FINAL REPORT

PROJECT: TEAM FORMATION

1. PROJECT SUMMARY:

The objective of this application is to automate and streamline the process of forming student teams based on key attributes like gender, ethnicity, and programming proficiency. The tool directly addresses the professor's (main stakeholder) need for a customizable, inclusive, and efficient team formation system. The application's algorithm uses these criteria to automatically generate diverse and balanced teams, ensuring gender equity, preventing isolation of underrepresented ethnic groups, and promoting a mix of programming skills. This system aims to save professors time by eliminating manual team assignments while promoting fairness and inclusion in the team formation process.

After logging into the application, professors can securely create multiple forms, upload student lists for each form, define key attributes (such as gender, ethnicity, and skill level), assign weightages to these attributes, and set submission deadlines. Once the form is published, students can access it, submit their data, and update their responses up until the deadline. After the submission period ends, professors can automatically generate teams based on the defined criteria. They also have the flexibility to modify the form, adjust attributes and their weightages, close the form, and republish it as needed. This functionality allows for ongoing adjustments and ensures that the team formation process remains dynamic, meeting the evolving needs of the course.

---

2. USER STORIES:

SNo.	User Story	Points	Description	Status
1	Professor previews the form	3	Allows professors to preview the form before publishing it to students.	Implemented in Sprint 1
2	Professor saves form as draft	3	Allows professors to preview the form before publishing it to students.	Implemented in Sprint 1
3	Professor sets a deadline for the form submission	3	Allows professors to set a submission deadline for students to complete the form.	Implemented in Sprint 1
4	Professor updates the deadline for the form submission	3	Enables professors to modify the submission deadline after setting it initially.	Implemented in Sprint 1

5	Professor uploads an Excel file with student information in the correct format	3	Professors can upload student data using an Excel file that follows the correct format.	Implemented in Sprint 1
6	Professor uploads an Excel with missing Email column	1	Alerts the professor when the uploaded Excel file is missing the Email column.	Implemented in Sprint 1
7	Professor uploads an Excel with missing UIN column	1	Notifies the professor if the uploaded Excel file is missing the UIN column.	Implemented in Sprint 1
8	Professor uploads a file which has unsupported format	1	Displays an error message if the uploaded file is in an unsupported format.	Implemented in Sprint 1
9	Professor uploads an empty excel	1	Warns the professor if an empty Excel file is uploaded.	Implemented in Sprint 1
10	Professor adds a text input question	3	Allows the professor to add a text input question to the form.	Implemented in Sprint 1
11	Professor removes a question from an existing form	3	Enables the professor to remove a question from an existing form.	Implemented in Sprint 1
12	Professor sets attribute weightage	3	Lets the professor define weightages for each form attribute (e.g., gender, ethnicity).	Implemented in Sprint 1
13	Successful login with valid credentials	3	Ensures that the professor can log in successfully using valid credentials.	Implemented in Sprint 1
14	Failed login due to incorrect credentials	3	Displays an error when the professor enters incorrect login credentials.	Implemented in Sprint 1
15	Professor creates a new form	3	Allows the professor to create a new data collection form for team formation.	Implemented in Sprint 1
16	Professor adds a specific attribute	3	Enables the professor to add a specific attribute (e.g., ethnicity or skill level) to the form.	Implemented in Sprint 1
17	Integrations among features and deployment	2	Ensures that all features are integrated and work properly when deployed.	Implemented in Sprint 1

18	Save as Draft	2	Allows the professor to save the form as a draft to edit later.	Implemented in Sprint 1
19	Display Teams for Students	2	Displays the final team assignment details to students once teams are generated.	Implemented in Sprint 2
20	Edit Student Details Form	3	Allows the professor to edit the student details form for each student.	Implemented in Sprint 2
21	Minimum Required Gender Representation in Teams	3	Ensures that teams meet the required minimum gender representation, such as at least two females per team.	Implemented in Sprint 2
22	Block Submissions After Deadline As a professor	2	Prevents students from submitting their forms after the specified deadline.	Implemented in Sprint 2
23	Add Multiple Choice Questions (MCQ) without correct answers	3	Enables the professor to add multiple choice questions without specifying correct answers.	Implemented in Sprint 2
24	Apply CSS styling to web pages	3	Applies consistent CSS styling across all web pages for visual consistency.	Implemented in Sprint 2
25	System ensures varied programming proficiency in teams	5	Ensures that teams have a mix of programming proficiency levels, from high to low skill.	Implemented in Sprint 2
26	Student enters invalid data into a field	1	Displays an error message if a student enters invalid data into any field.	Implemented in Sprint 2
27	Set Attribute Weightage with Total Sum Constraint	1	Ensures that the total weightage for all attributes sums up to a predefined value of 1.	Implemented in Sprint 2
28	System includes underrepresented groups in teams 1	5	The system ensures that underrepresented ethnic or gender groups are included in teams.	Implemented in Sprint 2
29	Successful login with valid credentials for students	3	Allows students to log in successfully using valid credentials.	Implemented in Sprint 2
30	Failed login due to incorrect credentials for	2	Displays an error message when students enter incorrect login	Implemented in Sprint 2

