TMU GradTrack

Final Report Document

Group 9

Sarah Hassan, ID: 500880917, Email: sarah.rose.hassan@torontomu.ca

Zain Zubair, ID: 501175960, Email: zain.zubair@torontomu.ca

Vicheka Oeun, ID: 501214846, Email: vicheka.oeun@torontomu.ca

Tahshin Shahriar, ID: 501105944, Email: tahshin.shariar@torontomu.ca

Department of Science, Toronto Metropolitan University

CPS 731: Software Engineering I

Professor: Dr. Sadaf Mustafiz

Dec 1, 2024

Table of Contents

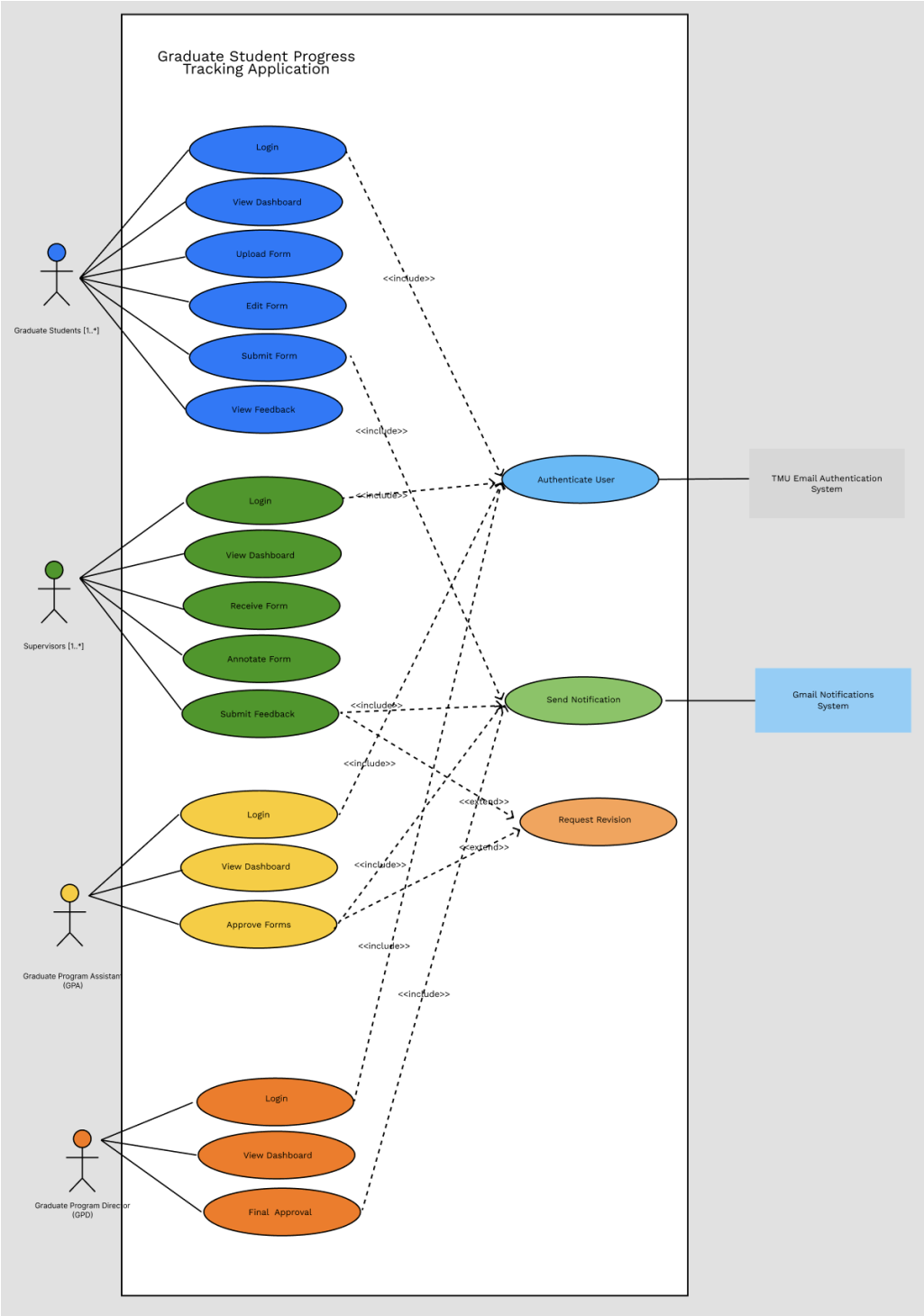
Problem Statement

Graduate students at TMU are required to submit a progress review and plan of study update form for each term in their program to their supervisors manually through email. The whole process is tedious and takes up a lot of wasted time as it requires a lot of back and forth emailing between graduate student(s), supervisor(s). Further complicating matters, these forms then require additional approvals from both the graduate program assistant (GPA) and graduate program director (GPD).

Our team's goal is to automate the submission & review process, eliminating manual labour and streamlining the tracking process through the ability to modify, annotate & track all forms involved with all parties.

Requirements Models

Use Case Model



Actors

Students:

- **Role:** A graduate student who submits progress forms.
- **Interactions:**
    - Uploads and submits progress forms to their supervisor.
    - Views feedback and approvals from the supervisor, Graduate Program Assistant (GPA), and Graduate Program Director (GPD).
    - Monitors the status of their submissions on the dashboard.

Supervisors:

- **Role:** A faculty member who reviews and provides feedback on student submissions.
- **Interactions:**
    - Reviews, annotates, and provides feedback on student submissions.
    - Submits feedback, which triggers a Gmail notification to the student.

Graduate Program Assistant (GPA):

- **Role:** A staff member responsible for the initial approval of student forms.
- **Interactions:**
    - Views submitted forms after the supervisor's feedback has been provided.
    - Approves the form, which moves it to the Graduate Program Director (GPD) for final approval.
    - A Gmail notification is sent to the student once the form is fully approved.
    - Each department or graduate program often has a Graduate Program Assistant (GPA) who handles administrative tasks and supports the program's operational needs, such as managing submissions and initial reviews for forms.

Graduate Program Director (GPD):

- **Role:** The academic director responsible for final approval of all student forms.
- **Interactions:**
    - Views submitted forms and feedback.
    - Gives final approval after the GPA's approval, triggering a Gmail notification to the student confirming the approval of the form.
    - Each graduate program typically has one Graduate Program Director (GPD) responsible for overseeing the academic aspects of the program. The GPD is generally a faculty member within that specific program, such as in the PhD programs for Management, Engineering, or Project Management, and they play a central role in approving forms and guiding students' academic progress.

External Systems

TMU Email Authentication System:

- **Role:** Authenticates the login credentials of users (Students, Supervisors, GPA, GPD).
- **Interactions:** Verifies credentials during login.

Gmail Notifications:

- **Role:** Sends email notifications when approvals or feedback are submitted.
- **Interactions:** Notifies students, supervisors, GPA, and GPD when approvals or feedback are made.

Use Case Descriptions

1. Login

- **Use Case Name**: Login
- **Scope**: Graduate Student Progress Tracking Application
- **Level**: User Goal
- **Intention in Context**: To allow users to securely access the system with TMU credentials
- **Multiplicity**: Each actor logs in individually
- **Primary Actor**: Student, Supervisor, GPA, GPD
- **Secondary Actor**: None
- **External System**: TMU Email Authentication System
- **Main Success Scenario**:
    - Actor enters TMU email and password.
    - System triggers **Authenticate User** to verify credentials with the **TMU Email Authentication System**.
    - Upon successful authentication, the actor gains access to their dashboard.
- **Extensions**:
    - Invalid login details: System displays an error message.

2. View Dashboard

- **Use Case Name**: View Dashboard
- **Scope**: Graduate Student Progress Tracking Application
- **Level**: User Goal
- **Intention in Context**: Provides an overview of form statuses, tasks, and notifications
- **Multiplicity**: Each actor views their dashboard individually
- **Primary Actor**: Student, Supervisor, GPA, GPD
- **Secondary Actor**: None
- **Main Success Scenario**:

    - Actor accesses the dashboard.
    - The system displays relevant information, such as form statuses and notifications, based on the actor's role.
- **Extensions**:
    - If no tasks or notifications are present, the dashboard remains empty.

3. Upload Form

- **Use Case Name**: Upload Form
- **Scope**: Graduate Student Progress Tracking Application
- **Level**: User Goal
- **Intention in Context**: Enables students to upload a form for review
- **Multiplicity**: Each form is uploaded individually
- **Primary Actor**: Student
- **Secondary Actor**: None
- **Main Success Scenario**:
    - Student selects "Upload Form."
    - System validates the format and content.
    - Form is saved as a draft in the system.
- **Extensions**:
    - Invalid file format: System requests a re-upload in the correct format.

4. Edit Form

- **Use Case Name**: Edit Form
- **Scope**: Graduate Student Progress Tracking Application
- **Level**: User Goal
- **Intention in Context**: Allows students to modify a saved form
- **Multiplicity**: Each form is edited individually
- **Primary Actor**: Student
- **Secondary Actor**: None
- **Main Success Scenario**:
    - Student selects the form to edit.
    - System enables editing options for form fields.
    - Student makes changes and saves the form.
- **Extensions**:
    - Save Failure: System displays an error and prompts the student to retry.

5. Submit Form

- **Use Case Name**: Submit Form
- **Scope**: Graduate Student Progress Tracking Application
- **Level**: User Goal
- **Intention in Context**: Allows students to submit their completed form for review
- **Multiplicity**: Each form is submitted individually
- **Primary Actor**: Student
- **Secondary Actor**: None
- **External System**: Gmail Notification System
- **Main Success Scenario**:
    - Student selects "Submit Form."
    - The system updates the form status to "Submitted."
    - System triggers **Send Notification** to inform the Supervisor via the **Gmail Notification System**.
- **Extensions**:
    - Incomplete form: System alerts the student about missing required fields.

6. View Feedback

- **Use Case Name**: View Feedback
- **Scope**: Graduate Student Progress Tracking Application
- **Level**: User Goal
- **Intention in Context**: Allows students to view feedback on their submitted forms
- **Multiplicity**: Each feedback is viewed individually
- **Primary Actor**: Student
- **Secondary Actor**: None
- **Main Success Scenario**:
    - Student opens the form to view feedback.
    - System displays feedback, annotations, and comments from the Supervisor.

7. Receive Form

- **Use Case Name**: Receive Form
- **Scope**: Graduate Student Progress Tracking Application
- **Level**: User Goal
- **Intention in Context**: Allows Supervisors to view submitted forms from students
- **Multiplicity**: Each form is received individually
- **Primary Actor**: Supervisor
- **Secondary Actor**: None
- **Main Success Scenario**:
    - Supervisor receives a notification of a new submission.
    - Supervisor logs in and views the form submitted by the student.

- **Extensions**:
    - If the form is unavailable, the system prompts the Supervisor to retry.

8. Annotate Form

- **Use Case Name**: Annotate Form
- **Scope**: Graduate Student Progress Tracking Application
- **Level**: User Goal
- **Intention in Context**: Allows Supervisors to add feedback to student submissions
- **Multiplicity**: Each form is annotated individually
- **Primary Actor**: Supervisor
- **Secondary Actor**: None
- **Main Success Scenario**:
    - Supervisor opens the form and adds comments.
    - Annotations are saved and displayed to the student.
- **Extensions**:
    - Save Failure: System prompts the Supervisor to retry saving.

9. Submit Feedback

- **Use Case Name**: Submit Feedback
- **Scope**: Graduate Student Progress Tracking Application
- **Level**: User Goal
- **Intention in Context**: Finalizes and submits feedback for student review
- **Multiplicity**: Each form's feedback is submitted individually
- **Primary Actor**: Supervisor
- **Secondary Actor**: None
- **External System**: Gmail Notification System
- **Main Success Scenario**:
    - Supervisor selects "Submit Feedback."
    - System finalizes the feedback and marks it as submitted.
    - **Send Notification** triggers to notify the student of feedback availability via the **Gmail Notification System**.

10. Approve Form

- **Use Case Name**: Approve Form
- **Scope**: Graduate Student Progress Tracking Application
- **Level**: User Goal
- **Intention in Context**: Allows GPA to approve forms after supervisor review

- **Multiplicity**: Each form is approved individually
- **Primary Actor**: Graduate Program Assistant (GPA)
- **Secondary Actor**: None
- **External System**: Gmail Notification System
- **Main Success Scenario**:
    - GPA reviews the form after supervisor feedback.
    - GPA approves the form, triggering a notification to the GPD via the **Gmail Notification System** for final approval.

11. Final Approval

- **Use Case Name**: Final Approval
- **Scope**: Graduate Student Progress Tracking Application
- **Level**: User Goal
- **Intention in Context**: Allows the GPD to grant final approval on student submissions
- **Multiplicity**: Each form is approved individually
- **Primary Actor**: GPD
- **Secondary Actor**: None
- **External System**: Gmail Notification System
- **Main Success Scenario**:
    - GPD reviews the forwarded form and provides final approval.
    - **Send Notification** informs the student of approval via the **Gmail Notification System**.
- **Extensions**:
    - Rejection: If the GPD rejects the form, the system triggers a notification for the student to make revisions.

Supporting Use Cases

Authenticate User

- **Use Case Name**: Authenticate User
- **Scope**: Graduate Student Progress Tracking Application
- **Level**: Subfunction
- **Intention in Context**: Validates user credentials during login
- **Multiplicity**: Each login instance is authenticated individually
- **Primary Actor**: System
- **Secondary Actor**: None
- **External System**: TMU Email Authentication System
- **Main Success Scenario**:
    - System sends credentials to **TMU Email Authentication System**.

    - Authentication succeeds or fails, determining access.

Send Notification

- **Use Case Name**: Send Notification
- **Scope**: Graduate Student Progress Tracking Application
- **Level**: Subfunction
- **Intention in Context**: Sends notifications triggered by actions
- **Multiplicity**: Each action triggers individual notifications
- **Primary Actor**: System
- **Secondary Actor**: None
- **External System**: Gmail Notification System
- **Main Success Scenario**:
    - System sends notification details to **Gmail Notification System**.
    - The recipient receives a notification regarding the update.

Request Revision

- **Use Case Name**: Request Revision
- **Scope**: Graduate Student Progress Tracking Application
- **Level**: Optional Subfunction
- **Intention in Context**: Requests additional changes when feedback or approvals require it
- **Multiplicity**: Each form can be requested for revision individually
- **Primary Actor**: System
- **Secondary Actor**: None
- **External System**: Gmail Notification System
- **Main Success Scenario**:
    - System notifies the student of necessary revisions.
    - The form is marked as "Revisions Required."

Assumptions and Constraints

Assumptions:

- All students, supervisors, GPAs and GPDs have valid TMU email accounts for authentication.
- Students will have access to the required devices and internet connection to use the system.

- Supervisors are familiar with providing feedback digitally.

Constraints:

- The system must integrate with TMU Email Authentication for secure login.
- Notifications must be sent via Gmail
- The system must comply with TMU's data privacy and security policies.

