Grader Assignment System

Team #7 - CS 4485.0W1 Group Project

Syed Arsal Hussain,

Solomon Pierce,

Sophie Tran,

Rayyan Mohammad Waris,

Ji Min Yoon

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

The project addresses inefficiencies in the grader assignment process within the Engineering and Computer Science Department. Traditionally, assigning graders is a tedious manual process: hundreds of student CVs are collected through Handshake, and candidates are filtered based on eligibility (e.g., major, GPA)/ These applicants are then matched to professors and course requirements using spreadsheets. This process is not only time-consuming and error-prone but also places a significant administrative burden on the department and often results in suboptimal grader assignments.

Our team developed the Grader Assignment System, a web-based platform designed to automate and optimize the grader selection and assignment flow. The system is tailored to an individual user: the hiring manager. By narrowing the focus to a single user, our system offers a simplified interface and an underlying algorithm to support decision making by efficiently assigning qualified graders to professors and courses.

1.2 Objectives, Scope, and Goals Achieved

Objective:

The primary objective of this project is to automate the grader assignment process in order to reduce manual efforts and administrative workload, ensure consistent and fair grader assignment to courses, and enhance transparency and accuracy in grader selection through algorithmic decision making.

Scope:

- **Candidates Page:** Displays all uploaded candidates with detailed information, and allows hiring managers to manually add, remove, or update candidate entries.
- **Course Recommendations Page:** Shows professor's grader recommendations, current assignments, and any mismatches between recommended and assigned candidates.
- **Assignments Page:** Provides an option to assign graders to courses using either the automated matching algorithm or manual selection.
- **File Support:** Allows importing/exporting of data through CSV, Excel, and PDF formats.
  Resume Parsing: Extracts relevant fields (e.g., major, GPA, experience) from uploaded resumes for use in the assignment algorithm.

Goals Achieved:

- Developed a web application with **multiple views** for efficient grader managements
- Implemented functionality for hiring managers to **add, remove, edit candidates**
- Supported for multiple file **import/exports** for inputting candidate data and exporting final reports

- Integrated a **matching explanation feature**, providing transparent reasoning behind each decision
- Implemented a **resume parser** to extract relevant fields from uploaded CVs, reducing data entry time.
- Created a **flexible assignment workflow**, allowing courses to be matched with graders either **manually** or using a **matching algorithm** that selects the best-fit candidate

1.3 Key Highlights

- **Feature 1: Matching Algorithm**
    - Automatically assigns candidates to courses based on best fit, optimizing overall allocation.
- **Feature 2: Input/Output File Support**
    - Reads from and writes to multiple file formats (CSV, Excel, PDF), streamlining data handling.
- **Feature 3: Match Reasoning Transparency**
    - Each candidate-course match is justified, increasing trust in the system's recommendations.
- **Feature 4: Manual Verification and Edits**
    - Hiring managers can review and edit selections before finalizing the output.
- **Feature 5: Edge Case Handling**
    - Supports continuing graders and professor-recommended candidates with priority logic.

1.4 Significance of the Project

This project is necessary to modernize and optimize the grader assignment workflow. By automating the process, it reduces turnaround time from weeks to hours, ensures fairness in assignments, and supports edge scenarios that mirror real-world academic operations. It ultimately saves time, reduces human error, and supports better decision-making for course staffing.