	students		credentials.	
31	Student views their assigned team details	3	Allows students to view the team they've been assigned to once teams are generated.	Implemented in Sprint 2
32	No of teams	5	Calculates the total number of teams to be generated based on number of students and their responses.	Implemented in Sprint 2
33	Successfully publishing a form	3	Allows the professor to publish a form after creating or modifying it.	Implemented in Sprint 2
34	Closing a published form	1	Enables the professor to close a form after it's been published, blocking further submissions.	Implemented in Sprint 2
35	Gender Logic - part 2	2	Implementing gender-related logic to ensure gender balance in teams.	Implemented in Sprint 3
36	Detailed Response	2	Captures detailed responses from students for each form attribute.	Implemented in Sprint 3
37	Ethnicity Logic - part 2	3	Implementing ethnicity-related logic to ensure diverse representation in teams.	Implemented in Sprint 3
38	Update Existing Form Responses On Editing The Form	2	Updates existing student responses if the form is edited after submission.	Implemented in Sprint 3
39	Block Submissions After Deadline	2	Prevents students from submitting responses once the deadline has passed.	Implemented in Sprint 3
40	Swap Team Members to Ensure Balanced Teams While Maintaining Gender and Ethnicity Constraints	3	Allows the algorithm to swap team members to ensure balanced diversity while maintaining constraints.	Implemented in Sprint 3
41	View Change Summary	2	Allows professors and students to view a summary of changes made to the form or teams.	Implemented in Sprint 3
42	Team Generation Button	3	Provides the professor with a button to generate student teams automatically.	Implemented in Sprint 3

43	Team Data Export	2	Allows the professor to export team data in a desired format (e.g., Excel or CSV or PDF).	Implemented in Sprint 3
44	Programming Proficiency Logic - part 2	3	Implementing the programming proficiency logic to ensure varied skill levels in teams.	Implemented in Sprint 3
45	Save Form Responses as Draft	2	Allows students to save their form responses as a draft before final submission.	Implemented in Sprint 3
46	Reopen Form	2	Enables the professor to reopen a closed form for additional edits or submissions.	Implemented in Sprint 3
47	Block Edit While Form is Published	2	Prevents changes to the form after it has been published to students.	Implemented in Sprint 3
48	Unify Deadline Over All Time Zones	3	Standardizes the deadline across all time zones to ensure consistency for all students.	Implemented in Sprint 3
49	Code Refactoring	2	Refactoring the code to improve efficiency and maintainability.	Implemented in Sprint 3
50	Populating teams based on programming proficiency (completion)	3	Implementing the team generation logic based on programming proficiency.	Implemented in Sprint 4
51	Restrict Ineligible Students from Accessing Forms with Error Notification	2	Prevents ineligible students from accessing the form, displaying an error message.	Implemented in Sprint 4
52	Ensure Correct Navigation and Naming for “Duplicate Form” Action	3	Ensures that navigation and naming conventions for duplicating forms are consistent and correct.	Implemented in Sprint 4
53	Limit “View Responses” to Display Submitted Student Responses Only	3	Ensures that only submitted student responses are displayed when viewing form responses.	Implemented in Sprint 4
54	Populating teams based on gender (completion)	3	Final Implementation of the team formation logic based on gender constraints.	Implemented in Sprint 4
55	Fix and Align Update_deadline.feature and	3	Alignment of deadline-related feature tests with the specified requirements.	Implemented in Sprint 4

	deadline_Error_handling. feature Scenarios with Requirements			
56	UI additions - Student Side	3	Implementation of UI enhancements specifically for the student-facing side of the application.	Implemented in Sprint 4
57	Enhance “Upload Students” on View Form Page with Guidelines, Sample Sheet, and Navigation	3	Improvement of the "Upload Students" feature with additional guidelines and sample sheet options.	Implemented in Sprint 4
58	Fix Unique Index Error for UIN During Bulk Insert in upsert_all Operation	2	Fixes an error related to unique UINs when performing bulk inserts in the database.	Implemented in Sprint 4
59	Update Redirection Logic and Step Definitions in Forms_controller.feature for Error Handling and View Page Access	1	Updates redirection logic to handle errors and page access during form submission.	Implemented in Sprint 4
60	Checking access for students before displaying form	2	Verifies student eligibility and access before allowing them to view the form.	Implemented in Sprint 4
61	Revise Button Availability and Route, and Correct Formatting in “Preview Form” Display	2	Ensures buttons and routes are correctly displayed and formatted in the “Preview Form” view.	Implemented in Sprint 4
62	UI additions - Instructor Side	2	Adds UI enhancements for the professor’s side of the application.	Implemented in Sprint 4
63	Allow Draft Saving Without Completion, Update Response Submission Redirect, and Add Navigation Button	3	Lets students save drafts without completing the form and improves navigation for submission.	Implemented in Sprint 4
64	Changes to form response page of students	3	Implements updates to the student form response page for a better user experience.	Implemented in Sprint 4

