Grader Assignment System

Team 11 - CS 4485.0W1 Group Project

Madison Hokstad

Do Kyung Lee

Alexandra Ontiveros

Gaby Salazar Mejia

Anh Tran

Supervisor: Professor Sridhar Alagar

Course Coordinator: Thennannamalai Malligarjunan

Erik Jonsson School of Engineering and Computer Science

University of Texas at Dallas

May 9, 2025

Table of Contents

1. Introduction

1.1 Background
1.2 Objectives, Scope, and Goals Achieved
1.3 Key Highlights
1.4 Significance of the Project

2. High-Level Design

2.1 System Overview & Proposed Solution
2.2 Design and Architecture Diagrams
2.3 Overview of Tools Used

3. Implementation Details

3.1 Technologies Used
3.2 Code Documentation
3.2 Development Process

4. Performance & Validation

4.1 Testing and Validation
4.2 Metrics Collected and Analysis

5. Lessons Learned

5.1 Insights Gained
5.2 Lessons from Challenges
5.3 Skills Developed and Improved

6. Future Work

6.1 Proposed Enhancements
6.2 Recommendations for Development

7. Conclusion

7.1 Summary of Key Accomplishments
7.2 Acknowledgements

8. References
9. Appendices

Chapter 1: Introduction

1.1 Background

When a new semester begins, UTD always comes across a problem with assigning graders to all the courses needed, as they have to manually assess the applicants based on the criteria of the classes. This process takes a long time and affects the grading of the individual courses; that's where this project comes in. GAS - the grader assignment system - aims to solve that problem by automating the assignment process of graders to classes based on criteria that graders put into the application. This will help relieve the work of faculty and advisors at the start of each school semester.

1.2 Objectives, Scope, and Goals Achieved

- **Objectives:**
  - Automate the matching process for graders
  - Find the optimal assignment of candidates to professors and courses
  - Ability to run the matching algorithm for single-grader assignment when desired
  - Provide different views and flexibility for the hiring manager
  - Provide ease of use for all users
- **Scope:**
  - File Handling & Data Processing:
    - Support reading and generating CSV and Excel files
    - Read and extract relevant data from PDF files
  - Matching Algorithm & Candidate Selection
    - Implement a matching algorithm that assigns graders to courses based on course requirements, grader qualifications, and professor recommendations
    - Provide reasoning for each assignment to allow the hiring manager to verify match reliability
  - Automation & Manual Review
    - Implement hiring manager ability to automate grader assignment process using the matching algorithm, review and modify assignments, and reassign graders manually if needed
    - Implement professor ability to submit recommended graders
  - Dynamic & Incremental Assignment Updates
    - Allow new courses and graders to be added at any time
    - Allow cancelled assignments to automatically trigger a reassignment process among available candidates

        - Allow new courses to be dynamically integrated into the matching process
    - User Authentication & Role-Based Access
        - Implement a login system for secure access
- **Goals Achieved:**
    - Support reading and generating CSV and Excel files for input
    - Able to read and extract relevant data from PDFs for resume analysis
    - Implemented matching algorithm that assigns graders to courses based on input CSV and PDF files
    - Provide reasoning for each assignment, allowing hiring manager to verify match reliability
    - Implemented hiring manager ability to automate grader assignment process using the matching algorithm
    - Implemented hiring manager ability to review and modify assignments and manually reassign graders if needed
    - Implemented professor ability to submit grader recommendations
    - New courses and graders can be added at any time and the matching algorithm can be run again
    - Implemented a login system for secure access

1.3 Key Highlights

- **Feature 1:** Reading and parsing input files
- **Feature 2:** Automatic matching of candidates to courses
- **Feature 3:** Clear displays to communicate the current state of assignments and facilitate manual adjustments
- **Feature 4:** Exporting assignments in excel/csv format once finalized

1.4 Significance of the Project

This project is necessary because manually looking through applications and resumes is tedious and a poor use of time. This project will greatly increase the efficiency of the process, especially as the size of the CS program grows larger and larger over time. By automating large chunks of the process, which our project does, this time can be better allocated to other administrative tasks.