Relationships and Dependencies

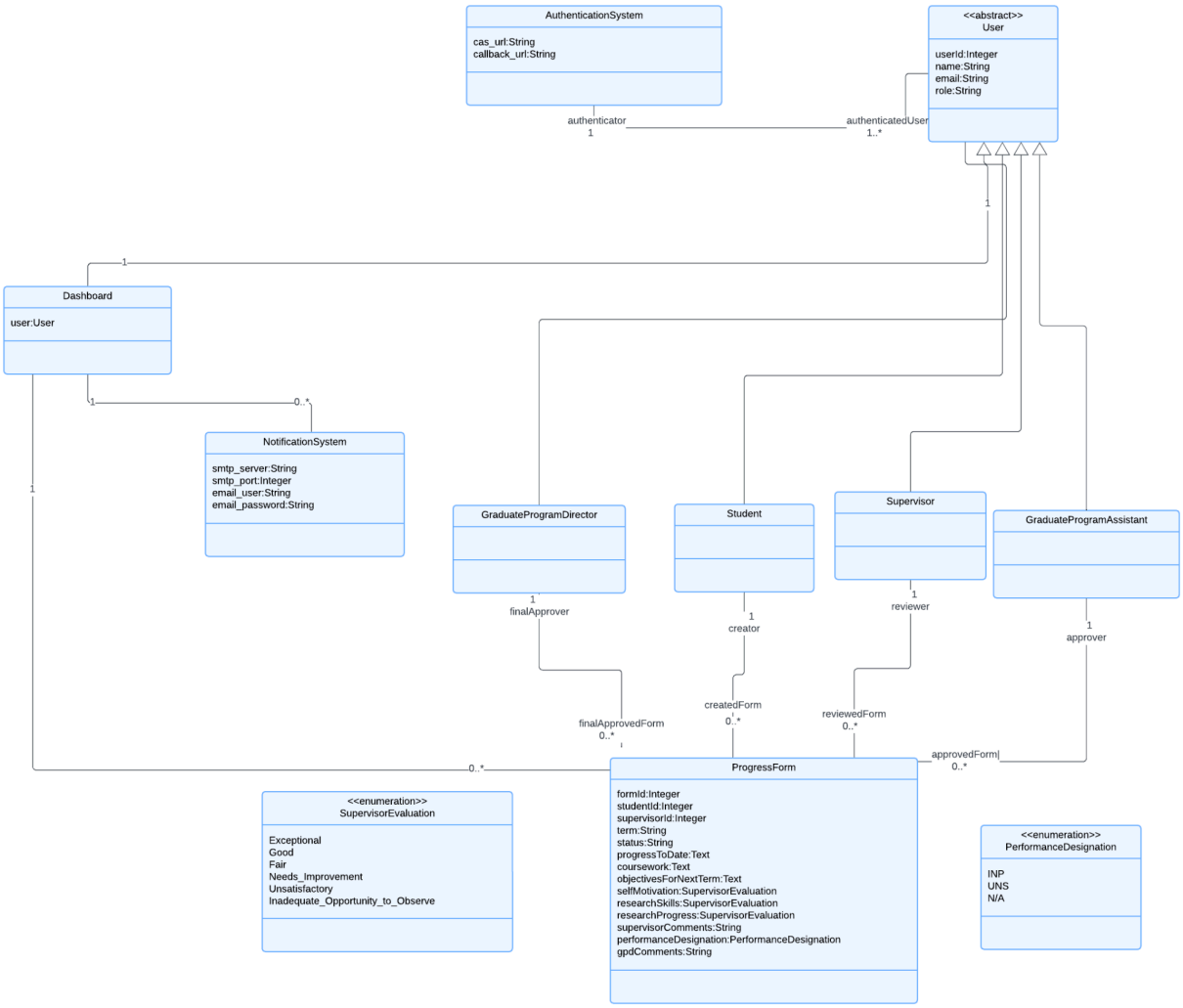
Relationships

- **<<include>> Relationships**:
    - Authenticate User is an included use case in Login for all user types (Graduate Students, Supervisors, GPA, GPD). This ensures that each login attempt requires authentication, handled by the TMU Email Authentication System.
    - Send Notification is included in Submit Form, Submit Feedback, Approve Form, and Final Approval to automatically trigger notifications via the Gmail Notification System whenever a significant action occurs in the workflow.
- **<<extend>> Relationships**:
    - Request Revision is an extended use case that may be triggered from Submit Feedback or Approve Form if revisions are required before further progress. This optional relationship helps streamline the process when corrections are needed, ensuring a clear path for feedback loops.

Dependencies

- **Resend Gmail Notification System**:
    - The system depends on the Resend Notification System to send email notifications to relevant parties (e.g., Supervisors, Students, GPA, GPD) based on actions taken within the application, such as form submissions and approvals. This dependency supports efficient communication and tracking of progress.

Domain Model



Design Models

Architectural Design Model

1. Choice of Architectural Pattern:

For this project, we have chosen the **Model-View-Controller (MVC)** combined with the **Client-Server architecture**. This hybrid model ensures a clear separation of frontend and backend responsibilities, making it well-suited for applications with multiple user roles and secure data handling, such as our Graduate Student Progress Tracking Application.

2. Justification:

The hybrid MVC and Client-Server architecture is ideal for our project, meeting both functional and non-functional requirements while supporting secure, scalable, and modular development. The MVC pattern enables tailored views for each user type (students, supervisors, GPA, GPD), while the Client-Server model enforces a clear separation of frontend (View/Controller) and backend (Model) operations. This approach allows the client (React + Next.js) to handle user interactions and routing, while the server (Supabase) securely manages data and business logic.

3. Key Components of the MVC + Client-Server Design:

Client Section

View (React Components):

- Displays tailored interfaces for each user role, showing relevant data and actions.
- Dynamically updates based on responses received from the server through API calls.

Controller (Next.js):

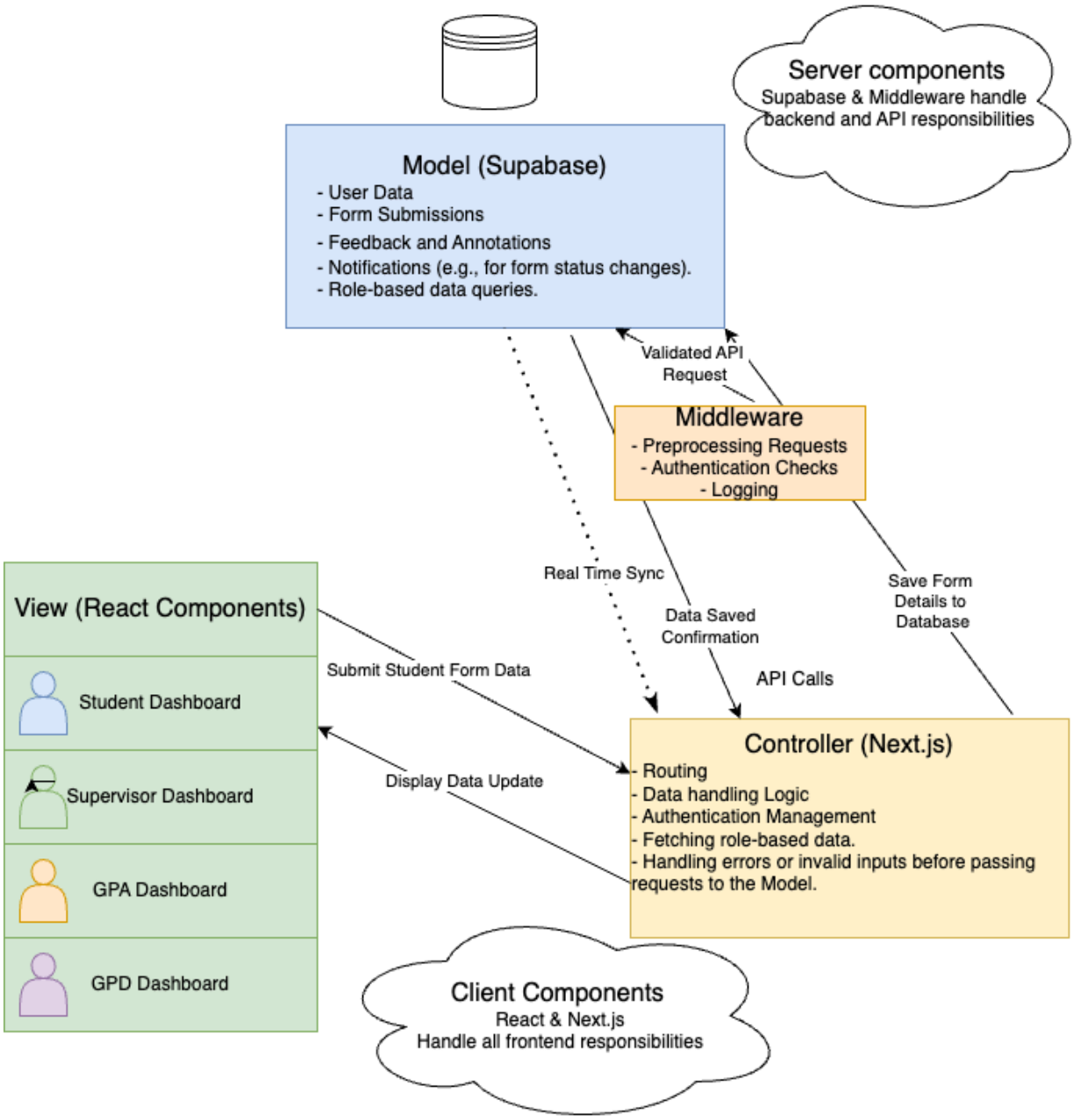
- Acts as the bridge between the client-side View and the backend Model, processing user inputs and making API requests.
- Handles routing and client-side logic, ensuring the correct data flows between the

Server Section:

Model (Supabase):

- Manages all backend operations, including data storage, retrieval, and role-based access for users (students, supervisors, GPA, GPD).
- Provides APIs to handle data interactions with the Controller, ensuring secure and synchronized updates.View and the Model.

4. Architectural Design Diagram:



https://drive.google.com/file/d/1Joa37KZISA855osUwC6LZUFjpzeKH0sq/view?usp=sharing

Example
For a form submission workflow:

<u>Client Section:</u>
View (React):

- User fills out a form and clicks "Submit."

    - Sends the data to the Controller.

Controller (Next.js):

- Validates the data and sends an API request to the Server.

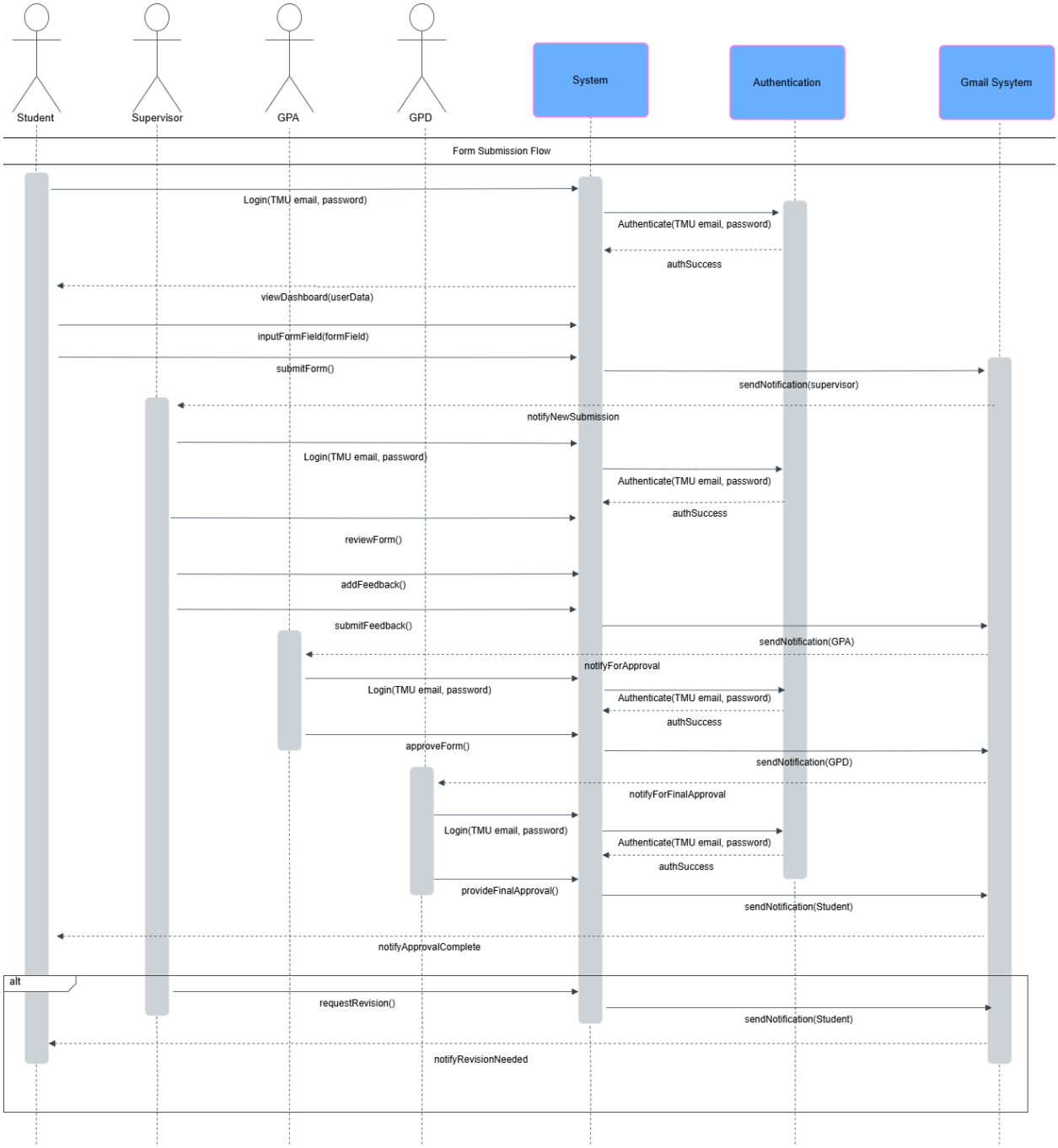
Server Section:

- Model (Supabase):
- Processes the request, saves the data, and returns a success response.