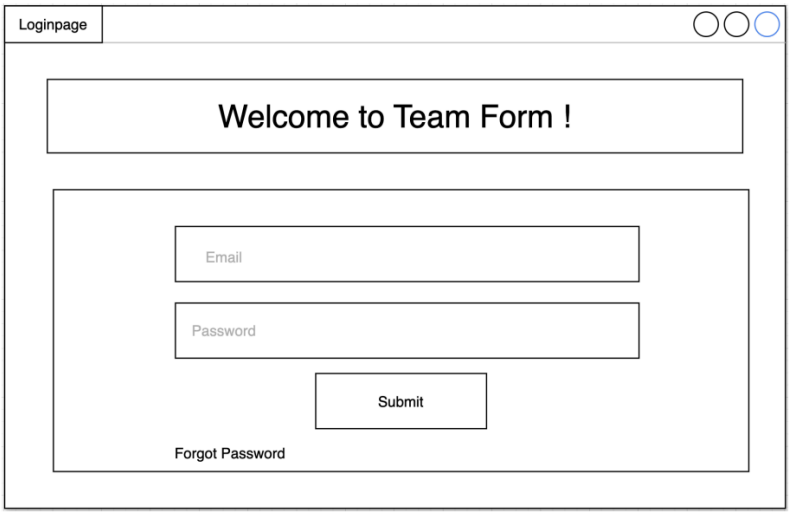
65	Populating teams based on ethnicity (completion)	5	Finalizes the team generation logic based on ethnicity constraints.	Implemented in Sprint 4
66	Making ethnicity information mandatory	1	Ensures that the professor must provide ethnicity as an attribute when creating the form.	Implemented in Sprint 4
67	Making gender information mandatory	1	Ensures that the professor must provide gender as an attribute when creating the form.	Implemented in Sprint 4
68	Ui improvements	5	General improvements to the user interface to enhance usability and aesthetics.	Implemented in Sprint 4
69	Weightage Adjustment for Gender and Ethnicity Attributes	3	Allows for adjustments to the weightage assigned to gender and ethnicity attributes.	Implemented in Sprint 4
70	Team Generation completion	3	Final implementation of the team generation process, ensuring all criteria are met.	Implemented in Sprint 4
71	Export Teams Fix	3	Fixes issues related to exporting team data correctly in required formats	Implemented in Sprint 4

Note: All the user stories outlined for this project were successfully implemented across multiple sprints. Each user story, from basic form creation to team generation based on various attributes, was tackled within its respective sprint, with points allocated based on complexity. There were no significant changes to the core user stories throughout the development process, as the requirements were well-defined and followed through in each sprint. The majority of the features were completed as planned, with the backend logic evolving over time to accommodate different team formation algorithms and improve accuracy.