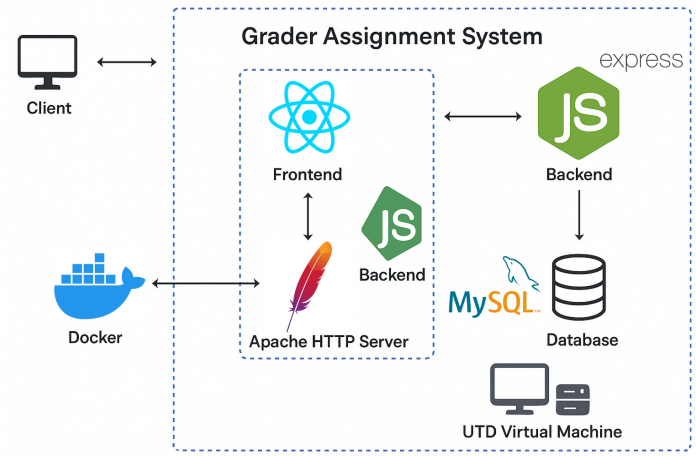
Chapter 2: High-Level Design

2.1 System Overview & Proposed Solution

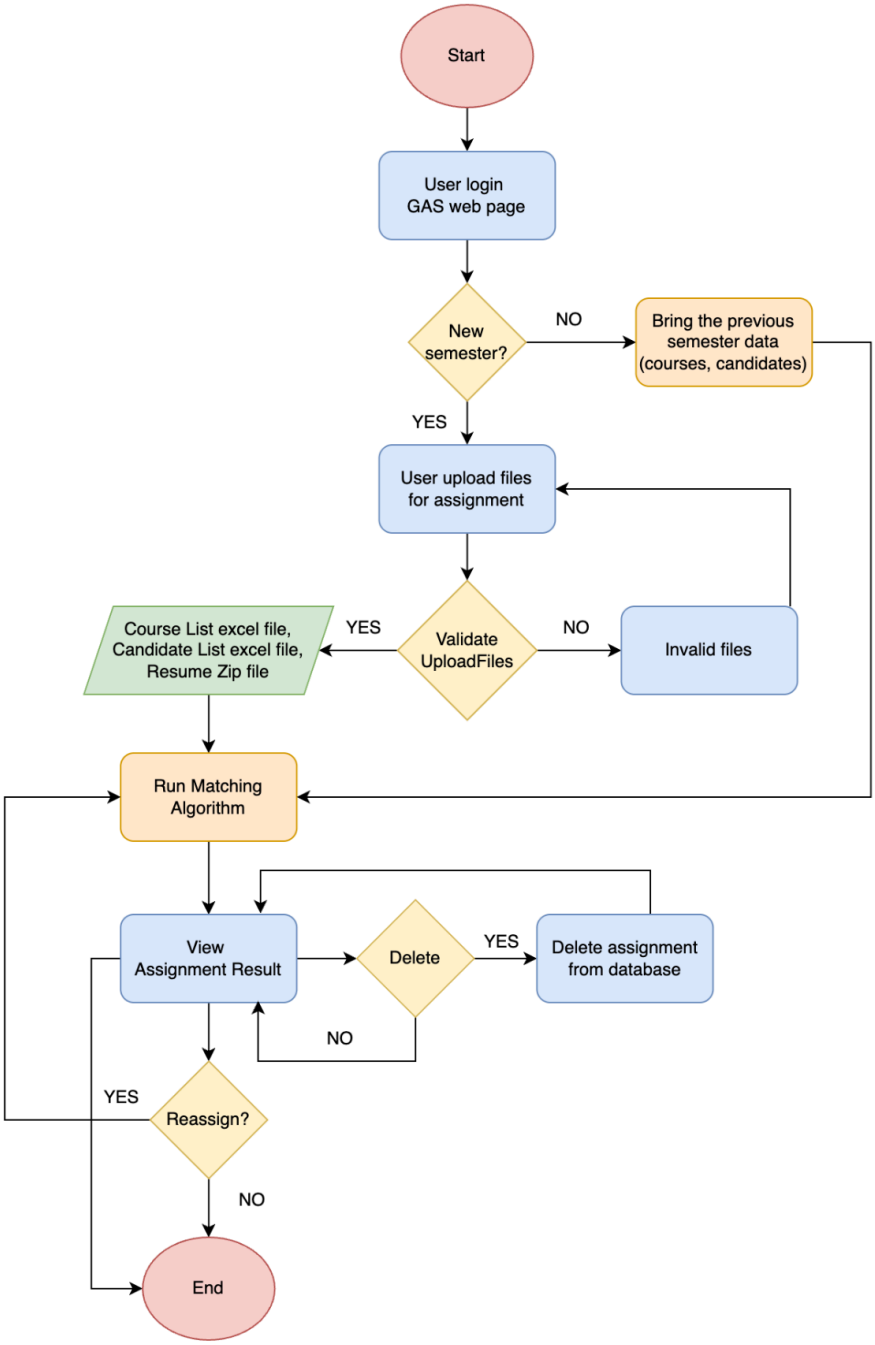
- Provide an overview of API specifications, key endpoints, and their purpose.