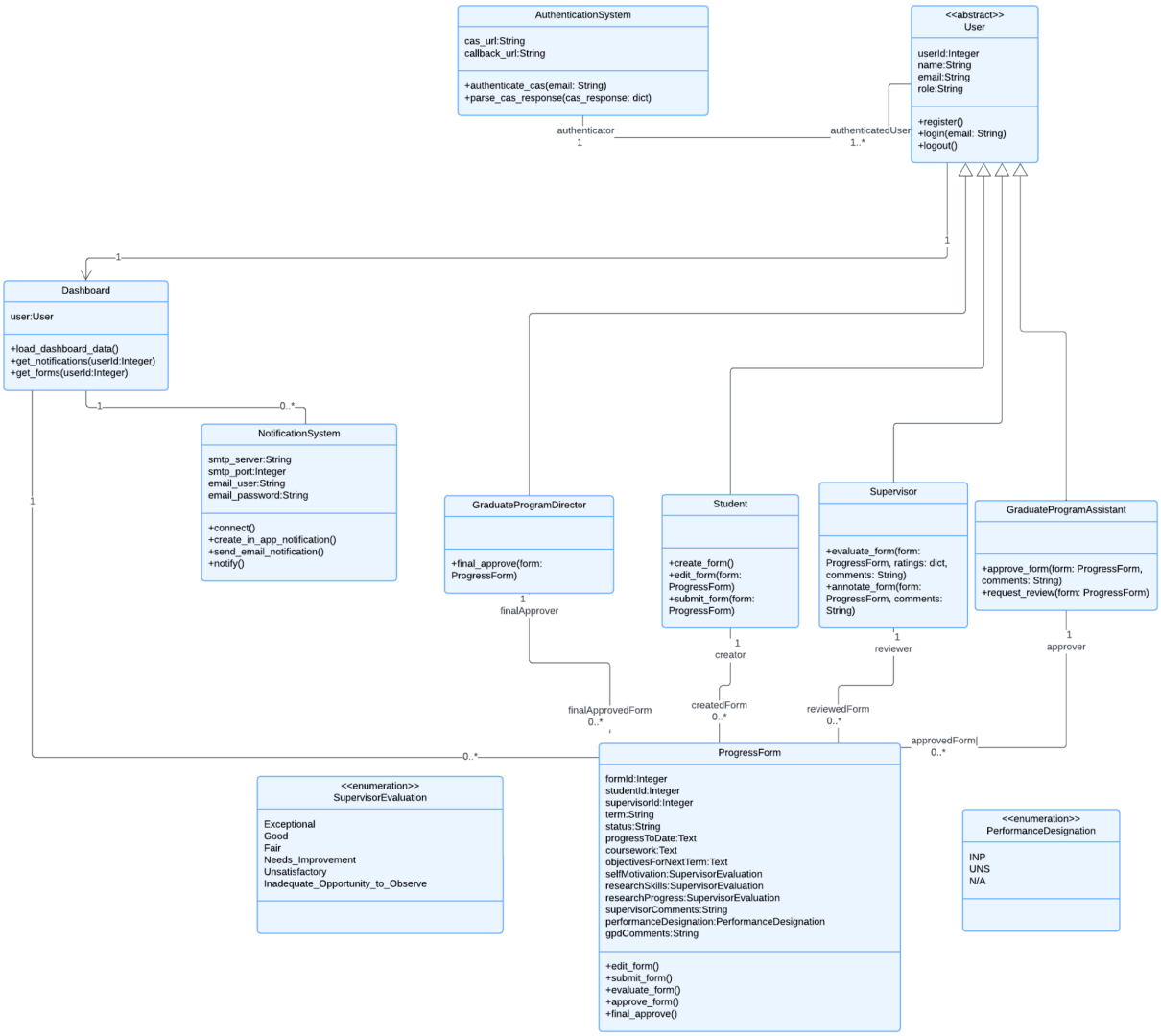
Middleware:

- Preprocesses the API request before passing it to the Model.

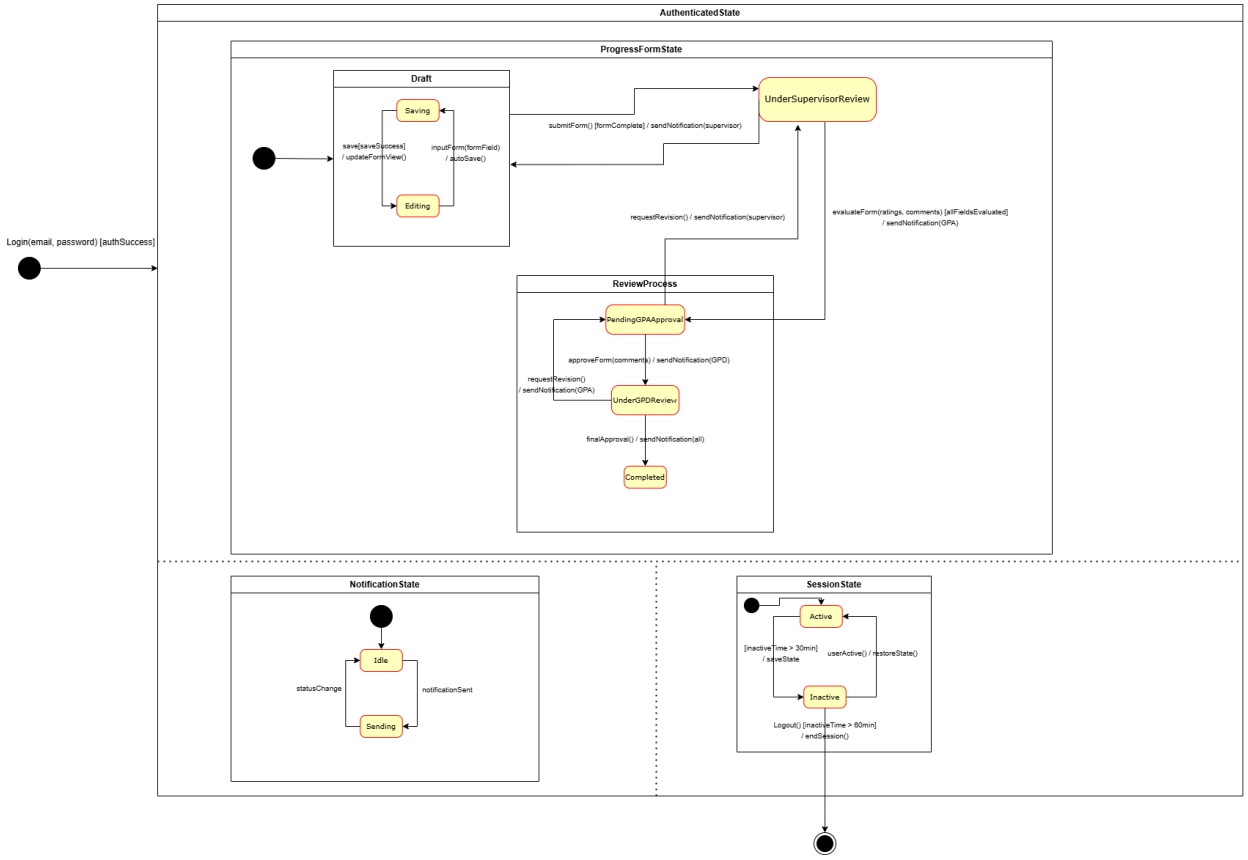
Interaction Model (UML Sequence Diagram)



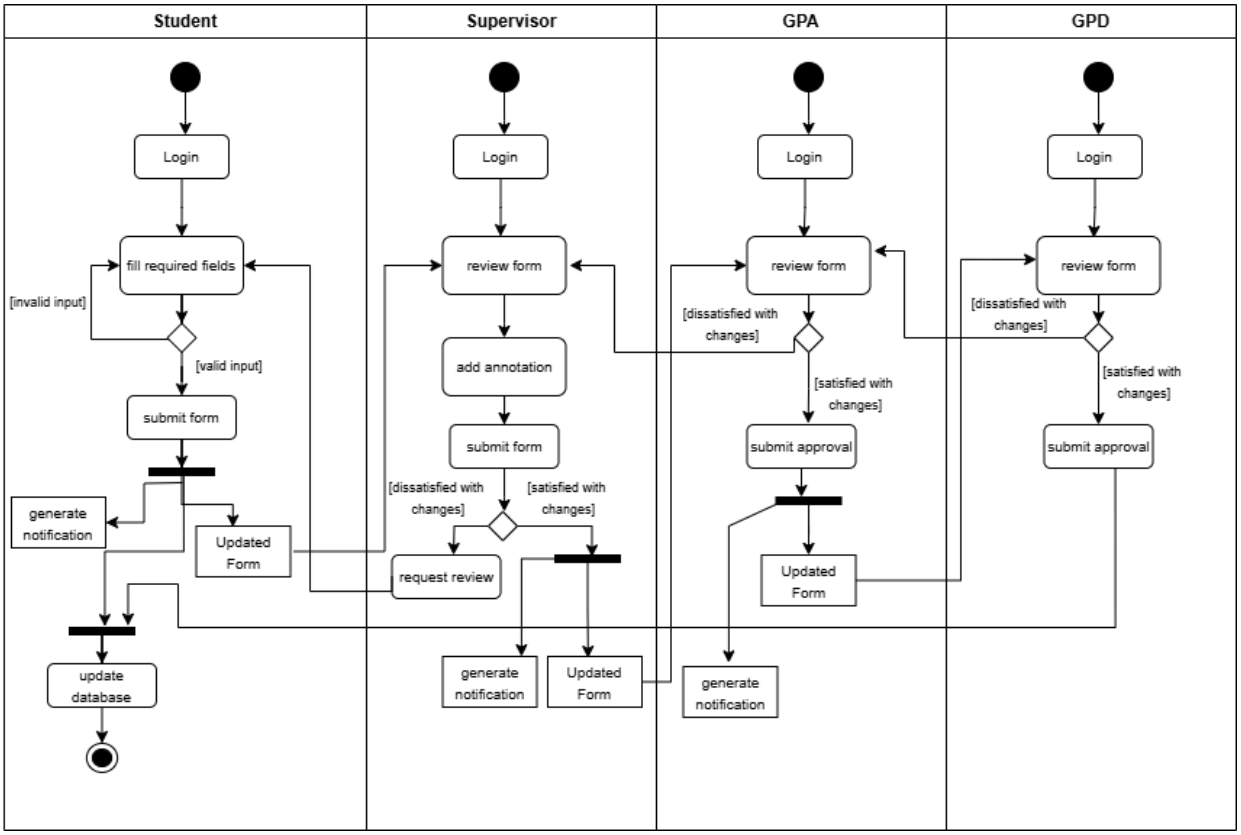
Design Class Model



State Machine Model



Activity Diagram



Test Cases (Revised)

Test Case Title	Steps	Example Input	Expected Result
Test Case Section			
1. User Authentication			
1.1 Valid TMU email and password login.	1. Navigate to login page. 2. Enter valid TMU email and password. 3. Click “Login.”	Correct email and password, example: zain.zubair@torontomu.ca validPasword123	User successfully logs in and accesses their dashboard.
1.2 Invalid TMU email or password login	1. Navigate to login page. 2. Enter invalid TMU email or password. 3. Click “Login.”	Incorrect email or password, example: zain.zubair@torontomu.ca incorrectPass	System displays an error message and denies access.

1.3 Unauthorized role access	1. Navigate to login page. 2. Enter non-authorized role email (e.g., non-student/staff). 3. Click “Login.”	Non-student or non-staff email, example: zain.zubair@tmu.ca	System denies access and displays an unauthorized access error.
2. Dashboard Access and Navigation			
2.1 Access dashboard with valid role	1. Login as student, supervisor, GPA, or GPD. 2. Navigate to dashboard.	Valid role login, example: Student role, GPA role, GPD role	Dashboard displays role-specific tasks and notifications.
2.2 No tasks or notifications available	1. Login and navigate to dashboard. 2. Confirm no tasks or notifications are present.	View dashboard, example: Click dashboard link	Dashboard displays empty notifications with a message.
3. Student Form Management			
3.1 Form submission to correct supervisor	1. Navigate down to reach progress form 2. Input valid fields for form 3. Click “Submit Form”	Valid inputs: Term: “Fall 2024”, Start Term: 1, Coursework: “CPS731”…etc.	Form would display error on screen when input is wrong or forgotten to input to required form.
3.2 View previous submitted forms	1. Navigate to previous submitted form section 2. Click on “view feedback” 3. Click “Submit Form”	When clicking view feedback, redirect to the feedback form page. Able to save & re-submit form	System returns error message on display if form resubmission is wrong or saving locally doesn’t work.
4. Supervisor Review and Feedback			
4.1 Supervisor receives form submission notification	1. Supervisor receives email notification indicating form submission.	Notification triggered by student form submission.	Supervisor email would have a new notification by Resend.
4.2 Supervisor receives new form from student and is able to view it	1. Open the form to review. 2. Add annotations and feedback in designated fields.	Annotated text, example: “Excellent work.”	Annotations save successfully, and student is notified.

4.3 Supervisor approves and sends to GPA	1. Click “Send to GPA”	Annotated form	GPA would receive it their pending review dashboard
5. GPA and GPD Approval Process			
5.1 GPA views and approves supervisor-reviewed form	1. GPA logs in and selects a form for review. 2. Reviews and approves the form.	GPA access form and clicks “Approve” button	GPA can view, approve, or request revisions.
5.2 GPD final approval on GPA-approved form	1. GPD logs in and selects form for review. 2. Approves form.	GPD access form and clicks “Approve” button	Final approval updates status of form on all Student, Supervisor and GPA dashboard
5.3 Reject form and request revisions	1. GPA or GPD selects a form to review. 2. Click “Request Revisions” with feedback details.	Access form and click “Review” button	System prompts revisions and sends to Supervisor form
6. Notification System			
6.1 Notification sent upon form submission	1. Student completes and submits form. 2. System triggers notification to supervisor.	Student clicks “Submit” button on form page	Supervisor receives notification.
6.3 Notification delay or failure	1. Simulate notification trigger during service outage.	Error flag returned from notification function	System retries at intervals, and if unsuccessful, logs an error.
7. Error Handling and User Feedback			
7.1 Incorrect form field data input	1. Enter incorrect data types in form fields (e.g., text in numeric fields). 2. Submit form.	Incorrect data in form fields, example: “Name” entered for student ID	System shows field-specific error messages.
7.2 Failed save during form upload	1. Attempt to upload form with network interruption.	Network interruption, example:	System prompts user to retry upload.

		Connection disconnects	
7.3 Exceeding character limits in form fields	1. Enter text that exceeds character limits in fields. 2. Attempt to save or submit form.	Data exceeding character limits, example: “Long message” for a text field of 4 characters	System shows error message indicating character limit.
8. Data Security and Privacy			
8.1 Encrypted data transfer verification	1. Submit form and monitor encryption status during transfer.	User clicks “Submit” button	Data remains encrypted during transfer.
8.2 Access audit logging	1. Log in and access sensitive data. 2. Check audit log for access record.	User clicks “Submit” button then user opens audit log page	System logs access in audit trail.
9. Performance and Load Handling			
9.1 Form upload response time	1. Select and upload a valid form with varying network conditions. 2. View logs to check performance	User clicks “Submit” form	System processes upload within 2000ms.
9.2 High volume of concurrent notifications	1. Simulate multiple form submissions to trigger notifications. 2. Check if notifications were successfully sent via logs	User submits multiple forms	System sends notifications without delay.
9.3 Application load under peak usage	1. Simulate high usage with concurrent logins and form actions. 2. Analyze performance	Test using looping structure to simulate high traffic	Application maintains responsiveness within specified limits.
End Test Cases			

Design and Implementation Decisions

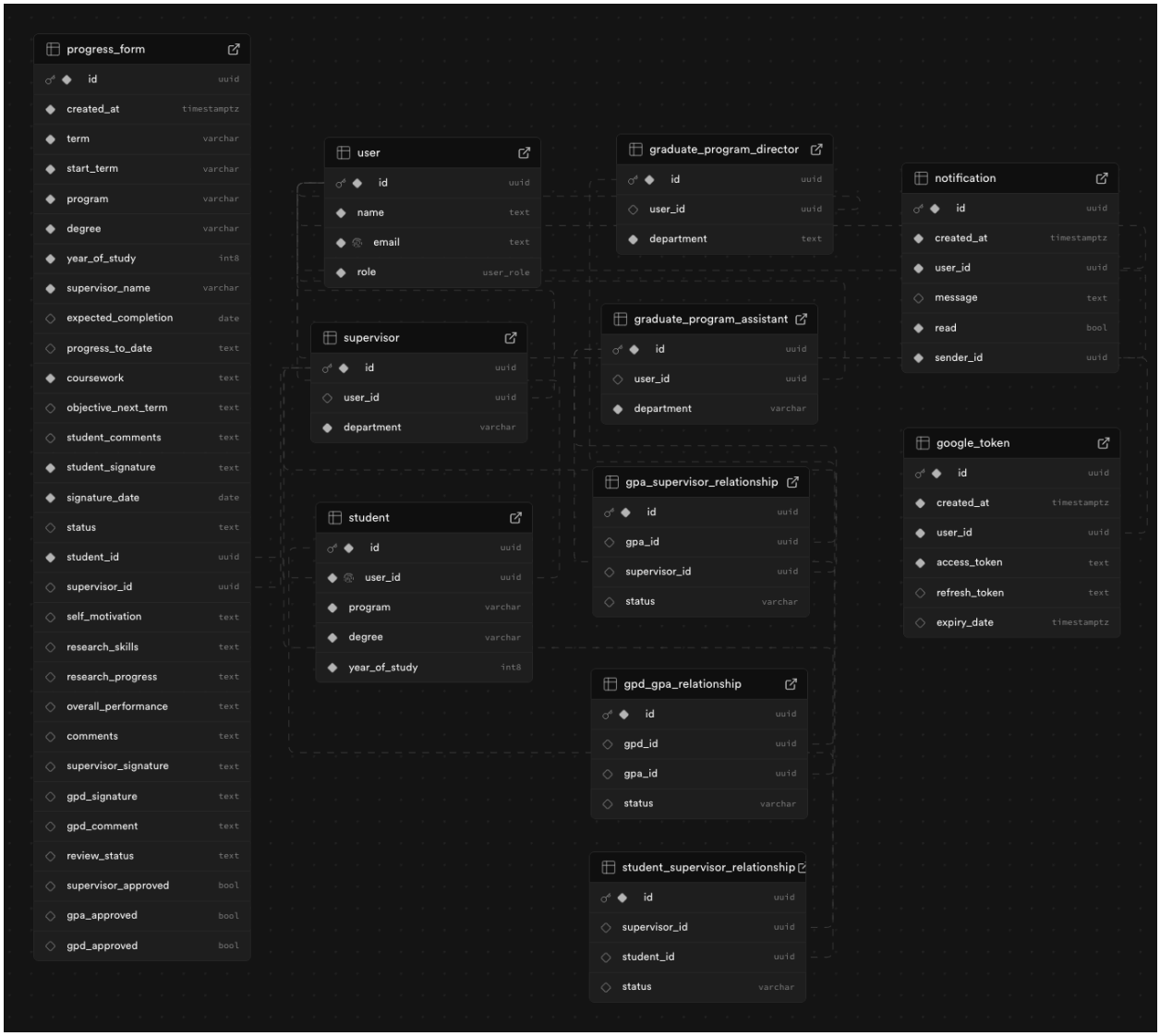
Frontend Framework

We used **React** with **Next.js** for its ease of use, efficient component-based development, and server-side rendering to improve performance.