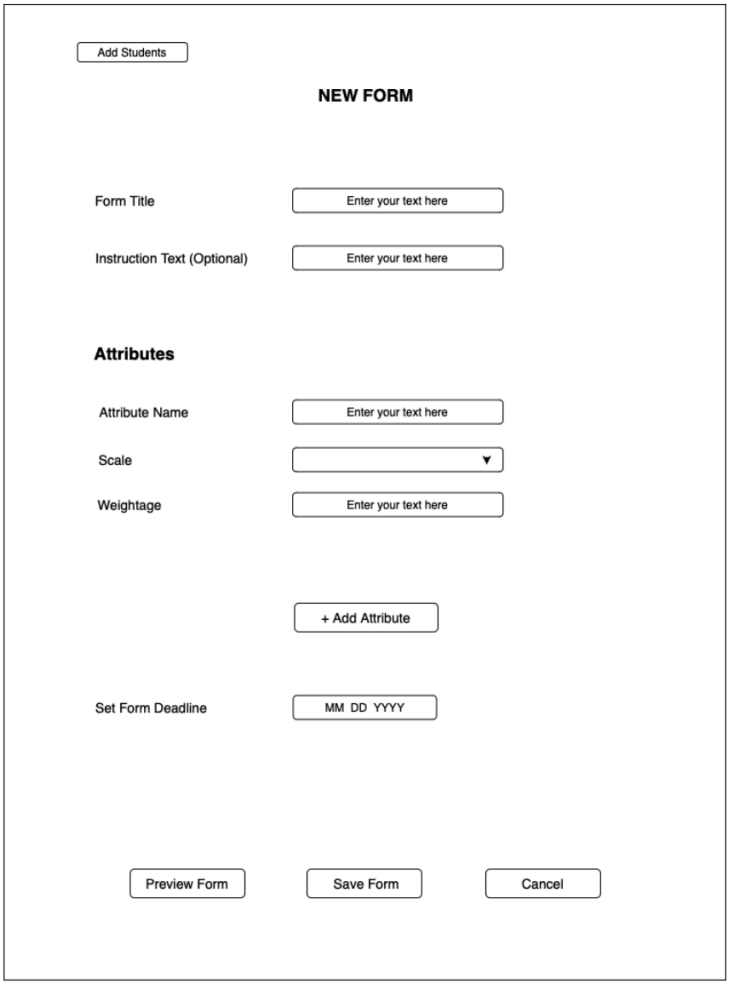
---

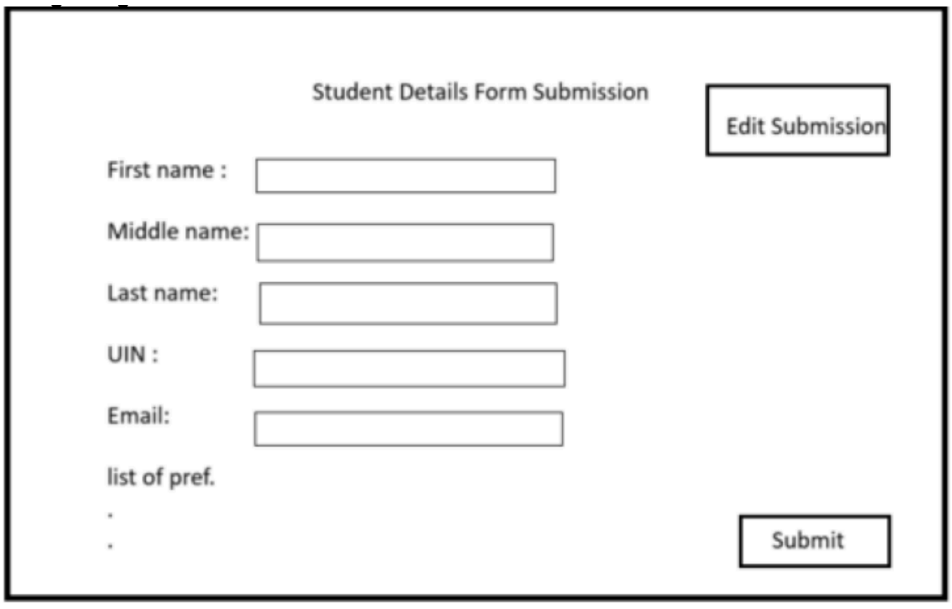
3. STORYBOARDS & UI MOCK-UPS

1. Login page:

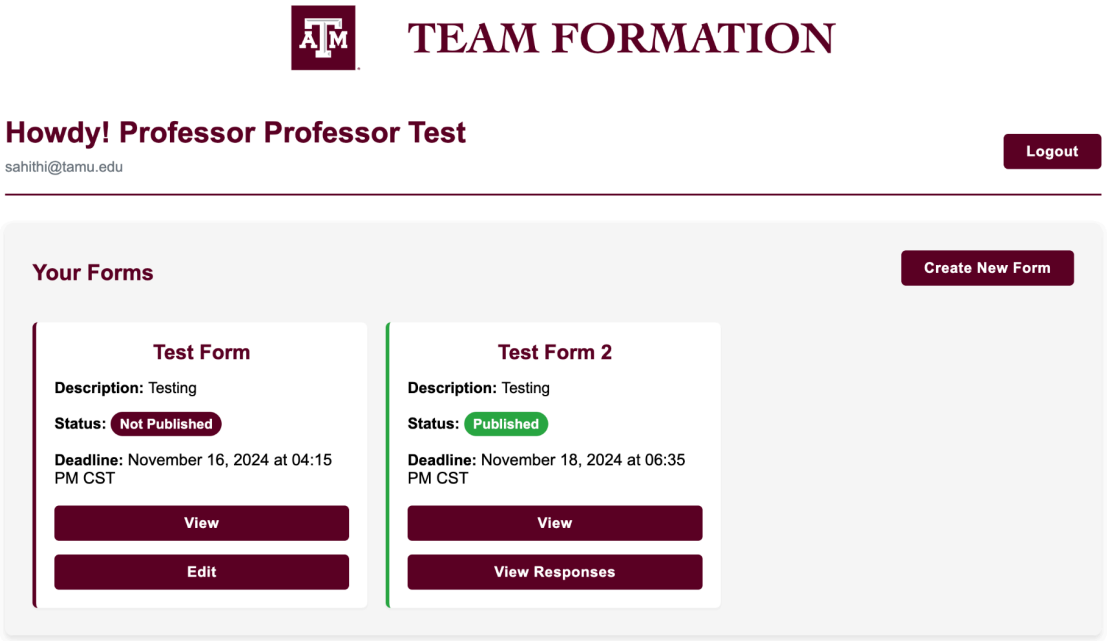


2. Form Page:


Add Students
NEW FORM
Form Title
Enter your text here
Instruction Text (Optional)
Enter your text here
Attributes
Attribute Name
Enter your text here
Scale
Weightage
Enter your text here
+ Add Attribute
Set Form Deadline
MM DD YYYY
Preview Form
Save Form
Cancel


Student Details Form Submission
Edit Submission
First name :
Middle name:
Last name:
UIN :
Email:
list of pref.
.
.
Submit

4. FINAL APPLICATION SCREENSHOTS :



TEAM FORMATION

Editing Form: Test Form

Back to Forms

Testing

Edit Form Details

Name

Test Form

Description

Testing

Set Deadline

11/16/2024, 04:15 PM

Save Deadline

Save Form | Preview Form | Duplicate Form | Upload Students

Ethnicity (mcq)