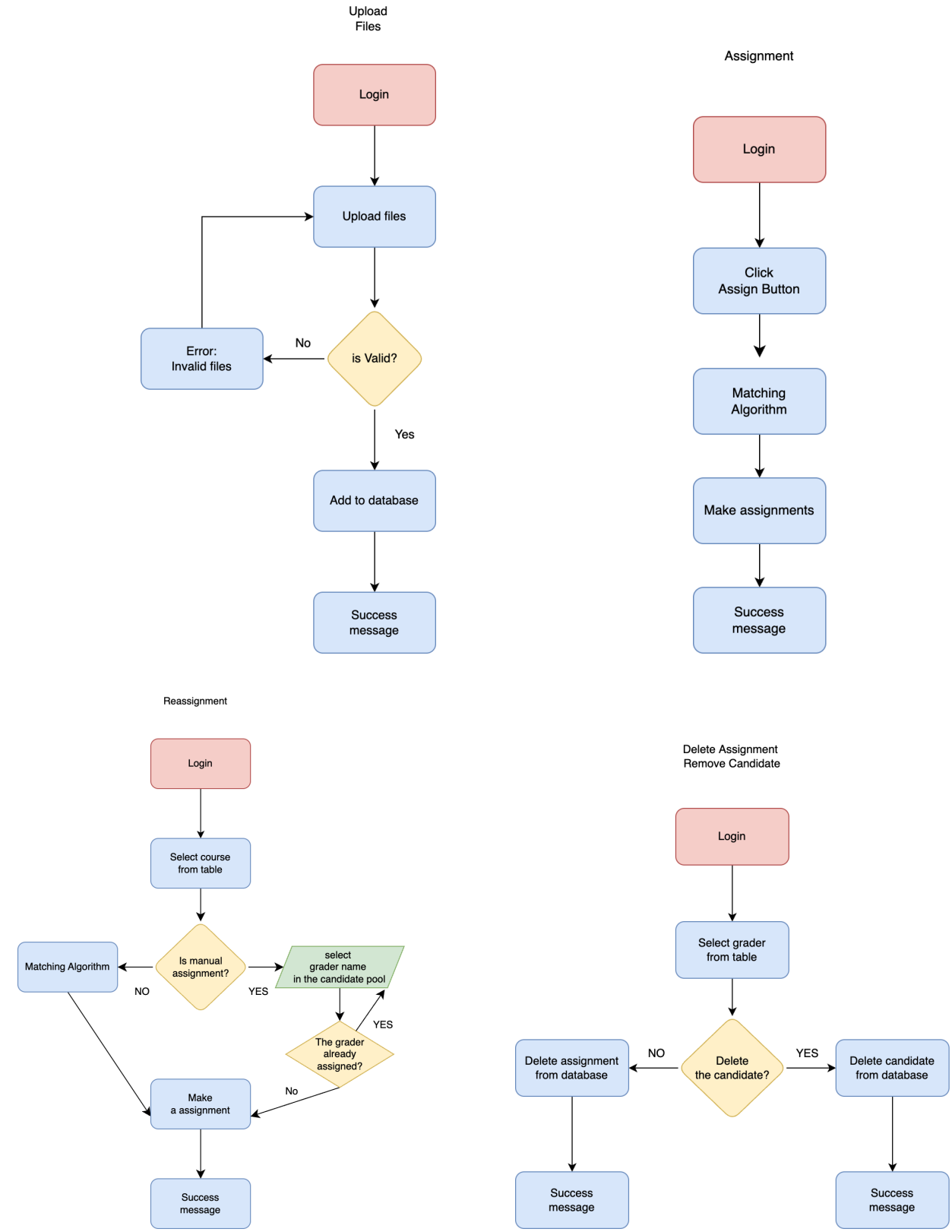
- **/applicants**
  - allows create, read, and delete operations for applicants
- **/assignments**
  - allows read, update, and delete operations for assignments
- **/login**
  - allows login operations
- **/courses**
  - allows create, read, and delete operations for courses
- **/match**
  - allows running matching algorithm
  - allows reading results of matching algorithm
  - allows running matching algorithm for reassignment
- **/upload**
  - allows upload operations for applicant resume files
  - allows upload operations for course, candidate, and recommendation files as CSV or Excel files