Backend and Database

Database

The database we decided to use is supabase, which is a serverless database built on top of PostgreSQL. The decision behind this choice was its built-in authentication and serverless storage, along with its compatibility with Next.js (specifically Next.js 13) that is used as our main front-end framework.
We began by designing our schema like so:



(photo from supabase gui for database schema)

Our schema design decisions:

We can break down the schema into 3 components:

- user - includes: a main user then child users like student, supervisor, gpa & gpd
- user relationships - includes: the relationship between student and supervisor, supervisor and gpa & gpa and gpd
- progress form - includes the bulk our schema, the progress form
- notification system - includes: the notification schema along with a google token schema that is used to authenticate the notification, which we used google's gmail api for

Why we chose to have a separate relationship schema:

We chose to have separate relationship schemas because each user can have relationships with many other users (example: a supervisor can supervise multiple students and so on). By having a separate relationship schema for each relationship, we are able to create these

one-to-many relationships. This also has another advantage in the fact that we don't have to create a foreign key for each user, but instead a foreign in each relationship.

Authentication

Our authentication system was built using supabase's built-in authentication. The authentication system is complex and is built with 3 specific components interacting with one another.

- **Supabase Client** - supabase client acts as an entry point for our services to interact with the rest of supabase's functionality. We start by creating a supabase client which requires our supabase api key and a supabase public url. The client can then be called with its different methods that we used: signUp for signing up with a custom account, signIn for signing in with the custom account or OAuth which is a third-party provider for supabase to authenticate into say a Google account.

  Client:

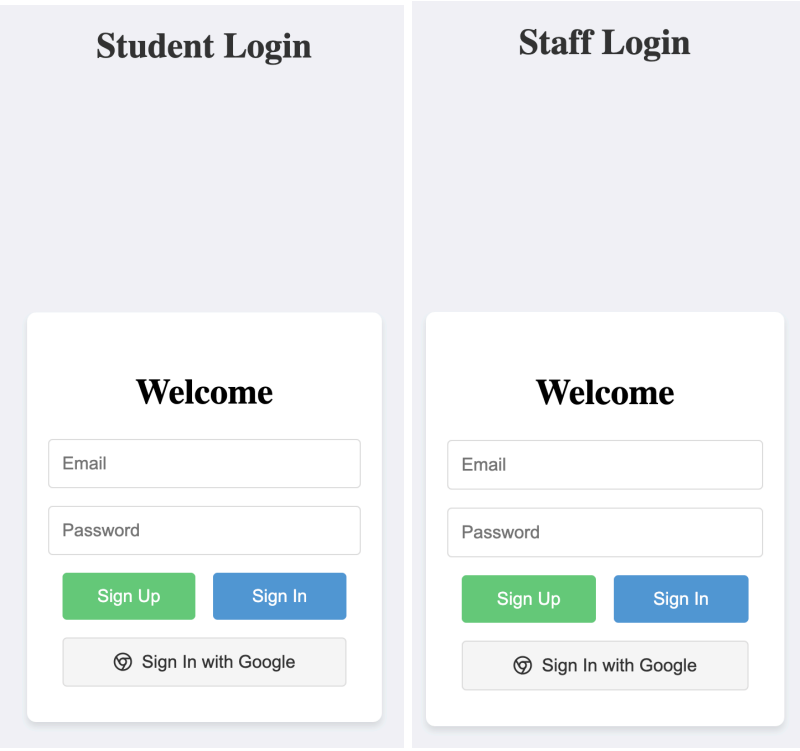
  ```
  import { createClient } from '@supabase/supabase-js'

  export const supabase = createClient(
    process.env.NEXT_PUBLIC_SUPABASE_URL,
    process.env.NEXT_PUBLIC_SUPABASE_ANON_KEY
  )
  ```

  Functions using supabase's authentication methods:

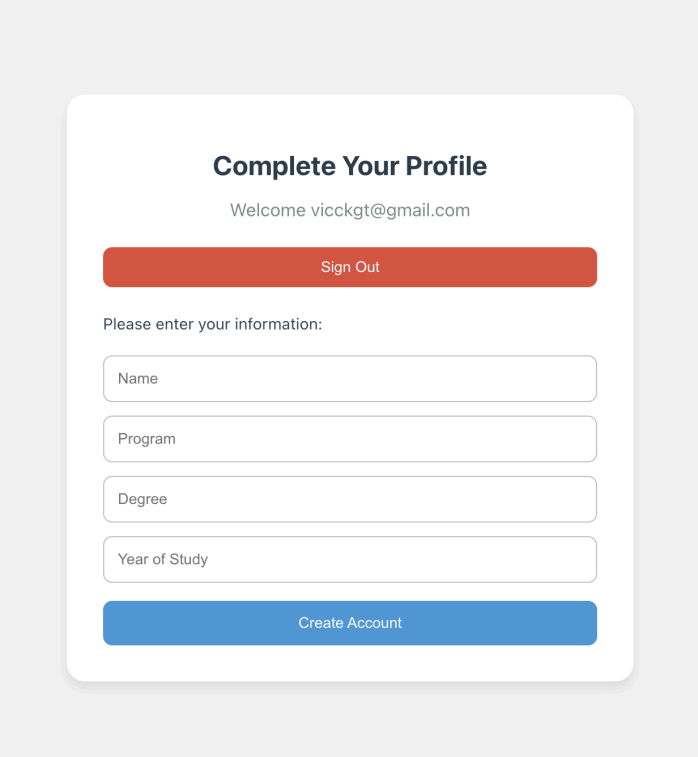
```
async function handleSignUp(event) {
  event.preventDefault()
  const { data, error } = await supabase.auth.signUp({ email, password })
  if (error) {
    setMessage(`Error signing up: ${error.message}`)
  } else {
    setMessage('Sign Up Successful!')
    router.push(role === 'staff' ? '/dashboard_staff' : '/dashboard')
  }
}

async function handleSignIn(event) {
  event.preventDefault()
  const { data, error } = await supabase.auth.signInWithPassword({
    email,
    password,
  })
  if (error) {
    setMessage(`Error signing in: ${error.message}`)
  } else {
    setMessage('Signed in successfully!')
    router.push(role === 'staff' ? '/dashboard_staff' : '/dashboard')
  }
}

async function handleSignInWithOAuth(event) {
  event.preventDefault()
  const { data, error } = await supabase.auth.signInWithOAuth({
    provider: 'google',
    options: {
      redirectTo: role === 'staff'
        ? `${window.location.origin}/dashboard_staff`
        : `${window.location.origin}/dashboard`,
    },
  })
  if (error) {
    setMessage(`Error signing in : ${error.message}`)
  }
}
```

- **API** - our api plays an important role in this setup, we created an auth endpoint, which listens to our request sent from our front-end application. Specifically our most important endpoint, the user endpoint which is used to create an account. The way the application works is, users are first authenticated into the app then they must provide our system with the information to create an account.

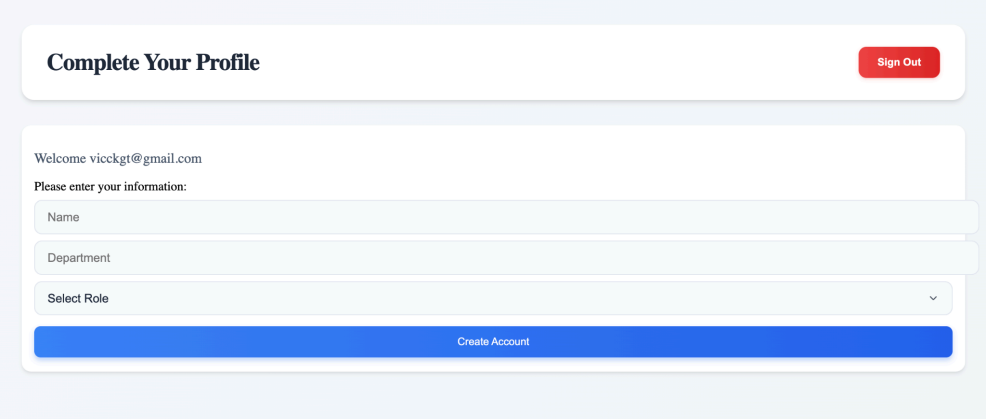
There are two clients in play here, a student and a staff.



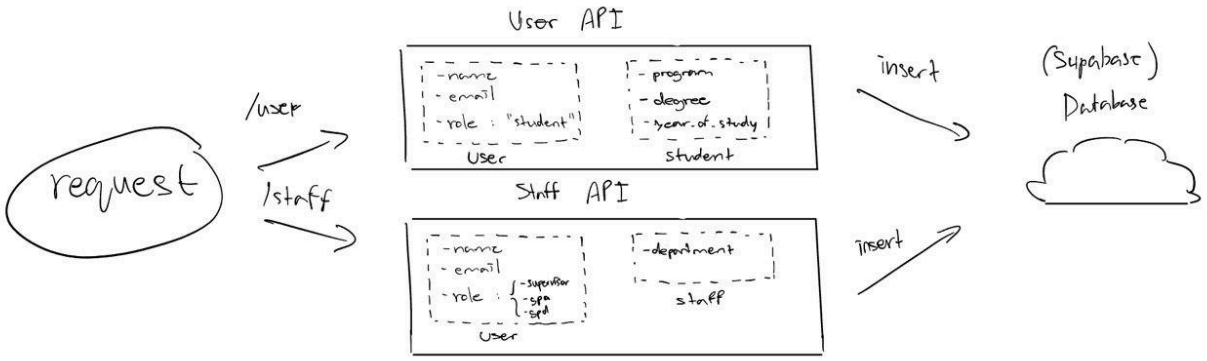
1. When a user signs in as a student, then they have to provide our application with credentials to create a student account.
    a. Student:



b. Staff:



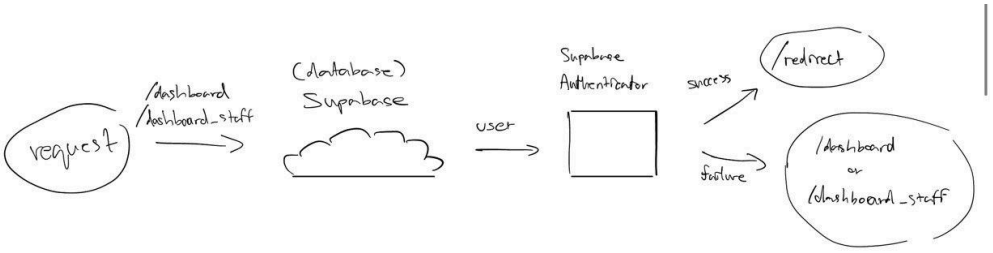
Then when a user account is created either as a student or as a staff member, it hits one of our endpoints (student or staff).



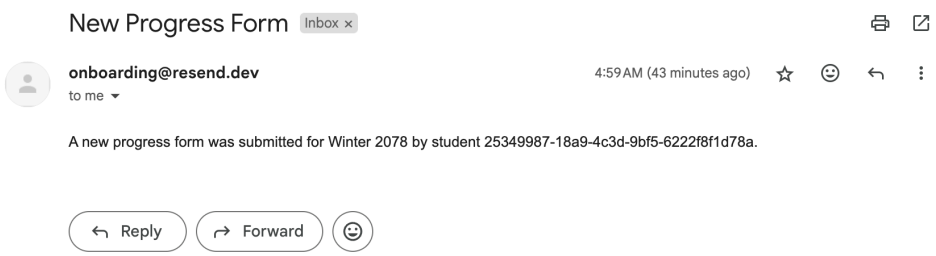
- **student endpoint**: when it hits our user api (generic for student), it sends an insert query to our supabase database to create a new user. Then send another insert query (using the id user as the foreign key) to create a new student.
- **staff endpoint**: when it hits our staff api, it sends an insert query to our database to create a new user, then sends another insert query to create a new staff member (based on the role as either a supervisor or gpa or gpd).

- **Database** - the database is not only relevant for creating a new user, but also used as part of the authentication process.

    - We have a route /dashboard which is where users are redirected to (either staff dashboard or generic student dashboard).
        - When a user requests to access the dashboard page, a request is sent to supabase's authenticator and checks if a user is authenticated.