- Option 1: Asian
- Option 2: Non Asian

Current Weightage: N/A

Destroy Attribute

Gender (mcq)

- Option 1: Male
- Option 2: Female
- Option 3: Other
- Option 4: Prefer not to say

Current Weightage: N/A

Destroy Attribute

Add Attribute

Attribute Name

Attribute Type

Text

Save Attribute

TEAM FORMATION

Upload Students

Choose a file to upload

Choose File No file chosen

Upload Students

Back to Form

Download Sample CSV

TEAM FORMATION

Teams for Test Form 2

Back to Form

Export Teams

Excel

Export

Section: 1

Team 1

Team Size: 4

Name	UIN	Email
Q	111222347	q@tamu.edu
P	111222346	p@tamu.edu
A	111222331	sahithi@tamu.edu
N	111222344	n@tamu.edu

Team 2

Team Size: 4

Name	UIN	Email
LongLongLongLongLongName	111222332	longlonglongglongglongemail@tamu.edu
F	111222336	f@tamu.edu
J	111222340	j@tamu.edu

Team 3

Team Size: 3

Name	UIN	Email
C	111222333	c@tamu.edu
LongLongLongLongLongName2	111222337	g@tamu.edu
K	111222341	k@tamu.edu

---

5. TEAM ROLES:

During the course of the project, the roles of Product Owner and Scrum Master were rotated across different sprints to provide everyone with leadership experience and ensure varied perspectives. The following changes took place:

1. **Sprint 1:**
    a. Product Owner: Sahithi Duppati
    b. Scrum Master: Phani Jyothi Kurada
2. **Sprint 2:**
    a. Product Owner: Hitha Magadi Vijayanand
    b. Scrum Master: Sai Jaideep Reddy Mure
3. **Sprint 3:**
    a. Product Owner: Ramya Reddy Gotika
    b. Scrum Master: Rithika Sapparapu
4. **Sprint 4:**
    a. Product Owner: Uday Kumar Reddy Mettukuru
    b. Scrum Master: Raja Karthik Vobugari

---

5. SCRUM ITERATIONS SUMMARY:

1. Iteration 1:
    a. Accomplishments:
        i. Secure login for professors with college-issued username and password.
        ii. Ability for professors to create, edit, and set deadlines for forms, as well as upload student details via Excel.
        iii. Students can log in via Google, view the form, and submit/edit responses until the deadline.
    b. Points completed: 48
2. Iteration 2:
    a. Accomplishments:
        i. Professors can create forms with multiple answer types (numerical, multiple-choice, text) and set deadlines.
        ii. Team formation based on class size, typically 3 or 4 students per team.
    b. Points completed: 56
3. Iteration 3:
    a. Accomplishments:
        i. Professors can view student responses, block submissions after the deadline, and view change summaries for forms.
        ii. Team generation button implemented for automatic team creation, along with team data export options.
        iii. Students can save responses as drafts and professors can reopen forms for further submissions.

iv. Various backend enhancements, including code refactoring and handling deadlines across time zones.

b. Points completed: 49

4. Iteration 4:
   a. Accomplishments:
      i. Backend logic for team creation was introduced, ensuring randomness in team assignments.
      ii. End-to-end backend development completed with robust testing.
      iii. UI finalization, including styling, navigation improvements, and enhancements for specific components (e.g., "Upload Students" and "Preview Form").
      iv. Team population logic completed, with mandatory fields for gender and ethnicity.
      v. Error handling and response page updates implemented.
   b. Points completed: 91

---

6. TEAM MEMBER CONTRIBUTION SUMMARY:

Name	**Sprint 1**		**Sprint 2**		**Sprint 3**		**Sprint 4**		**Total**
	Stories	Points	Stories	Points	Stories	Points	Stories	Points	Points
Raja Karthik Vobugari	7	5	3	7	2	5	5	14	31
Sai Jaideep Reddy Mure	2	8	3	8	2	5	4	9	30
Uday Kumar Reddy Mett…	3	8	3	11	6	13	1	1	33
Sahithi Duppati	1	3	2	6	2	5	5	16	30
Hitha Magadi Vijayanand	4	8	2	6	3	6	3	10	30
Phani Jyothi Kurada	3	3	2	6	2	5	5	16	30
Rithika Sapparapu	2	6	2	6	2	5	5	13	30
Ramya Reddy Gotika	3	7	2	6	2	5	3	12	30
							Total		**244**
							Avg/student		30.5