2.2 Design and Architecture Diagrams

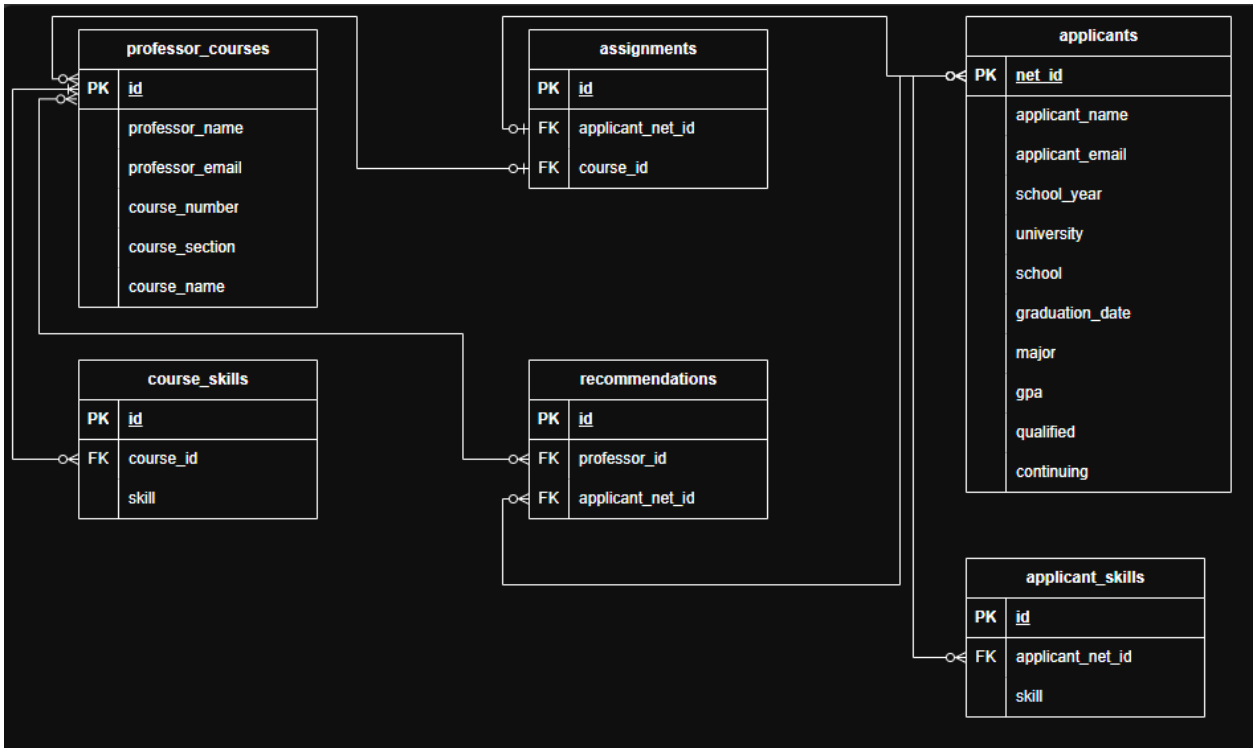


(Figure 1. System Architecture Diagram)