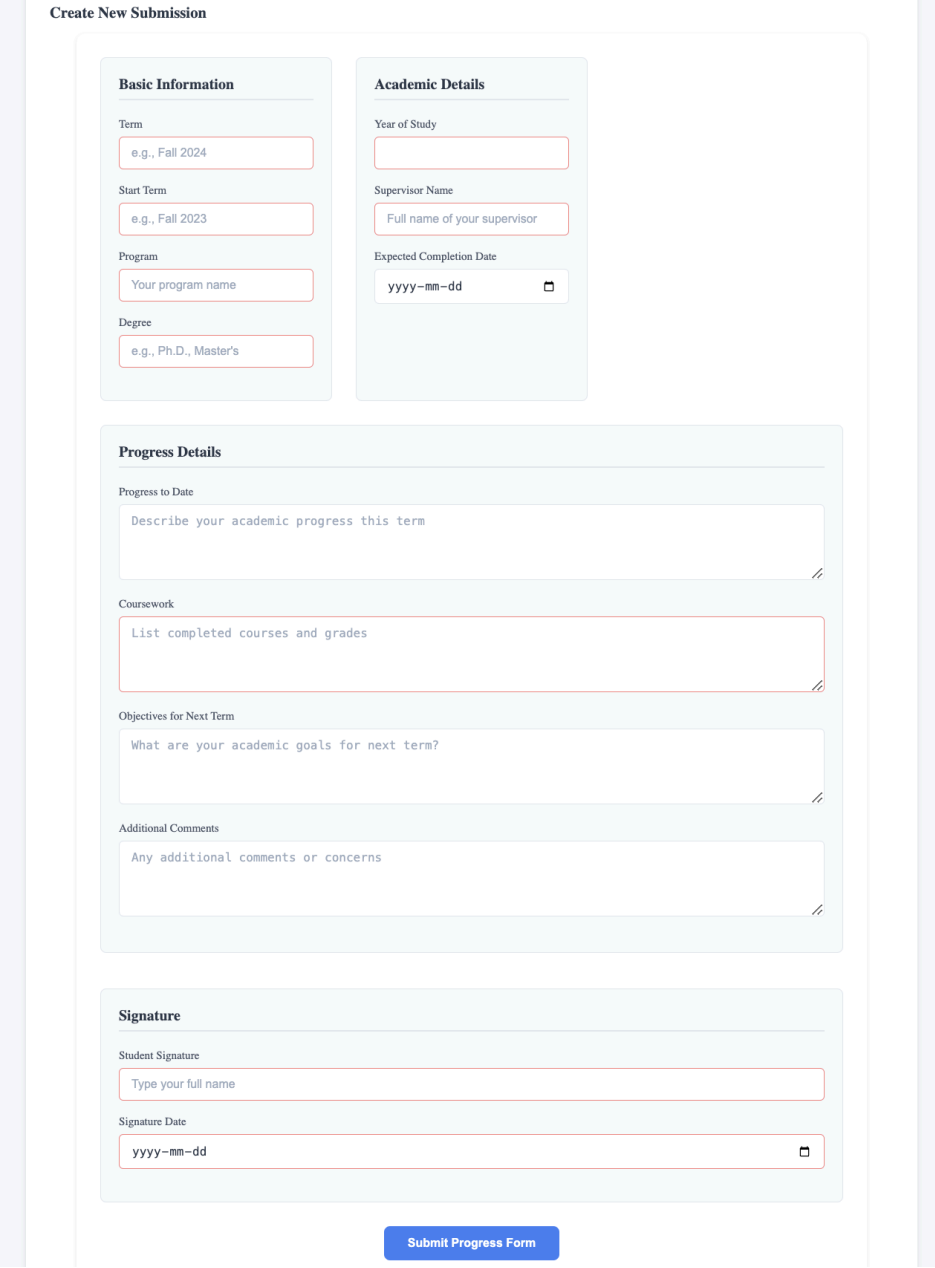


        - Here's a sample message:



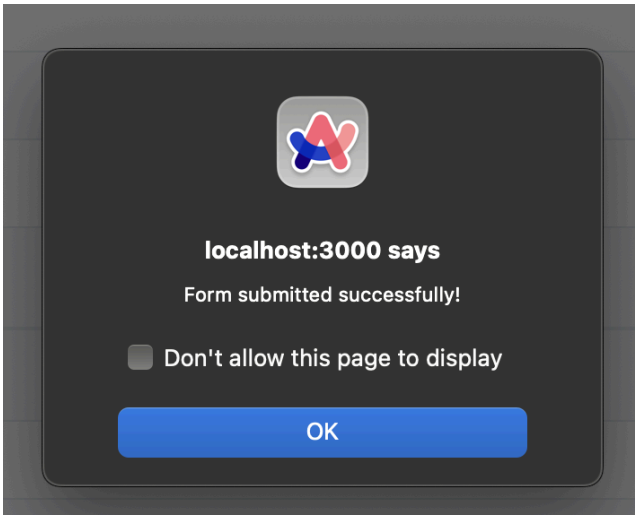
Student Form & Submission

1. Initially, a student would request to access their student dashboard, once authorized (via Supabase) the dashboard would automatically request for any previous form submitted by the student through a request to the database. Then the student's dashboard would be displayed.
2. When a student submits a form, they must first fill the necessary form details via an HTML input form. These information include:



3. Once a student properly submits, the form would send a POST request to Supbase's database. Supabase would then check if the student is assigned to a supervisor via the student_supervisor relationship table.

a. If yes, the form is submitted to said table and a window's alert would pop up displaying the success message. Then an email notification would be sent out to the supervisor.

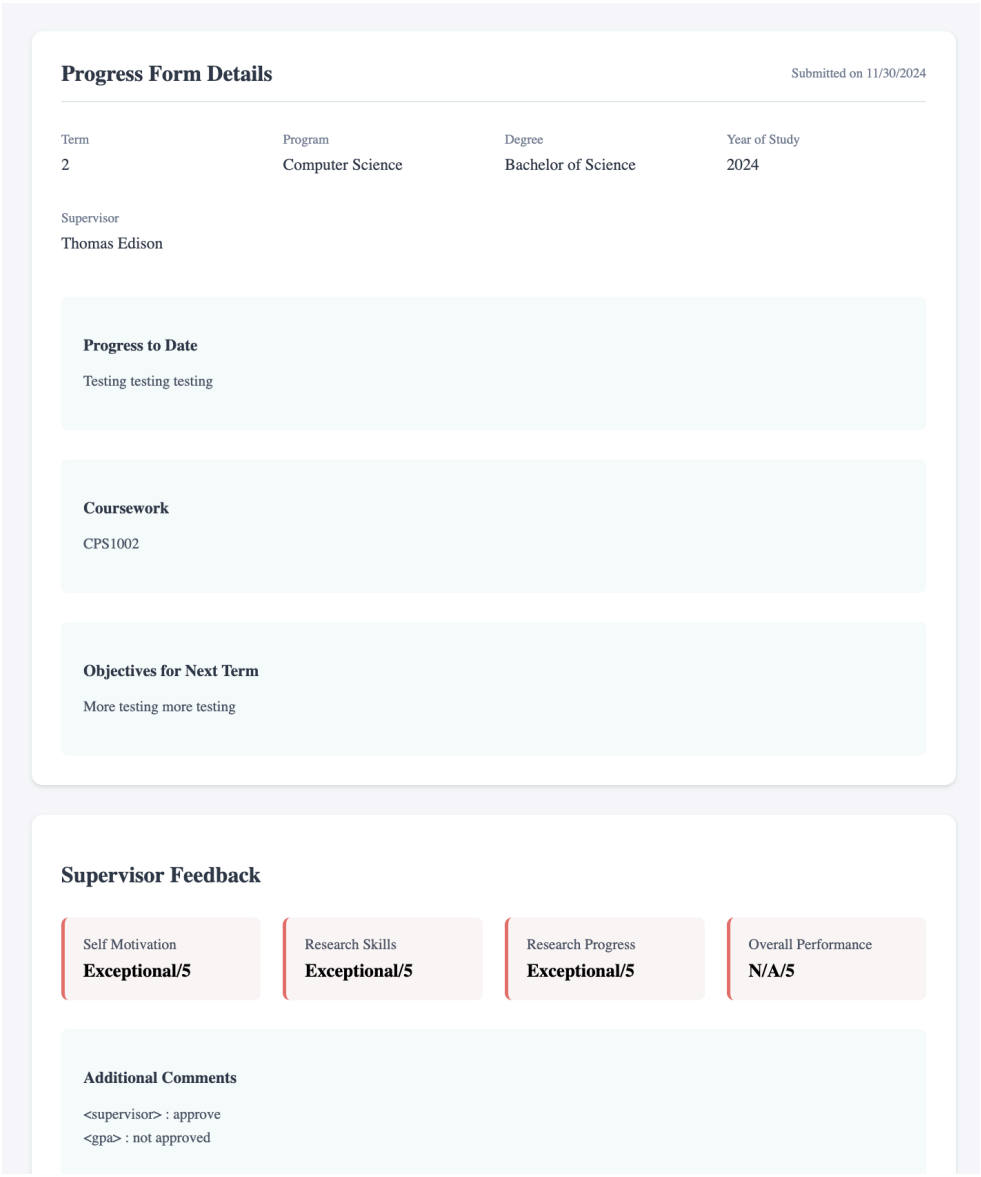


Progress form table in database would be populated like so:

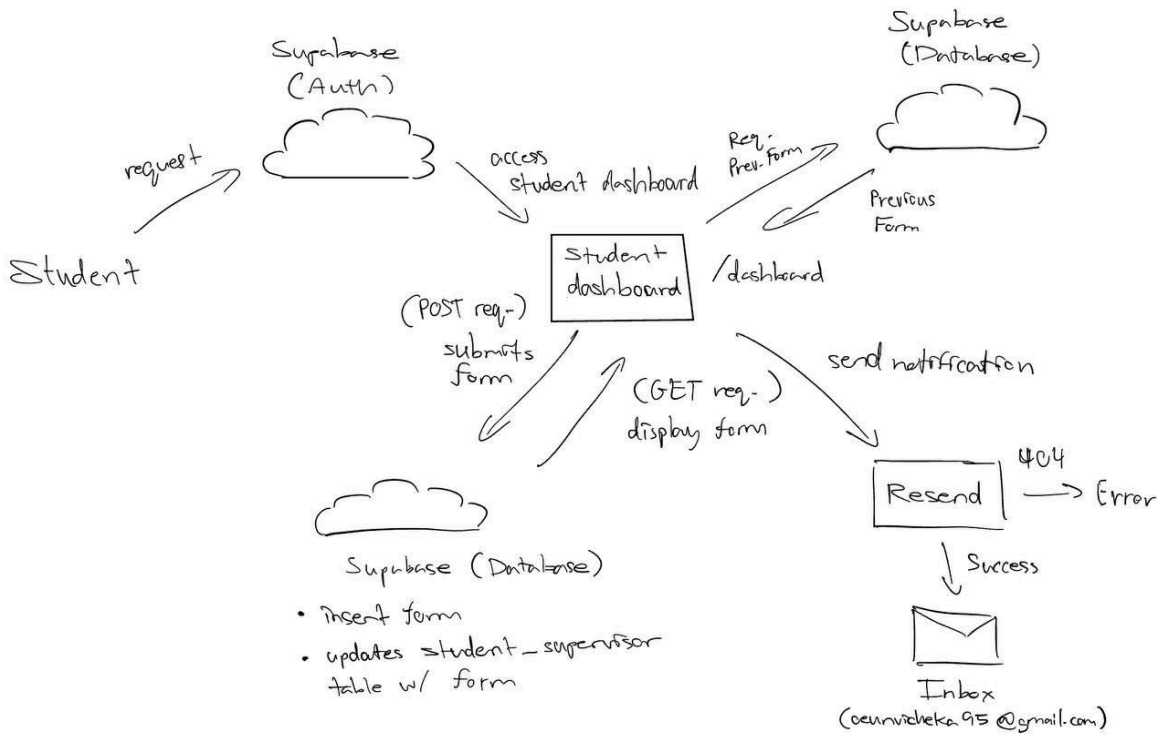
id uuid	created_at timestamptz	student_id uuid	supervisor_id uuid	term varchar
00c9e897-7480-4bdb-9c1b-7c8188e6e63(	2024-11-30 19:20:28.972128+00	d66a6cc6-cccc-4d9b-84a0-6eadf...	NULL	Date
06cdbf29-12c5-432d-8750-a7933ef81cce	2024-12-01 09:44:54.292597+0(	25349987-18a9-4c3d-9bf5-6222f8...	babcb6f2-9f5d-4d07-8bae-816b1d...	Spring 2080
2a9ad8f8-ad34-4cc2-b0a0-9af3541653e8	2024-12-01 09:59:09.895729+0	25349987-18a9-4c3d-9bf5-6222f8...	babcb6f2-9f5d-4d07-8bae-816b1d...	Winter 2078
56176160-81a8-4c4a-b96b-8ae8d595958(	2024-12-01 03:09:17.400576+0(	25349987-18a9-4c3d-9bf5-6222f8...	babcb6f2-9f5d-4d07-8bae-816b1d...	7
6383e06d-147d-4736-a40b-7ad1a861c1e3	2024-11-30 22:15:57.631549+00	1c3aff30-73e1-40f8-bc27-f98c1d5...	44d8b09b-22a1-4d7b-8b80-049c...	3
7feffcdc-184b-454a-99d2-7eea6aa591e5	2024-12-01 01:02:09.010939+0(	25349987-18a9-4c3d-9bf5-6222f8...	babcb6f2-9f5d-4d07-8bae-816b1d...	2
8356adf2-dea9-4821-8656-d8139d2f495(	2024-12-01 09:23:30.512578+0(	25349987-18a9-4c3d-9bf5-6222f8...	babcb6f2-9f5d-4d07-8bae-816b1d...	Fall 2025
94970846-1f78-4a01-adc5-69eafb03029a	2024-12-01 09:31:11.761211+00	25349987-18a9-4c3d-9bf5-6222f8...	babcb6f2-9f5d-4d07-8bae-816b1d...	Winter 2029
ac163d31-4df1-44e6-aec1-dac6200e62b4	2024-12-01 03:03:00.300544+0(	d66a6cc6-cccc-4d9b-84a0-6eadf...	44d8b09b-22a1-4d7b-8b80-049c...	test
d448df9c-382d-457c-aa29-4696c9f7943(	2024-12-01 11:26:48.505125+00	25349987-18a9-4c3d-9bf5-6222f8...	babcb6f2-9f5d-4d07-8bae-816b1d...	Fall 2025

b. If no, a 404 error would be returned

4. Students can also view previous submission & feedback like, this is accomplished through a GET request on the initial access to the dashboard (as mentioned earlier).