---

7. CLIENT MEETING DETAILS

1. Meeting 1: Introductory Meeting
    a. Date: 23 September 2024
    b. Time: 4:30 PM
    c. Place: ZACH 424
    d. Discussion Points:
        i. Project Understanding: Team presented the project's objective to automate student team formation using collected student data.
        ii. Client Requirements: Emphasis on collecting programming experience, gender, and ethnicity data for team formation.
        iii. Team Formation Logic:
            1. Prioritizing programming experience for balanced teams.
            2. Gender and ethnicity considerations to avoid isolation.
            3. Handling missing information in team formation.
        iv. Other Considerations:
            1. Historical data from previous semesters available for testing.
            2. Team size of 4 members, with 3-member teams for smaller classes.
            3. Request for downloadable CSV reports for team assignment uploads to Canvas.
2. Meeting 2: Sprint-1 MVP Demo
    a. Date: 30 September 2024
    b. Time: 4:30 PM
    c. Place: Online Meeting
    d. Discussion Points:
        i. UI Style Approval: Client approved the current UI style, with potential future improvements.
        ii. "Add Students" Feature: Functionality was implemented, but UI visibility was requested.
        iii. Focus on Student Side: Client emphasized completing student-side features, including form access, editing, and submission before deadlines.
        iv. Team Formation Logic: Prioritize the development of the team formation algorithm.
3. Meeting 3: Sprint-2 MVP Demo
    a. Date: 07 October 2024
    b. Time: 4:30 PM
    c. Place: Online Meeting
    d. Discussion Points:
        i. Student Module: Client satisfied with functionalities allowing students to fill and submit forms.
        ii. Backend Logic: Flowchart presentation for backend logic, approved by client.
        iii. Backend Development: Discussion on the progress and strategy for the upcoming sprint.

        iv. Form Responses: Client requested the ability to access all submitted student form responses.
4. Meeting 4: Sprint-2 MVP Demo (Physical)
    a. Date: 21 October 2024
    b. Time: 4:30 PM
    c. Place: ZACH-425
    d. Discussion Points:
        i. Student Module Approval: Client approved the student-side features implemented.
        ii. Backend Logic Approval: The backend flowchart was explained, and the client approved the logic.
        iii. Completion Strategy: Discussed next steps in the backend logic and enhancements for the following sprint.
        iv. Form Response Access: Client requested features for accessing all submitted form responses.
5. Meeting 5: Sprint-3 Update
    a. Date: 21 October 2024
    b. Time: 4:15 PM
    c. Place: ZACH 424
    d. Discussion Points:
        i. Website Review: Client satisfied with the website's design and functionality, noting improvements.
        ii. Response Visibility: Client requested the ability to view student responses.
        iii. Feedback: Client expressed satisfaction with current progress, requesting backend logic for the next sprint.
        iv. Timeline: Discussed the implementation timeline for backend features and data processing.
6. Meeting 6: Sprint-3 MVP Demo
    a. Date: 07 November 2024
    b. Time: 1:00 PM
    c. Place: ZACH 424
    d. Discussion Points:
        i. Feature Demonstrations:
            1. Block submissions after deadline
            2. Reopen form feature
            3. Unified time zone handling
            4. Save responses as drafts
            5. Automated team generation
        ii. Backend Logic: Detailed explanation of backend logic for each feature, with client approval.
        iii. Next Steps: Discussed Sprint 4's prioritized stories and requested client feedback on planned app enhancements.
7. Meeting 7: Sprint-4 Update

a. Date: 07 November 2024
b. Time: 1:00 PM
c. Place: ZACH 424
d. Discussion Points:
    i. Feature Demonstrations:
        1. Team formation logic, including balancing by gender, ethnicity, and skill
        2. UI improvements
    ii. Backend Logic: Detailed explanation of backend functionality, with client approval.
    iii. Next Steps: Discussed Sprint 4's enhancements and planned app improvements, awaiting client feedback.

8. Meeting 8: Group by Ethnicity Feature Demo
    a. Date: 11 November 2024
    b. Time: 1:00 PM
    c. Place: ZACH 424
    d. Discussion Points:
        i. Feature Demonstrations:
            1. "Group by Ethnicity" feature demonstrated using a pool of 50 students, including edge cases.
            2. Client approval of the logic.
        ii. Feature Finalization: Proposed improvements for Sprint 4 discussed and open for feedback.
        iii. Next Steps: Gathered client feedback on potential improvements to the functionality.
9. Meeting 9: Skill Balance Feature Demo
    a. Date: 14 November 2024
    b. Time: 1:00 PM
    c. Place: ZACH 424
    d. Discussion Points:
        i. Feature Demonstrations: "Skill Balance" feature demonstrated both as an individual functional block and as part of the full team formation algorithm using a pool of 75 students.
        ii. Client Data Review: Received data from the client for application testing.
        iii. UI Enhancements: Presented implemented UI improvements to date.
        iv. Next Steps:
            1. Continue with application testing based on the provided data.
            2. Prepare for final review of Sprint 4 features.
10. Meeting 10: Sprint-4 Final MVP Demo
    a. Date: 25 November 2024
    b. Time: 1:00 PM
    c. Place: ZACH 424
    d. Discussion Points:
        i. Feature Demonstrations:

1. End-to-end team formation logic demonstration, including grouping by gender, ethnicity, and skill balance.
2. UI updates with enhanced usability.

ii. Client Feedback: Client expressed satisfaction with the demonstrated features, confirming alignment with project goals.

iii. Next Steps:
1. Finalizing detailed documentation for the application.
2. Preparing Software Requirements Specification (SRS) for formal documentation.
3. Planning for the next meeting to review project closure and documentation.