Start
User login
GAS web page
New
semester?
NO
Bring the previous
semester data
(courses, candidates)
YES
User upload files
for assignment
Validate
UploadFiles
YES
NO
Course List excel file,
Candidate List excel file,
Resume Zip file
Invalid files
Run Matching
Algorithm
View
Assignment Result
Delete
YES
Delete assignment
from database
NO
YES
Reassign?
NO
End



(Figure 2. Flow Diagram)



(Figure 3. ER Diagram)

2.3 Overview of Tools Used

- **Frontend**
  - React
- **Backend**
  - Node.js
  - Express.js
- **Database**
  - MySQL
- **Cloud & Hosting**
  - Docker
  - Apache
- **Version Control & Project Management**
  - Github
- **UI/UX Design**
  - Figma

Chapter 3: Implementation Details

3.1 Technologies Used

- **Frontend**
  - React: React was used to build the user interface of the web application, which enabled the user to create and handle reusable components and dynamic data. This led to creating an interactive user experience, ensuring the best user-friendly interface.
  - Node.js: Node.js was used by the runtime environment as a form of executing Javascript code on the server side. This allowed for the functionality of backend services to connect to frontend services.
  - Express.js: Express.js was used as a web-application framework that built upon Node.js, which was used for routing and handling and managing HTTP requests.
- **Database**
  - MySQL was used for storing and managing application data. It provided reliable data storage, supported SQL queries, and maintained data integrity and security.
- **Cloud & Hosting**
  - Docker was used for packaging the application, which made it so that all of its dependencies were contained in a single unit called the container. It ensured that across these containers, consistency was held amongst the development, testing, and production environments, which simplified deployment and dependency management.
  - Apache acted as a web server that served both the frontend and the backend, handling HTTP requests and managing any possible traffic routing issues that may have arised.
- **Version Control & Project Management**
  - Github was used as a source managing and team collaboration tool. It allowed for trafficking issues to be resolved, version control, code reviews, and project documentation to be resolved amongst the team,
- **UI/UX Design :** Figma was used as a collaborative design tool that was to facilitate the design of the web's application's user interface. It enabled the use of designing a cohesive and user-friendly interface.

3.2 Code Documentation

  - https://github.com/catally3/grader-assignment-system/tree/main
  - The main key functions and logic included in this codebase are the search functionality, sorting functionality,and reassignment functionality. These functionalities are used across most of the user views, and are used to provide a more efficient user experience.
  - The search functionality enables the user to search through any of the fields based on what search term the user inputs and is looking for. The sorting functionality allows the user to input whichever field they would like to search through. Afterwards, the user inputs whatever search term they would like to search for, and based on that criteria, the results are produced. The reassignment functionality, allows the user to reassign any

of the graders of the courses that have been assigned. It is called by either selecting the reassignment button or by deleting the current grader assigned to it. For both views, after deleting the current grader assigned to the course, a list of the current available graders will be displayed, which enables the user to directly select which one they would like to assign to the course.

3.3 Development Process

Everyweek, we set clear goals that determined what we were going to complete that week and also talked about any issues that had arisen in our previous week of work. Using Github Projects made it simple to understand these topics as well as helping us debug any possible issues that had arisen that week, because although every person was free to simply upload the code had worked on for that week first, they first had to ensure that any changes that were shared through Github through pull requests had to be reviewed first reviewed by team members before merging. This allowed us to catch any mistakes and ensured the quality of our code. This, alongside clear communication through throughout the team members in our weekly meetings and beyond, helped debug any issues that could have possibly arised.

Chapter 4: Performance

4.1 Testing and Validation

The testing strategies that our team used were employed throughout the Agile development process. We made sure that as each component was built,console logging and manual UI validation was used to debug. Debugging was handled collaboratively, with any issues arising talked about amongst team members. Before moving on to deployment, we made sure that every part of the application was working as intended,

4.2 Metrics Collected and Analysis

This application was only designed for small scale-use, so we did not focus on handling large use-cases that included lots of data and a heavy traffic load. Our primary goal was to ensure a stable performance under typical usage conditions. Our pages loaded quickly in the local environment, with the average being around 0.55 seconds and response time average around 0.57 seconds, without any noticeable delays. Our API returned data consistently, within reasonable time frames and low error handling was observed as any bugs identified during testing were resolved before merging. Overall the application was able to be used appropriately for our project.

