frameworks like RSpec or Cucumber to perform end-to-end testing or acceptance testing for web applications.	This simulates the test cases by interacting with the components of the application.
cucumber-rails	Integrates Cucumber, a behavior-driven development (BDD) tool with Ruby on Rails applications.	We have used it to write Cucumber scenarios and step definitions for our application.
rspec-rails	Provides a behavior-driven development (BDD) approach to testing. It is an extension of the RSpec framework, tailored specifically for testing Rails applications.	We have used it to write the test cases for the functionalities implemented in our application - mainly used to test the functionality of our controllers.
factory_bot_rails	factory_bot_rails is a Ruby gem that works in conjunction with the factory_bot gem to provide a flexible and convenient way to define and create factory objects for testing purposes in Rails applications.	We created instances of each model, such as professors, courses, students, users, and many more. This helps perform tests easily.

	Factories are used to create instances of model objects with predefined attributes, making it easier to set up test data.	
pdf-reader	pdf-reader is a Ruby gem that provides a simple and convenient way to parse and extract information from PDF documents. It allows you to read the contents of a PDF file, including text, metadata, and other information.	This is used to read the resumes that are in the pdf file format (the student resumes).
tf-idf-similarity	The tf-idf-similarity gem is a Ruby gem that provides a Vector Space Model (VSM) implementation with term frequency-inverse document frequency (tf-idf) weights. It enables you to calculate the similarity between texts based on their shared vocabulary and the relative importance of each term. This gem is particularly useful for tasks like text classification, document retrieval, and topic modeling.	This is being used in the resume parsing feature. It calculates the TF-IDF (Term Frequency-Inverse Document Frequency) similarity between text documents. In our case this is the project descriptions and the resume's text.
bootsnap	Bootsnap is a gem improving the startup time of Ruby applications. It does this by caching Ruby's precompiled bytecode, which reduces the need to load and parse Ruby files at the start of each application. This can significantly speed up the boot time of Ruby applications, especially those with large codebases.	This is used in our application to speed up boot time by caching expensive operations.

XII. Repo Contents and Process/Scripts Used to Deploy Code:

Heroku Deployment Instructions:
https://github.com/Capstone-Matching/capstone-matching/blob/main/documentation/Fall 2023/deploy_instructions.pdf

This document is located in the documentation/Fall2023/ directory, and it contains a guide that gives initial set up instructions if you are deploying on Heroku. The instructions are listed in the exact order that you should perform them which essentially follows this order: 1. Initial Cloning (installing all of the gems and setting up the development DB) 2. Google OAuth setup (getting your client id and client secret) 3. Heroku deployment (adding Heroku Postgres and connecting your local repo to the Heroku repo, and adding the environment variables to your Heroku instance), 4. Completing OAuth setup (adding your Heroku deployment URL to the authorized URIs for your OAuth token) 5. Setting up the database on Heroku (using seeds) 6. Running tests (Cucumber and Rspec). This document provides additional commands that can be used to manage the database if changes need to be made.

AWS Setup Script:
https://github.com/Capstone-Matching/capstone-matching/blob/main/cloud-init.sh

We also have instructions on how to deploy to AWS in the script linked above also in the repo README, but those are likely not going to be relevant if you want to use OAuth unless you choose to purchase your own domain name.

For testing, an important thing to note is that the simplecov coverage report will be automatically added to the coverage directory, and you will be able to view it if you open index.html in that directory. To ensure that you are only getting coverage info from a specific test (Cucumber or Rspec), make sure to delete .last_run.json and .resultset.json from the coverage/assets/ directory - if you want to get the coverage of all tests then you do not need to remove them after running 1 type of test.

Seeds: https://github.com/Capstone-Matching/capstone-matching/blob/main/db/seeds.rb

Another important thing to note is that whenever a new team takes over the project, they **MUST ADD THEMSELVES TO THE SEED AS ADMINS.** This ensures that they are able to log in to the application in production (for development, this doesn't matter as much because we don't have the page checks in the development environment). The seed file is in db/seeds.rb/ - see our existing seeds for examples of how to add yourself as an admin.

XIII. Links

PM Tool Page: https://github.com/orgs/Capstone-Matching/projects/3

GitHub Repo: https://github.com/Capstone-Matching/capstone-matching

Deployment (will likely be moved later to an instance on customer's account):
https://capstone-matching-tamu-test-770362a3bf72.herokuapp.com/

Presentation + Demo Video: https://youtu.be/PIhjdXbvH_0

Final Slides:
https://github.com/Capstone-Matching/capstone-matching/blob/dev/documentation/Fall2023/Final%20Presentation.pptx

UML Diagram:
https://www.figma.com/file/EpeNjNgHs6UihGPy9S9gx8/csce606db?type=whiteboard&node-id=0%3A1&t=5FfOMtMaqCWFCYw2-1

Group Contact Info:

- Hunter Zacha - hunterzacha@tamu.edu
- Aakashdeep Sil - asil@tamu.edu
- Yash Patil - yapatil@tamu.edu
- Shripad Kulkarni - shripad@tamu.edu
- Zachary Laguna - lagunazachary@tamu.edu
- Sathvik Kote Revanasiddappa - sathvikkote@tamu.edu