---

8. BDD/TDD PROCESS

Approach :
For this project, we used TDD as our primary approach for ensuring code quality and functionality. The process started with writing tests before implementing any new feature. This approach helped us clearly define the expected behavior of each feature and catch potential issues early. We used RSpec for writing unit tests for the backend logic, including validations, model methods, and controller actions. The tests focused on ensuring the correct handling of form submissions, validation of user inputs, and the accuracy of team formation algorithms based on the professor's input criteria (e.g., gender, ethnicity, programming proficiency). In addition to RSpec for backend testing, we also used Cucumber for Behavior-Driven Development (BDD) to write acceptance tests.

Benefits :
TDD helped us ensure that every function we wrote had a corresponding test, resulting in fewer bugs and better-quality code. By writing tests first, we were forced to design our code in a modular, testable way from the start. This also made it easier to refactor code with confidence, knowing that the tests would catch any regressions. By running tests frequently, we were able to identify bugs early in the process. This allowed us to address issues quickly, minimizing their impact on the overall development schedule. With automated tests in place, it was easier to pinpoint which feature or logic change caused a problem.

Challenges :
Initially, the team faced a steep learning curve, especially with integrating **Cucumber** and **RSpec** for BDD and TDD. Understanding the structure and syntax of Cucumber scenarios, as well as writing tests for complex algorithms like team formation, was time-consuming at the start. However, as we got more comfortable with the tools, the process became smoother. As the project evolved and features were added, maintaining the growing test suite became more challenging. Some tests needed to be updated to reflect changes in requirements or logic. If the tests were too specific, changes in the application's structure or flow could cause a large

number of tests to break, requiring significant effort to fix. Writing tests for complex logic, such as the team formation algorithm, was difficult because it required extensive test cases to cover all possible scenarios (e.g., gender and ethnicity constraints, skill-level diversity, etc.). While it was possible to define many edge cases, it was a time-consuming task and often required iterations to get the right balance of tests.

---

9. CONFIGURATION MANAGEMENT

Branching Strategy :
For this project, we adopted a branch-based configuration management approach to ensure that our codebase remained clean, manageable, and aligned with the development flow. The primary strategy involved creating a separate branch for each user story or feature being worked on. This helped us with isolated development, ease of collaboration and simplified code review.

- **Main Branch:** This branch always reflected the most stable version of the code.
- **Feature Branches:** These branches were created for each user story and feature (one branch per user story), where the development work took place.

Release Strategy :

Releases were typically aligned with the completion of major milestones or sprint goals. We had one release per sprint (aligned with each major feature set being completed). This meant that at the end of each sprint (e.g., Sprint 1, Sprint 2, etc.), we performed a release to ensure the project was in a deployable state.

- **Release Branches:** These were created for the purpose of preparing the application for deployment at the end of each sprint. After the feature branches were merged, a release branch was created, tested, and deployed to Heroku.

Spikes :

We did have some instances where we had to perform spikes, which are time-boxed investigations to explore solutions for complex or unknown technical challenges. Examples of spikes in our project included:

1. **Algorithm Testing for Team Formation**: During the development of the team formation algorithm, we encountered challenges in balancing gender, ethnicity, and skill levels. A spike was conducted to investigate how best to implement the algorithm that would meet the diversity criteria while ensuring teams had balanced proficiency.
2. **Handling Excel File Uploads**: Initially, we had to investigate how to efficiently handle and validate Excel file uploads from professors, as there were multiple formats, potential errors, and varying data quality to consider. This led to a spike on third-party libraries for handling Excel parsing and validation.

10. TOOLS AND GEMS

We used the following tools to enhance our development process:

1. **Tools:**
    a. **GitHub:** Version control and collaboration platform for managing code.
        i. Benefits: It facilitated easy collaboration, version history management.
        ii. Challenges: We faced challenges ensuring a consistent branching strategy and handling merge conflicts during collaboration.
    b. **GitHub Projects:** Project management tool for organizing tasks and issues.
        i. Benefits: It helped us track progress, organize features/bugs, and efficiently prioritize tasks within the repository.
        ii. Challenges: The interface was a bit cumbersome for a project of our scale, and managing multiple workflows was time-consuming.
    c. **CodeClimate:** Code quality analysis tool.
        i. Benefits: It provided insightful reports on code maintainability and test coverage, helping us maintain a high-quality codebase
        ii. Challenges: At times, it offered false positives, and required fine-tuning to configure for optimal use.
    d. **SimpleCov** : Test coverage measurement tool for Ruby projects.
        i. Benefits: It helped us track test coverage and identify untested code, ensuring we maintained proper test coverage.
        ii. Challenges: It required proper configuration to ensure accurate coverage reporting.
2. **Gems :**
    a. **Core Gems :**
        i. rails - Framework for building web applications
        ii. omniauth-google-oauth2 - Google OAuth2 strategy for OmniAuth
        iii. JSON Web Token gem for handling authentication tokens
        iv. Ruby's built-in library for handling CSV files
    b. **Development & Testing Gems :**
        i. rubocop - Ruby static code analyzer (styling and linting)
        ii. rubocop-rspec - RuboCop extension for RSpec code
        iii. rspec-rails - RSpec testing framework for Rails
        iv. cucumber-rails - Cucumber integration for acceptance testing in Rails
        v. simplecov - Code coverage analysis tool for Ruby
    c. Test Specific Gems :
        i. capybara - Integration testing tool for simulating user interactions
        ii. selenium-webdriver - WebDriver for browser automation in tests
    d. Development Gems :
        i. web-console - Debugging console for Rails exceptions
        ii. sqlite3 - SQLite database (used in development/test environments)
    e. Production Gems :