Chapter 5: Lessons Learned

5.1 Insights Gained

- End-to-end file ingestion is complex. Handling CSV/Excel for candidate lists plus ZIP-wrapped PDFs required a robust, layered pipeline: normalize raw headers, parse tabular data, then fall back to resume text parsing for missing fields.
- Input normalization is critical. We discovered dozens of subtle header variants ("Recommended Student ID" vs. "Doc IDs", date ranges in MM/YYYY vs. Month YYYY), and learned that a small normalization layer (lowercase/strip & pick-first) saves enormous downstream headaches.
- Server-side filtering & pagination scales. Moving search/filters out of React and into Sequelize not only sped up the UI for 400+ records, but also simplified state management on the client.
- A clear model boundary prevents confusion. Flattening nested Assignment → Applicant + Course results in the controller gave us a single, predictable shape on the front end, avoiding endless row.Course?.course_number lookups.

5.2 Lessons from Challenges

- Varied resume formats. Detecting "Data Scientist Intern at Little-Cooper (2022 – 2023)" versus "Company, City 01/2015 – Present" forced us to rethink our header-splitting regex. We generalized to capture MM/YYYY, Month YYYY, year–year, and "at"-delimited titles.
- Database key constraints. Default values on net_id and composite PKs led to unique-key conflicts on repeated uploads. We resolved this by switching to an auto-increment candidate_id and using upsert/bulkCreate({ updateOnDuplicate }) for idempotent imports.
- Bridging front-end/back-end expectations. Our React table assumed flat fields; the nested Sequelize included broke that. Flattening on the server or adapting the JSX were both viable, but we learned it's far cleaner to let the controller return exactly what the UI needs.
- Asynchronous orchestration. Coordinating Excel parse → DB seed → ZIP resume parse → DB merge required careful error handling and transaction boundaries so partial failures don't leave the database in an inconsistent state.

5.3 Skills Developed and Improved

Technical
Advanced **regex** and parsing strategies to handle real-world, unstandardized resume headers and date formats.

- Mastery of pdf-parse, XLSX and csv-parser for heterogeneous file types.
- Deep dive into Sequelize relationships (hasMany/belongsTo), composite primary keys, and bulk upserts (updateOnDuplicate) to enforce idempotent imports.
- Building dynamic React filter/search UIs wired to paginated, server-side APIs—simplifying client logic and improving performance.

End-to-end debugging of file uploads, middleware (Multer), controllers, and error propagation back to Postman/React.

Teamwork & Process

- Iterative specification refinement through regular demos: each new corner-case (missing student IDs, recommended students, continuing graders) was captured, prioritized, and baked into the next sprint.
- Clear separation of concerns: backend APIs own filtering, matching, and data shape; frontend owns presentation and minimal state orchestration.
- Strengthened code-review discipline around schema changes (DB migrations + model updates + routes) to avoid "it works on my machine" misalignments.

Chapter 6: Future Work

6.1 Proposed Enhancements

Advanced NLP résumé parsing

- Integrate a lightweight NLP model (e.g. spaCy or a fine-tuned transformer) to better recognize role titles, company names, and project descriptions—reducing reliance on brittle regex.
- Extract semantic entities (skill clusters, seniority levels) so the system can automatically categorize "Senior Data Scientist" vs. "Data Scientist Intern."

Broader document support

- Add DOCX and TXT parsing alongside PDFs, and even support LinkedIn profile imports via simple HTML scrapes.

Real-time processing & notifications

- Move résumé-and-course ingestion into a message queue (e.g. RabbitMQ or AWS SQS) so uploads can be processed asynchronously and report back progress.
- Push real-time notifications (e.g. via WebSocket or email) when a candidate's résumé has been fully parsed and matched.

Enhanced matching algorithm

- Tune matching weights using historical assignment success data (e.g. student performance scores) to continuously improve "best fit."
- Expose a small feedback UI so graders can "thumbs-up" or "thumbs-down" assignments, feeding back into the algorithm.