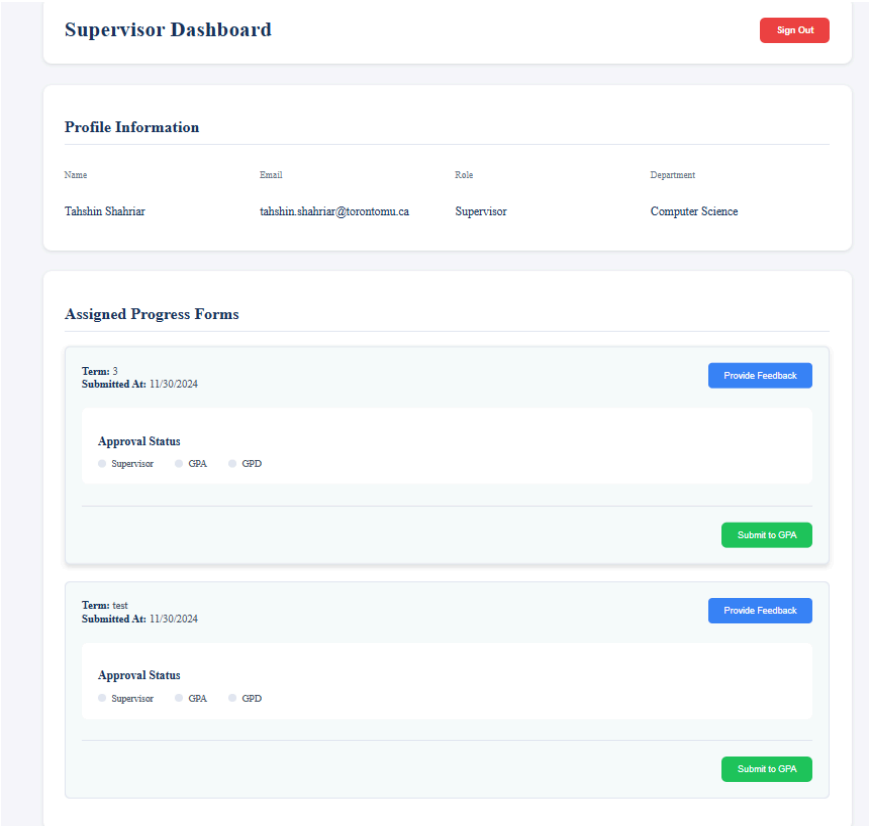


5. A high-level overview of the system:

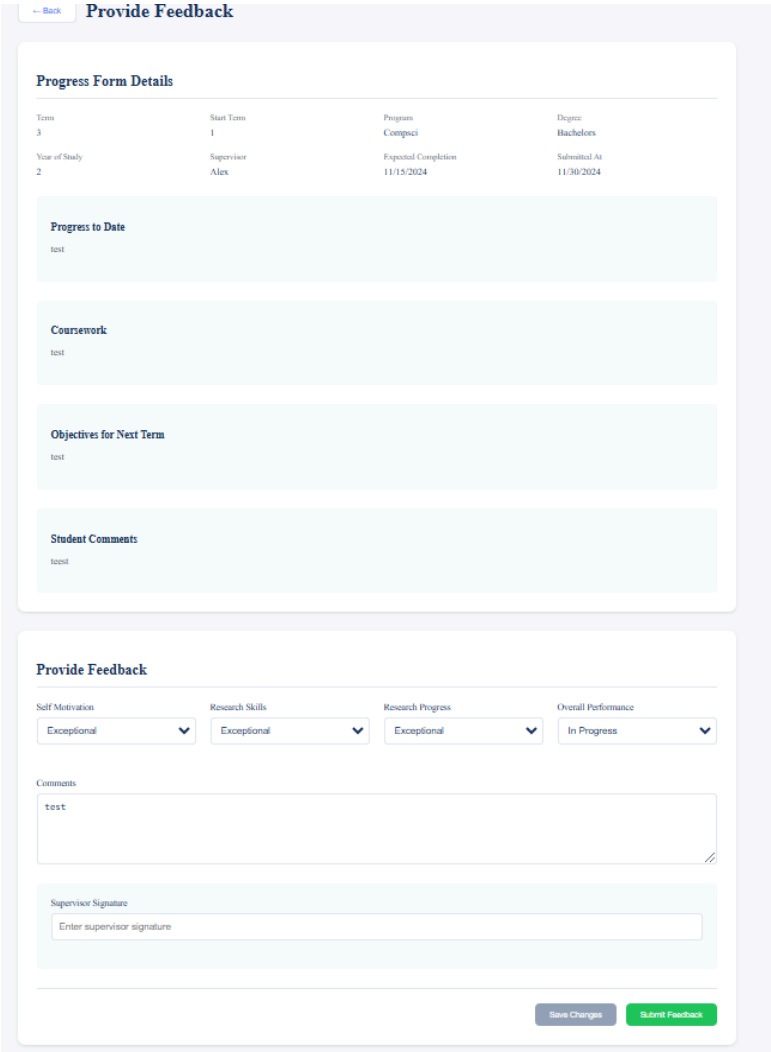


Supervisor Form & Submission

Supervisor’s dashboard displays all the forms submitted by students under them. Supervisors can click on the “Provide Feedback” button next to each form and the system displays a form details page containing the form details and a section to provide feedback . The feedback section contains editable fields, categorized into areas such as "Self-Motivation," "Research Skills," “Research Progress” and "Overall Performance." Supervisors can also add their comments in the comment field and sign off the form. Supervisors can perform two main actions on this page



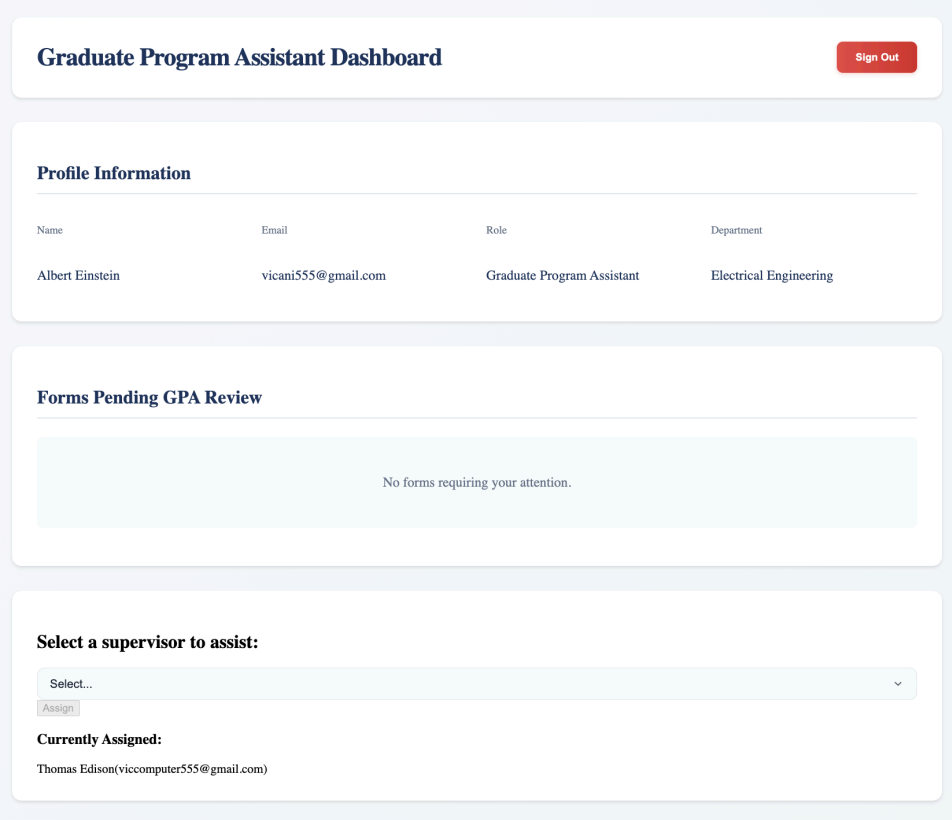
1. **Save Feedback:** This allows the user to save the feedback inputs temporarily without saving them in the database. Upon clicking on the “Save Changes” button, the feedback data is saved locally and the system alerts the user that their changes have been saved.
2. **Submit Feedback:** Once the supervisor finalizes their feedback, they can click on the ‘Submit Feedback’ button to save the feedback into the database. Once the submit button is clicked, the system validates the form to ensure that all the required fields are completed. If any fields are missing, it alerts the user to complete the fields. Upon successful submission, the feedback form data is permanently saved in the database and the review_status of the form is updated to “submitted”. The system alerts the supervisor upon successful submission.



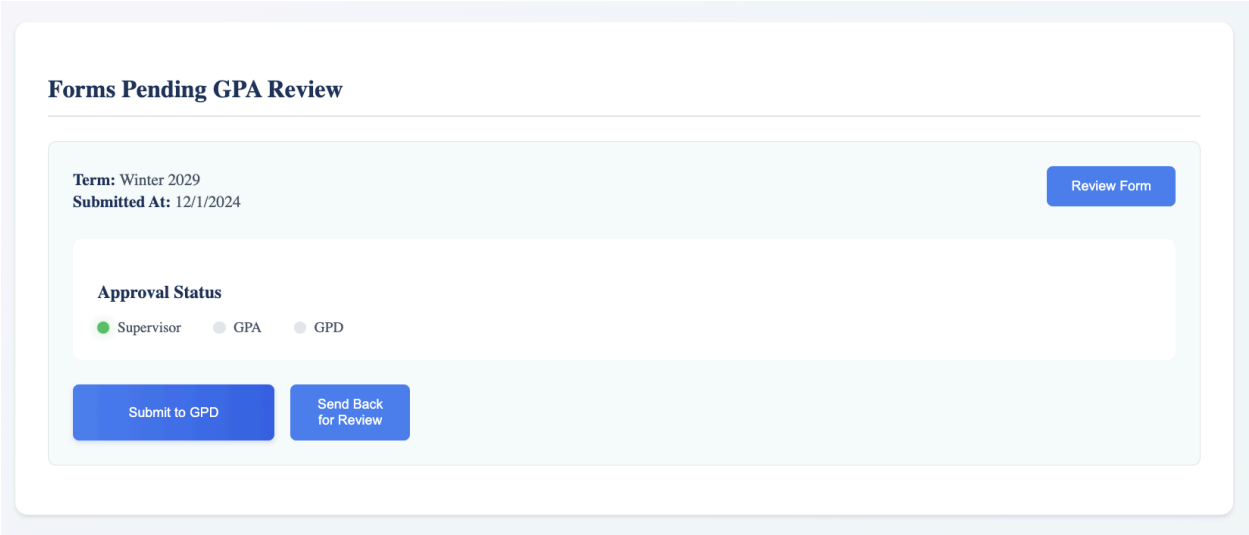
GPA & GPD Form & Submission

The GPA and GPD form works just like how the forms for supervisor works with the exception of submitting to either the GPD if it's the GPA or GPD approving the entire process.

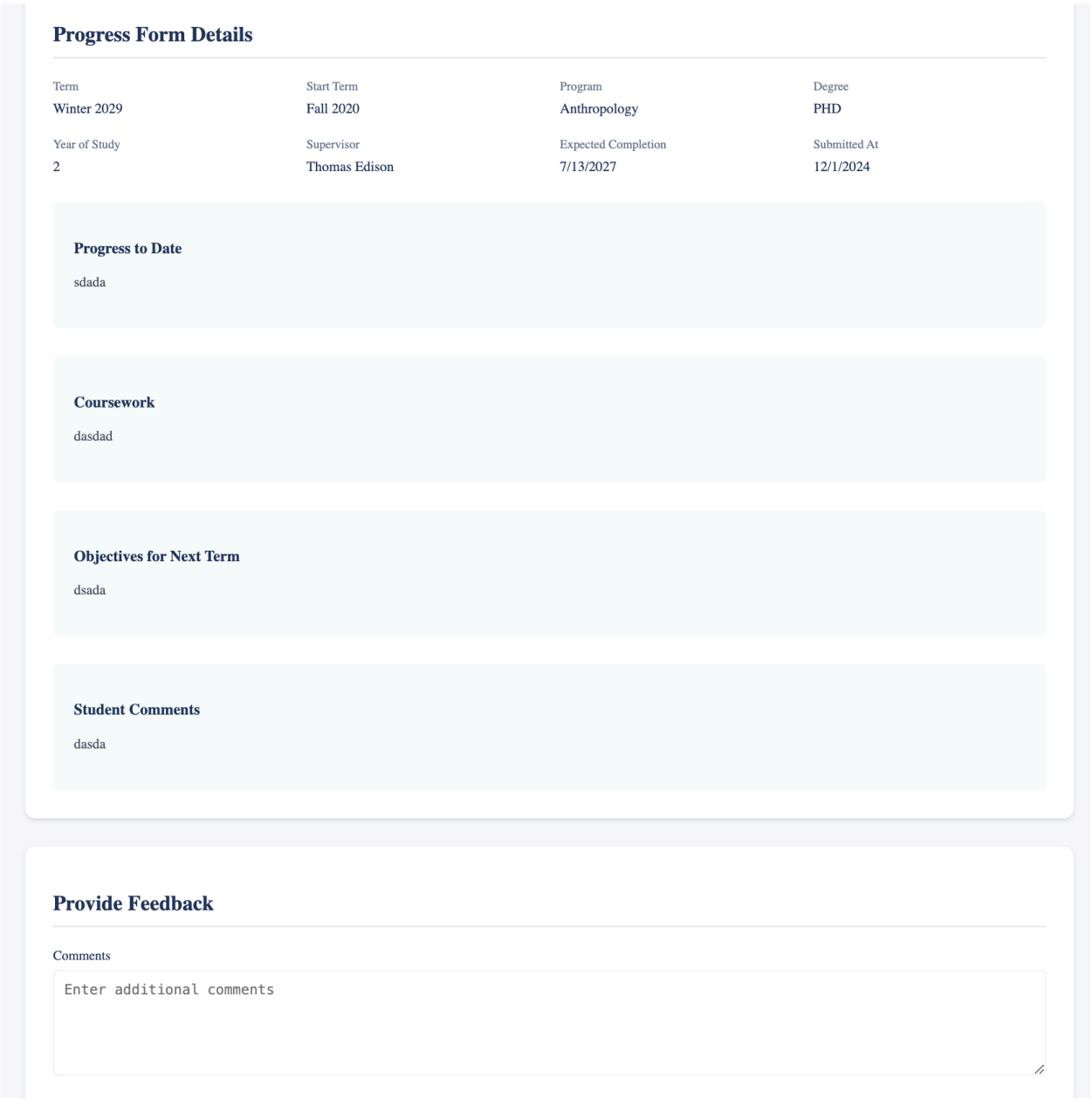
1. **GPA:**
    a. The GPA would be authenticated into their dashboard and displayed with any pending forms submitted by the supervisor.
        i. Example of no pending forms to review:



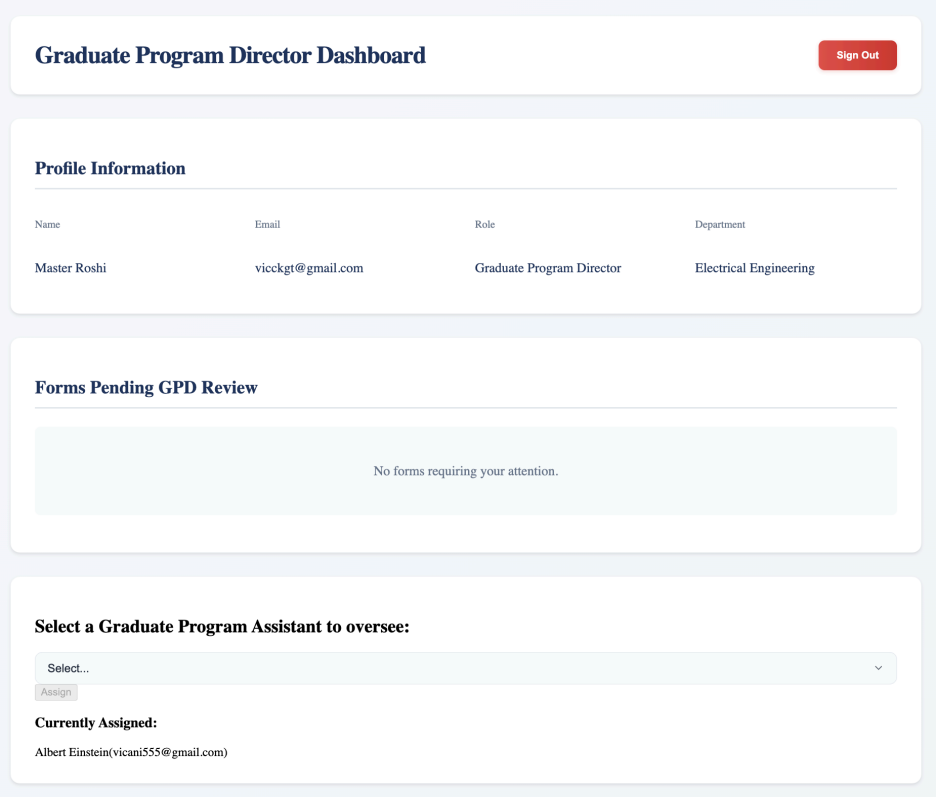
ii. Example of some pending forms to review:



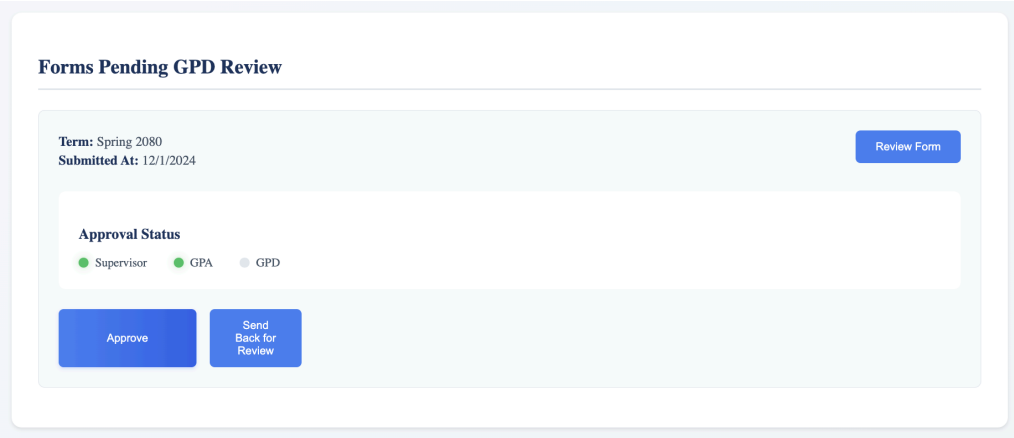
b. The GPA then can view the form, provide feedback and ask to resubmit if they disapprove of the work or send it to the GPD.