i. pg - PostgreSQL database adapter for ActiveRecord

f. Miscellaneous Gems :

i. rufus-scheduler - Scheduling gem for running jobs/tasks
ii. rubyzip - ZIP file handling
iii. axlsx - Library for creating and modifying Excel files
iv. caxlsx_rails - Rails integration for `axlsx`
v. wicked_pdf - PDF generation from HTML (via wkhtmltopdf)
vi. wkhtmltopdf-binary - Binary for wkhtmltopdf, used in `wicked_pdf`

---

11. DEPLOYMENT PROCESS AND SCRIPTS

All code, including Cucumber and RSpec, has been pushed to the public GitHub repository. This section outlines the process used to deploy the code on Heroku. We have thoroughly verified that everything needed for the deployment process is present, ensuring a smooth deployment without any legacy project issues.

Step 1: Log in to Heroku

To begin, log in to your Heroku account by running the following command in your terminal:

heroku login

Step 2: Create a Heroku App

Create a new Heroku application by using the command: heroku create <app-name>

Replace <app-name> with your desired app name.

Step 3: Set Up Heroku Postgres

1. Log in to your Heroku account through your web browser.
2. Navigate to the newly created app's dashboard.
3. In the **Resources** tab, search for **Heroku Postgres** and add it to your application.

Step 4: Configure Environment Variables

1. In the **Settings** tab of the Heroku app, click on **Reveal Config Vars**.
2. Add the following environment variables:
    - **GOOGLE_CLIENT_ID**
      Value:
      765932404922-vvd8svdheovq1kf1u4hkrof6d0a4o04g.apps.googleusercontent.com

- **GOOGLE_CLIENT_SECRET**
  Value: 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

Step 5: Add Heroku Git Remote

To link your local repository to Heroku, run the following command:
git remote add heroku https://git.heroku.com/<app-name>.git.

Replace <app-name> with the name of your Heroku application.

Step 6: Modify omniauth.rb for Google OAuth2

To use environment variables for storing Google OAuth2 credentials, you need to modify the omniauth.rb configuration file:

1. Open the omniauth.rb file in your project.
2. Replace the following line:
   provider :google_oauth2, google_credentials[:client_id],
   google_credentials[:client_secret]

   With:
   provider :google_oauth2, ENV['GOOGLE_CLIENT_ID'],
   ENV['GOOGLE_CLIENT_SECRET']

3. Commit the changes and push them to Heroku:
   a. git add .
   b. git commit -m "google client details"
   c. git push heroku main

Step 7: Authorize the Application URL

- The app's URL must be authorized in the Google Credentials settings.
- Currently, you need to contact **saijaideepreddymure@tamu.edu** to authorize the app URL.
- If you want to deploy the app to your own Heroku account, create an account and restrict the domain to **tamu.edu**. Add the following URL in the **Authorized redirect URIs** section:
  https://<app-name>.herokuapp.com/auth/google_oauth2/callback

Step 8: Run Database Migrations

After deploying the code, you need to apply database migrations:

heroku run rails db:migrate

Troubleshooting:
If you encounter an error during migration, you may need to modify the migration file located at db/migrate/20241003222758_modify_users_table1.rb. Specifically, remove this line:
remove_column :users, :name, :string

Once you've made the change, commit and deploy the changes again:

- ➔ git add .
- ➔ git commit -m "fix migration issue"
- ➔ git push heroku main

Then, run the migration command again: heroku run rails db:migrate

Step 9: Add Users to the Application

To allow users to log in, you must create user records in the database.

1. Run the following command to open the Rails console on Heroku:
   heroku run rails console -a <app-name>
2. Create a new user by running the following command:
   User.create(email: "saijaideepreddymure@tamu.edu", name: "JaideepReddy", uin: "12345")

Once these steps are completed, your application should be successfully deployed on Heroku and accessible for use.

---

12. LINKS TO PROJECT RESOURCES

- Project Management Tool Page: https://github.com/users/phanijyothi11-tamu/projects
- Public GitHub Repository: https://github.com/phanijyothi11-tamu/team-formation
- Heroku Deployment: https://teamformation-c5eaebc1d53b.herokuapp.com

---

13. LINK TO DEMO & PRESENTATION VIDEOS

- https://youtu.be/OYswq9mM3zY