UI/UX improvements

- Replace the basic text-input filter with dynamic typeahead dropdowns for applicant IDs and names (with autocomplete).
- Add bulk-reassign actions (multi-select rows) and export matched assignments to CSV/Excel in one click.
- Introduce keyset ("infinite scroll") pagination for very large applicant pools.

Monitoring, metrics & audit logs

- Instrument the parser and API with Prometheus metrics (parse-time, error rates) and a lightweight ELK/EFK stack for log aggregation.
- Record a tamper-proof audit trail of all uploads, assignments, and reassignments for compliance.

6.2 Recommendations for Development

Adopt a rigorous testing strategy

- Write **unit tests** for each parser component (Excel header normalization, PDF regex, duration extraction).
- Add **end-to-end tests** (e.g. Cypress) that simulate a full ZIP+Excel upload and confirm the database state.

Establish CI/CD & schema migration workflows

- Use GitHub Actions (or similar) to run linting, tests, and deploy to staging on every PR.
- Manage database changes with a migrations tool (e.g. Sequelize-CLI or Flyway) so future schema tweaks never break production.

Maintain clear API contracts

- Document each endpoint (Swagger/OpenAPI), including query parameters for filtering and pagination.
- Version your APIs (e.g. /v1/assignments) so you can add new features without breaking existing clients.

Modularize parsing logic

- Keep file-type parsers (CSV, Excel, PDF) as independent modules behind a common interface. That makes it easy to swap in improved parsers later.

Foster collaborative code reviews

- Encourage small, focused PRs that touch only one area (e.g. “Improve header normalization”).
- Peer-review regex patterns and parsing heuristics—those are easy to get wrong on corner cases.

Plan for incremental roll-outs

- Release enhancements (NLP parser, async queue) behind feature flags. This lets you A/B test and roll back quickly if needed.

Gather user feedback continuously

- Schedule short demos with graders and administrators after each sprint to capture new edge-cases (e.g. “some resumes embed tables”).
- Build a lightweight in-app “Report a parsing error” link so real users can flag documents that are mis-parsed.

Chapter 7: Conclusion

7.1 Summary of Key Accomplishments

We have completed the base functionality of the goal application to improve the efficiency of assigning graders to courses. Within the duration of this project, we created the application, which can read in files containing applicant and course data, automatically create matches, and allow for administrator viewing and adjustment. We reacted dynamically to feedback and to changing and adapting requirements over the course of the project.

7.2 Acknowledgements

Thank you to Dr. Sridhar Alagar and Thennannamalai Malligarjunan for the feedback and advice throughout the course of this project.