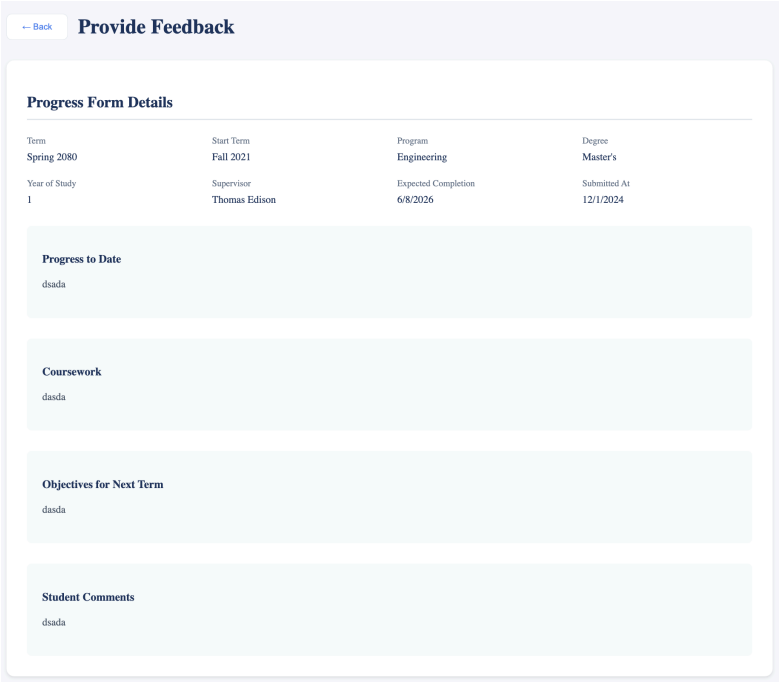
2. **GPD:**
    a. The GPD would be authenticated into their dashboard and displayed with any pending forms submitted by the GPA.
        i. Example of no pending forms to review:

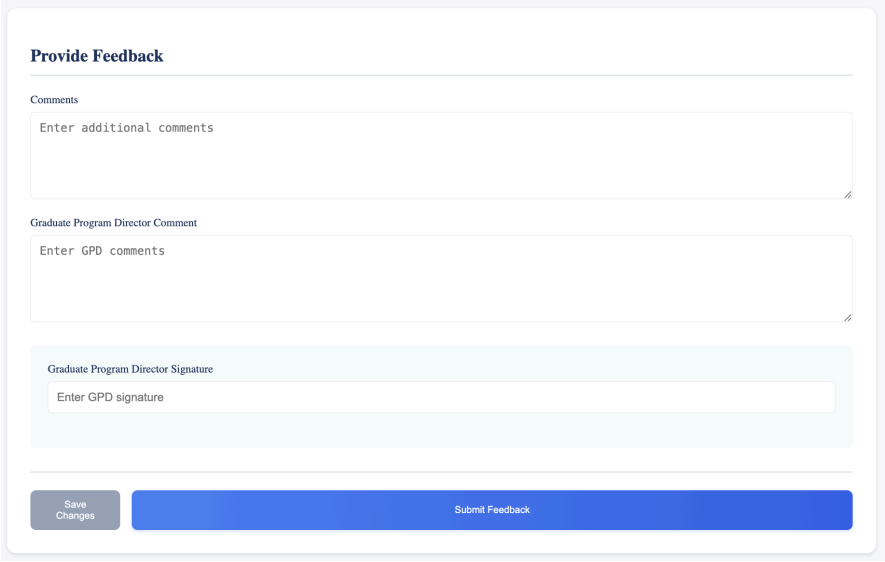


ii. Example of some pending forms to review:



b. The GPD then can view the form, provide feedback and ask to resubmit if they disapprove of the work or send it to the GPA.

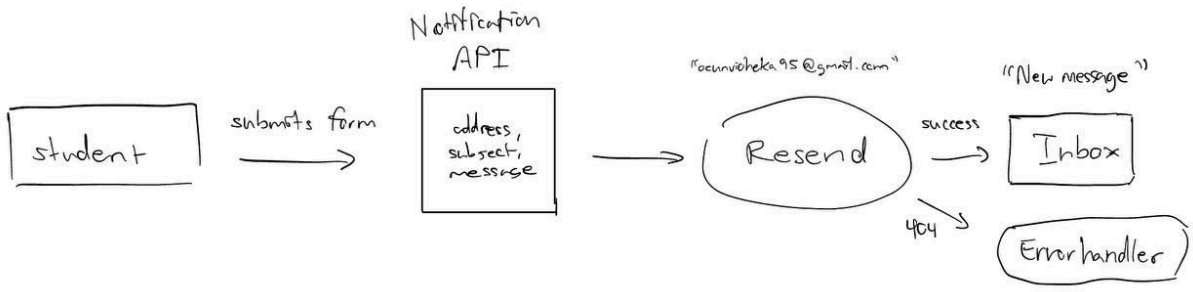




Notification System

Our notification system is implemented using Resend's API, Resend is an Email API designed for testing sending notifications via verified emails. Our notification system is only in development mode, therefore we are only able to send email to our test email *oeunvicheka95@gmail.com*. Having said the way our notification system works is:

- When a user submits a form, the form submission handler would trigger a request to be sent to the notification API locally, then that notification API sends a request to Resend's API (using Resend API key) to pass a message to *oeunvicheka95@gmail.com*. If failed, Resend will respond with a 404 error message.



Styling

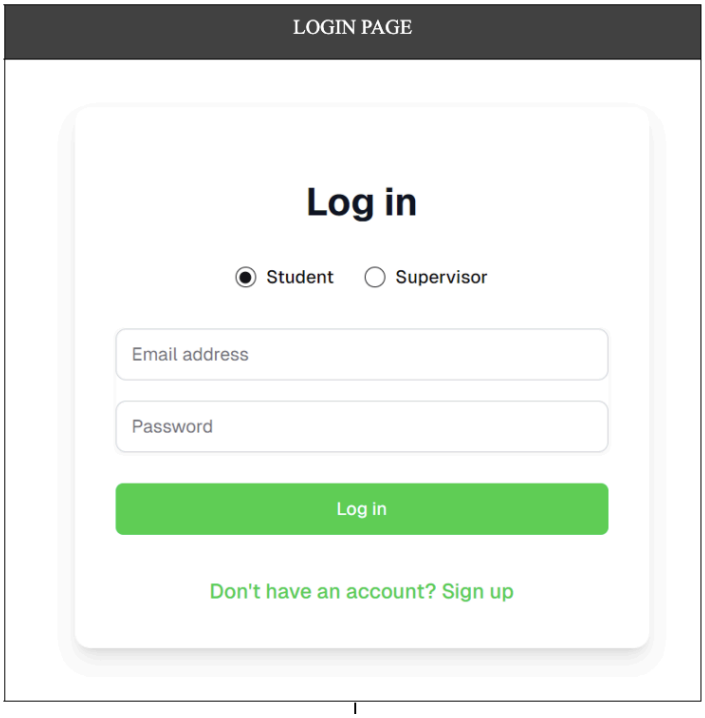
Tailwind CSS was used to design a clean and responsive UI quickly and consistently.

UI Design Iterations

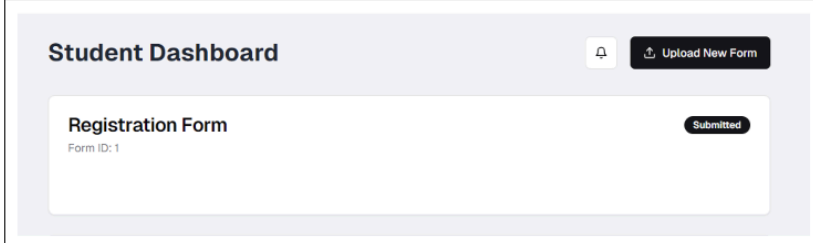
1. **Initial Concept**:
   The first UI version focused on basic functionality, with a simple login page containing minimal styling.
2. **Second Iteration**:
   Based on team feedback, we added custom styling using Tailwind CSS, introduced role-based dashboards, and experimented with layout improvements for clarity.
3. **Final Iteration**:
   To simplify the user experience, we finalized a minimalistic design that aligns with the project requirements. We focus on accessibility and ease of navigation.

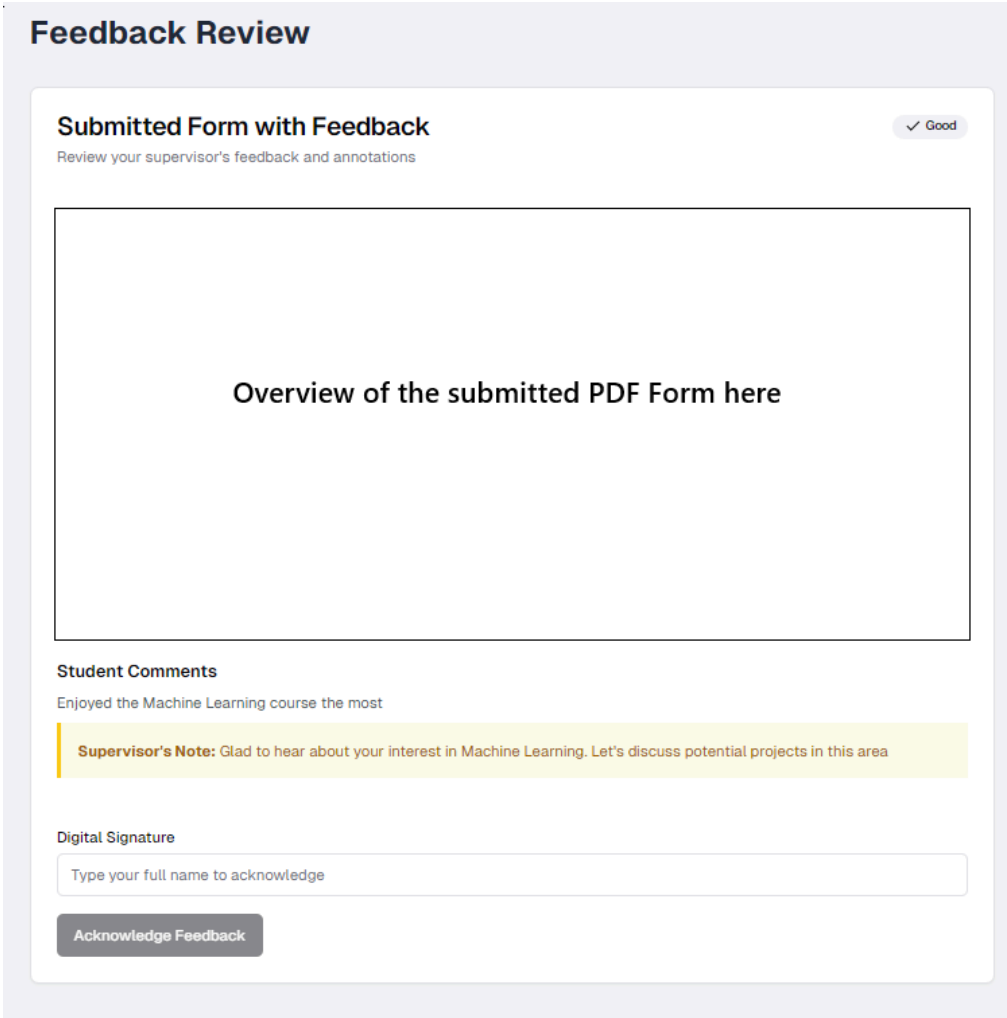
Initial Design:

LOGIN PAGE



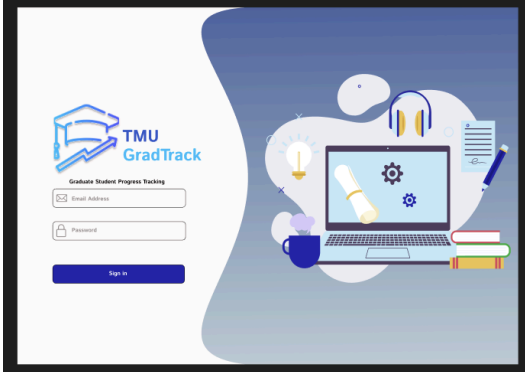
STUDENT DASHBOARD



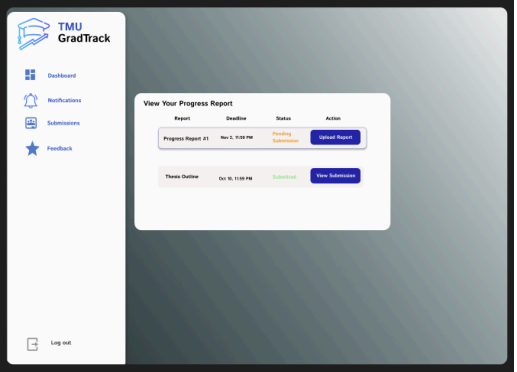


Intermediate Design:

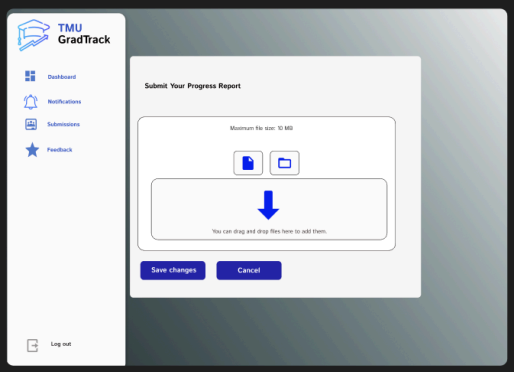
Login Page



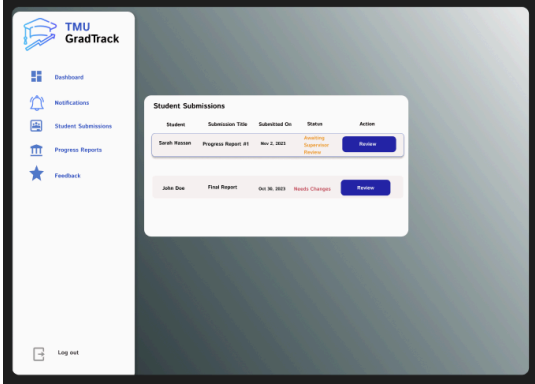
Student Dashboard



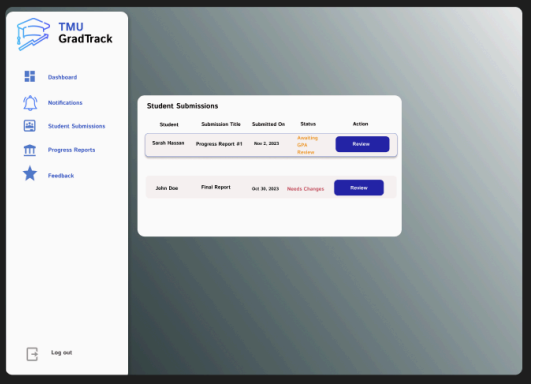
Student Upload



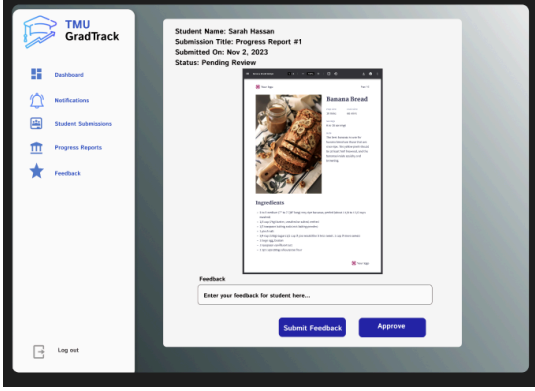
Supervisor Dashboard



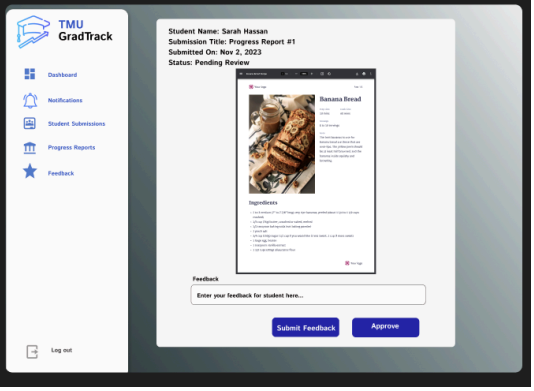
Graduate Program Assistant (GPA) Dashboard



Supervisor Review



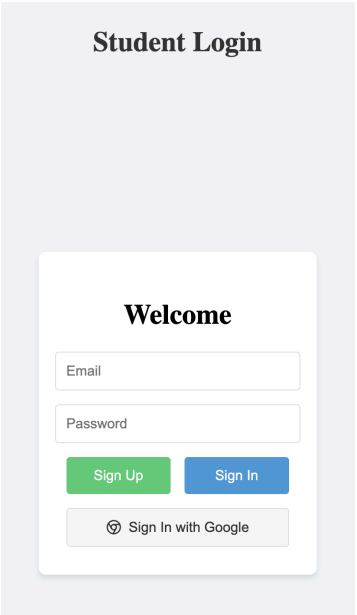
Graduate Program Assistant (GPA) Review

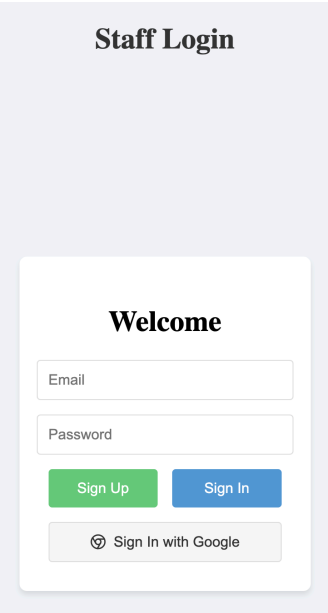


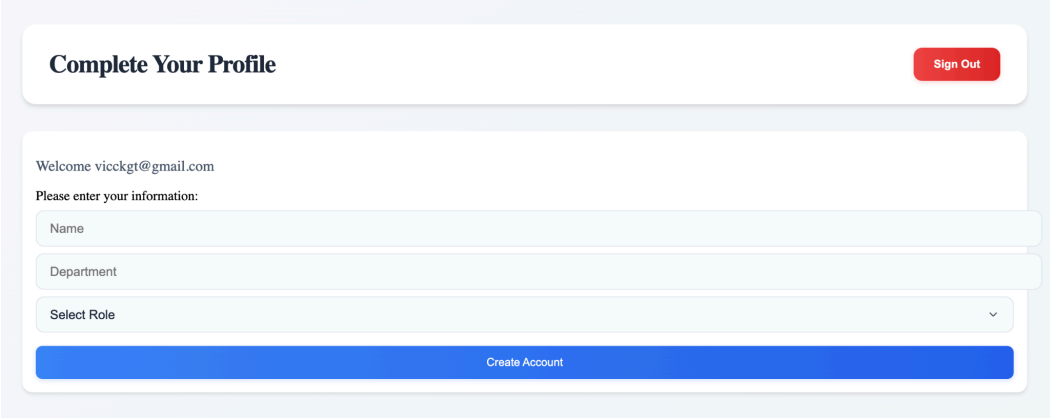
Final Design:

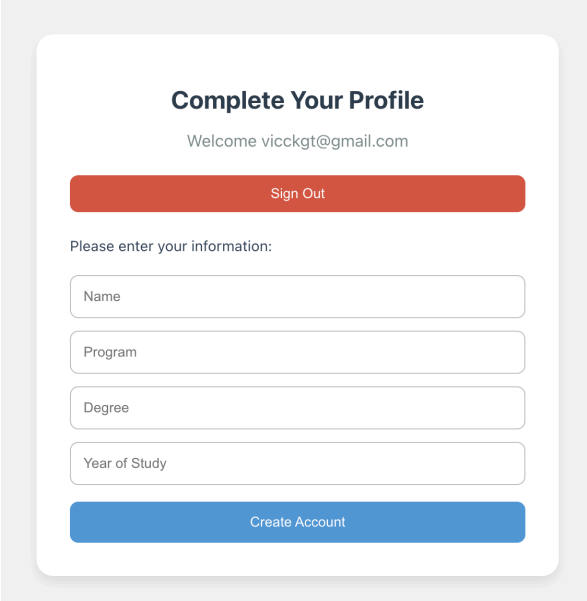
Sign Up

Student Staff


Student Login
Welcome
Email
Password
Sign Up
Sign In
Sign In with Google


Staff Login
Welcome
Email
Password
Sign Up
Sign In
Sign In with Google


Complete Your Profile
Sign Out
Welcome vicckgt@gmail.com
Please enter your information:
Name
Department
Select Role
Create Account


Complete Your Profile
Welcome vicckgt@gmail.com
Sign Out
Please enter your information:
Name
Program
Degree
Year of Study
Create Account

Student Dashboard

Sign Out

Profile Information

Name
Vicheka Oeun

Email
oeunvicheka95@gmail.com

Academic Information

Program
Computer Science

Degree
Bachelor of Science

Year of Study
2030

Supervisor Information

Supervisor Name
Thomas Edison

Supervisor Email
viccomputer555@gmail.com

Department
Electrical Engineering

Graduate Progress Form

Previous Submissions

Progress report for term: Fall 2025
Submitted at 12/1/2024
View Feedback

Progress report for term: Winter 2029
Submitted at 12/1/2024
View Feedback

Progress report for term: Spring 2080
Submitted at 12/1/2024
View Feedback

Progress report for term: Winter 2078
Submitted at 12/1/2024
View Feedback

Progress report for term: 2
Submitted at 11/30/2024
View Feedback

Create New Submission

Basic Information

Term

e.g., Fall 2024

Start Term

e.g., Fall 2023

Program

Your program name

Degree

e.g., Ph.D., Master's

Academic Details

Year of Study

Supervisor Name

Full name of your supervisor

Expected Completion Date

yyyy-mm-dd

Progress Details

Progress to Date

Describe your academic progress this term

Coursework

List completed courses and grades

Objectives for Next Term

What are your academic goals for next term?

Additional Comments

Any additional comments or concerns

Signature

Student Signature

Type your full name

Signature Date

yyyy-mm-dd

Submit Progress Form

Supervisor Dashboard

Sign Out

Profile Information

Name	Email	Role	Department
Thomas Edison	viccomputer555@gmail.com	Supervisor	Electrical Engineering

Assigned Progress Forms

Term: Fall 2025
Submitted At: 12/1/2024

Provide Feedback

Approval Status

Supervisor GPA GPD

Submit to GPA

Term: Winter 2029
Submitted At: 12/1/2024

Provide Feedback

Approval Status

Supervisor GPA GPD

Submit to GPA

Term: Spring 2080
Submitted At: 12/1/2024

Provide Feedback

Approval Status

Supervisor GPA GPD

Submit to GPA

Graduate Program Assistant Dashboard

Sign Out

Profile Information

Name	Email	Role	Department
Albert Einstein	vicani555@gmail.com	Graduate Program Assistant	Electrical Engineering

Forms Pending GPA Review

No forms requiring your attention.

Select a supervisor to assist:

Select...

Assign

Currently Assigned:

Thomas Edison(viccomputer555@gmail.com)

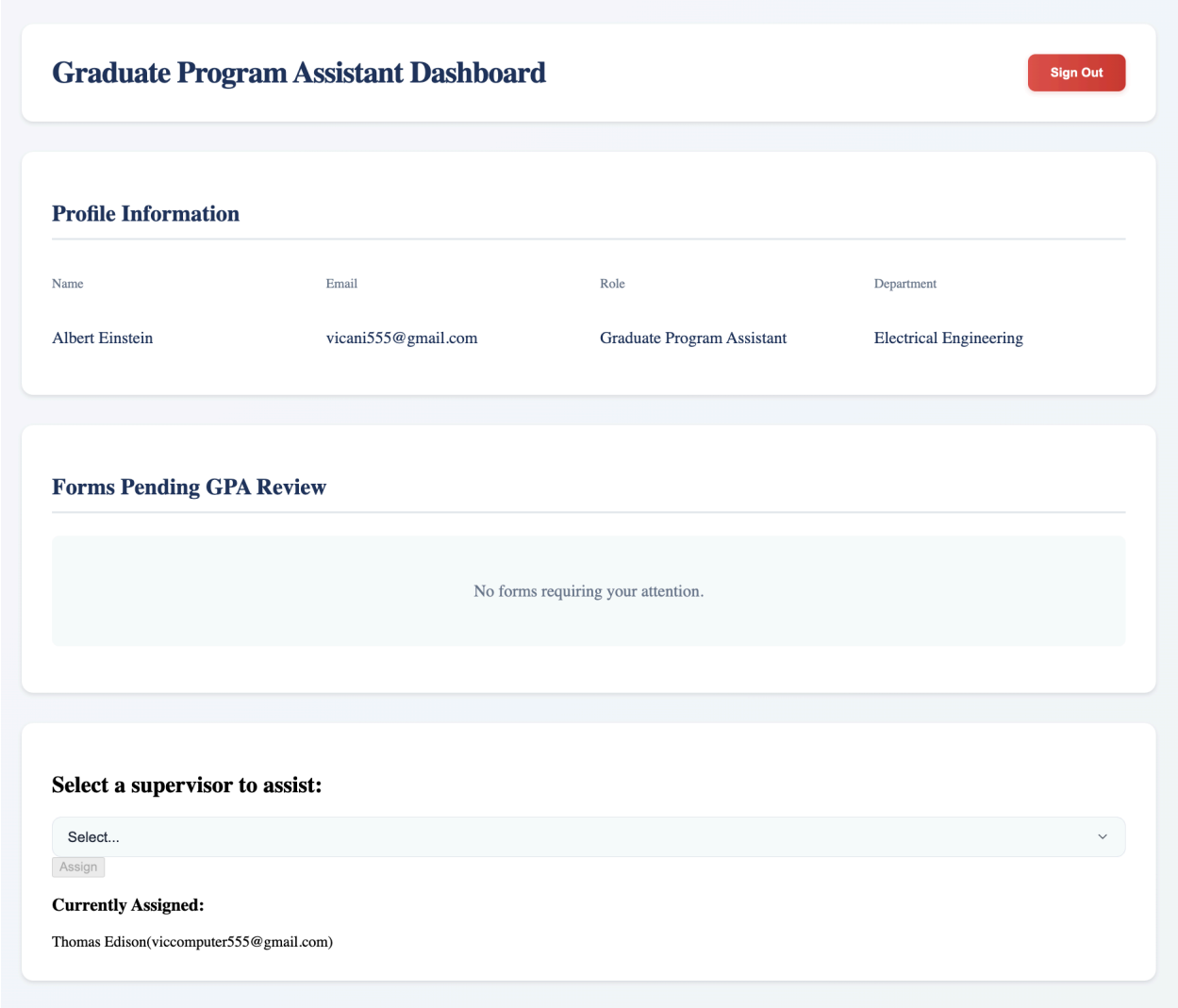
Select a student to supervise:

Select...

Assign

Currently Assigned:

Vicheka Oeun(oeunvicheka95@gmail.com)



Review of Project Plan

This section outlines our reviewed project plan model based on our chosen software process model, which is the Incremental Development Model.

Project milestone:

1.1. Identified requirements & stakeholders
1.2. Chose a software process
2.1. Completed use case
2.2. Identified NFR's

2.3. Completed domain model design
2.4. Finished designing user interface
2.5. Chose & Completed project planning model
3.1. Designed database schema
3.2. Implemented user interface
3.3. Developed dashboard for student(s), supervisor(s), GPA and GPD
3.4. Integrate authentication with Supabase
4.1. Implemented form upload functionalities
4.2. Implemented form editing functionalities
4.3. Developed form submission
5.1. Implemented feedback functionality
5.2. Developed annotation tools
5.3. Developed student(s) feedback viewing
6.1. Completed integration with Resend for notifications
6.2. Implemented notification triggers for form submission and feedback
7.1. Completed integration of all increments
7.2. Completed testing
7.3. Performed user acceptance testing
8.1. Completed setup of environment
8.2. Deployed application
8.3. Conducted and gathered user feedback

Project Task & Activities:

Activity	**Time Estimate (in days)**

Project Proposal	
Activity 1.1. Identify requirements & stakeholders	1
Activity 1.2. Select software process	1
Project planning	
Activity 2.1. Use case modeling	3
Activity 2.2. Identify non-functional requirements	1
Activity 2.3. Domain model design	2
Activity 2.4. User interface design	3
Activity 2.5. Complete Project planning model	1
Increment 1	
Activity 3.1. Design database schema	2
Activity 3.2. Implement user authentication	5
Activity 3.3. Development of dashboard for student(s), supervisor(s), GPA and GPD	7
Activity 3.4. Integrate authentication with Supabase	6
Increment 2	
Activity 4.1. Implement form upload functionality	2
Activity 4.2. Develop form editing capability	2
Activity 4.3. Create student form submission process	2
Increment 3	
Activity 5.1. Implement feedback functionality	2
Activity 5.2. Develop annotation tools	2
Activity 5.3. Create student feedback viewing system	2
Increment 4	

Activity 6.1. Completed integration with Resend for notifications	2
Activity 6.2. Implement notification triggers for form submission and feedback	2
Increment 5	
Activity 7.1. Integrate all increments	2
Activity 7.2. Write and conduct system testing	2
Activity 7.3. Perform user acceptance testing	1
Final Phase	
Activity 8.1. Setup deployment environment	1
Activity 8.2. Deploy application	1
Activity 8.3. Gather user feedback	1