References

Back-end

https://expressjs.com/en/guide/routing.html

https://expressjs.com/en/guide/writing-middleware.html

https://docs.docker.com/guides/

https://gitlab.com/autokent/pdf-parse

https://github.com/mafintosh/csv-parser

https://github.com/exceljs/exceljs

https://github.com/ryu1kn/csv-writer

https://httpd.apache.org/

Algorithm

https://github.com/addaleax/munkres-js

Front-end

https://react.dev/

https://github.com/react-pdf-viewer/react-pdf-viewer

https://joinhandshake.com/

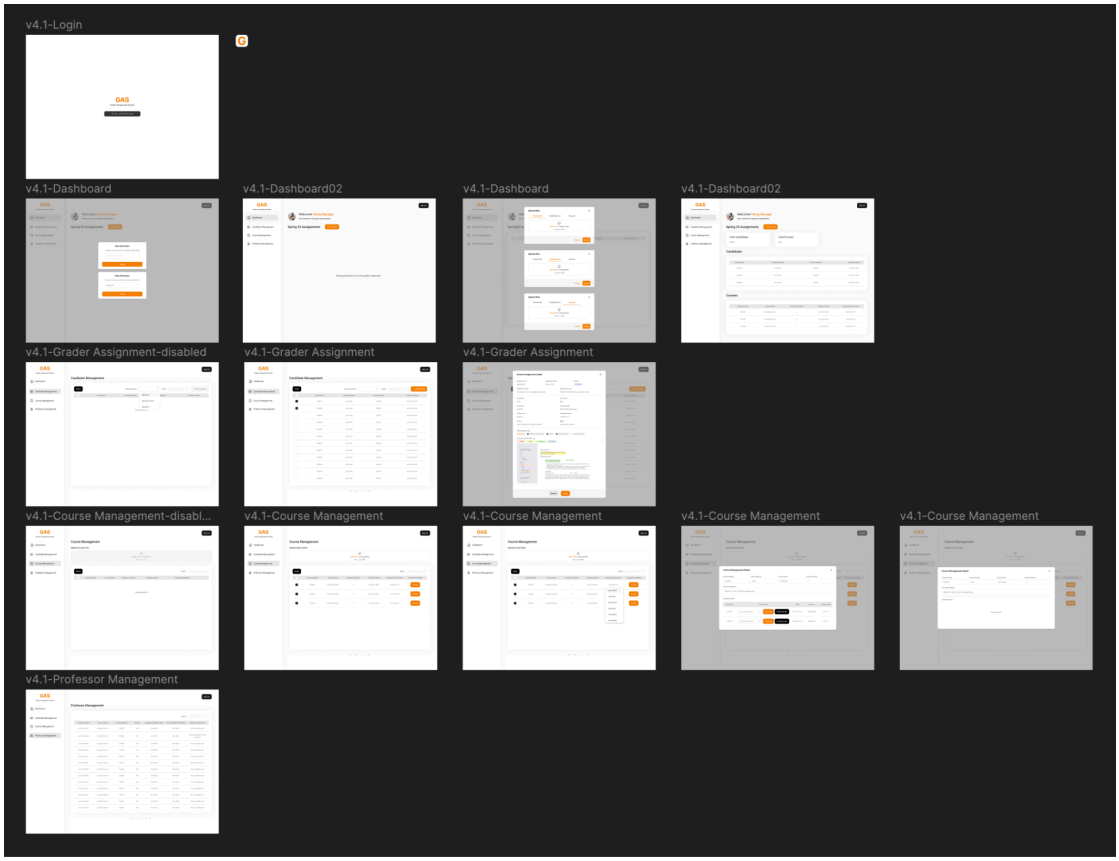
https://www.figma.com/community/file/1153320445661469840/sales-dashboard-design

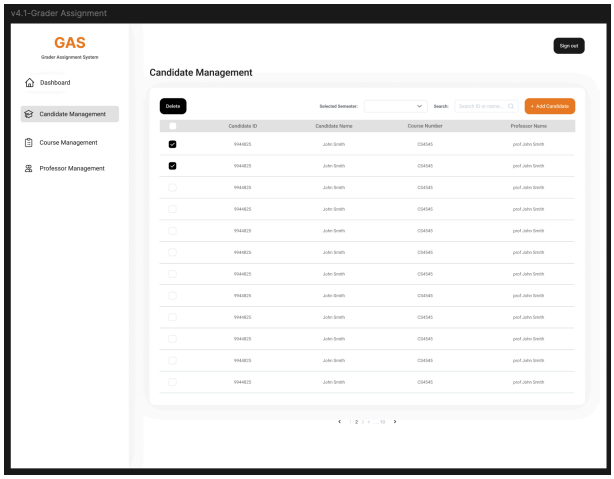
https://react-chartjs-2.js.org/

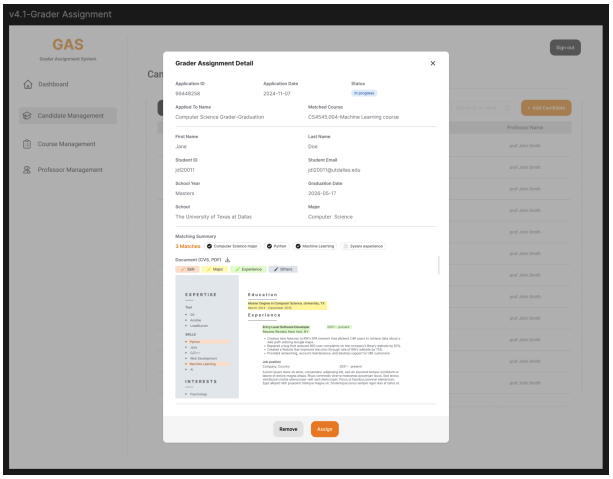
https://axios-http.com/docs/api_intro

Appendices

Figma Design







Postman for testing api