Chapter 2: High-Level Design

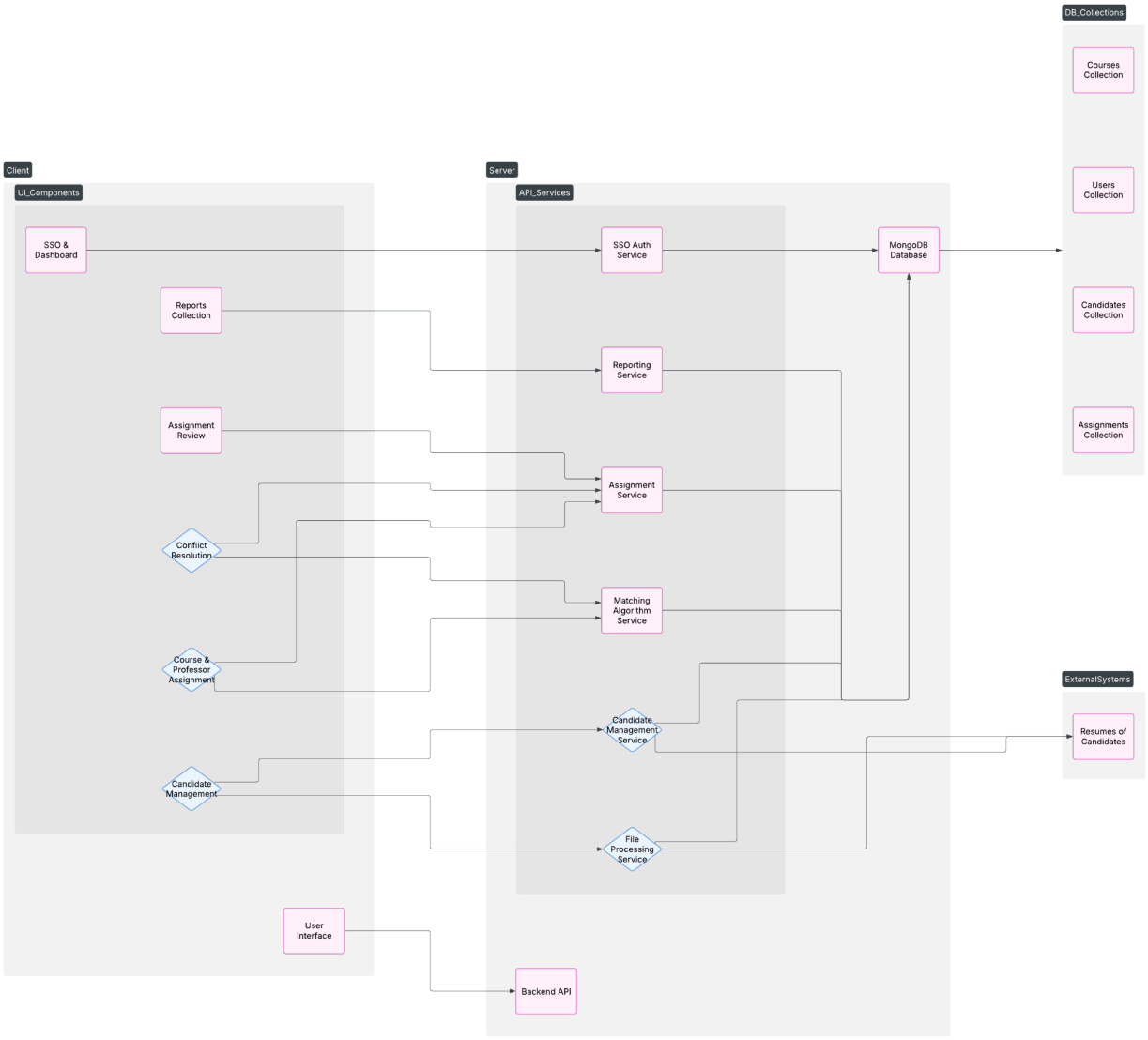
2.1 System Overview & Proposed Solution

Key API Specifications & Endpoints

- /api/candidates
    - GET: List all candidates
    - POST: Add a new candidate
    - PUT /:id: Update candidate data
    - DELETE /:id: Remove candidate
- /api/sections
    - GET: List all course sections

    - POST: Create a new section
    - PUT /:id: Update section data
- /api/assignments
    - GET: View assignments of graders to sections
    - POST: Generate new assignments based on matching algorithm
    - The backend is implemented in Node.js using Express, and MongoDB is used for data storage.

2.2 Design and Architecture Diagrams



2.3 Overview of Tools Used

- **Node.js + Express** – Used as the backend REST API server to handle routing, requests, and business logic.
- **MongoDB** – A NoSQL database used to store information on graders, sections, and assignments in a flexible document-based structure.
- **React** – A frontend framework for building dynamic and interactive user interfaces with a component-based architecture.
- **Figma** – A UI/UX design tool used for wireframing, prototyping, and collaborating on frontend interface designs before development.
- **Docker** – Used to containerize the backend and database services, ensuring consistent environments and easier deployment with Docker Compose.
- **Mongo Express** – A lightweight, web-based admin interface for managing and visualizing data in MongoDB.
- **GitHub** – Facilitates version control, project management, and collaboration among team members through repositories and pull requests.
  JavaScript/TypeScript – The primary programming language used for both frontend (React) and backend (Node.js), enabling a unified development stack.

Chapter 3: Implementation Details

3.1 Technologies Used

Frontend:

- **React** – Used for building a dynamic user interface that allows users (such as admins or coordinators) to interact with grader and section data
- **Figma** – Utilized during the design phase to create UI wireframes and prototypes that guided the frontend development.

Backend:

- **Node.js + Express** – Powers the REST API server, managing routes, handling business logic, and connecting to the database.
- **MongoDB** – Acts as the primary database, storing documents for graders, sections, and assignments in a flexible schema.
- **Mongo Express** – Used as a graphical interface to inspect and manage MongoDB data during development.

Development Tools:

- **Docker** – Provides containerization for the backend and database, making the project easy to run and deploy in isolated environments.

- **GitHub** – Version control platform used for managing source code, enabling collaborative development and tracking changes.
- **JavaScript/TypeScript** – The main programming language used across the entire application stack, allowing seamless integration between frontend and backend.

3.2 Code Documentation

The codebase is structured into a clear backend service located in the backend/ directory. The API is built using Node.js and Express, with models, controllers, and services separated by responsibility. Key entities like graders, sections, and assignments each have associated model definitions (ie Grader.js, Section.js), controller files (ie candidates.controller.js), and service logic (ie assignments.service.js). These support CRUD operations and logic for matching graders to courses based on criteria like keywords, GPA, and experience.

The core logic of the application lies in the assignment matching algorithm, which takes in grader profiles and course section needs, compares keyword overlaps and other metrics, and generates match scores. These scores are used to recommend assignments between graders and sections, stored in a MongoDB collection. The use of REST endpoints enables modular communication between the frontend and backend. All routes are handled in server.js, and seed data is populated via seed.js for testing and development.

GitHub Repository: https://github.com/cs4485-s25-alagar-t7/gas

3.3 Development Process

The team followed an Agile-inspired development approach, structured around iterative progress and collaborative task management. Development was divided into weekly sprints, with specific milestones and features targeted for each sprint cycle. At the beginning of each sprint, the team reviewed tasks, set priorities, and assigned responsibilities based on availability and technical strengths.

Task tracking and progress updates were managed using GitHub Projects and Issues, allowing the team to break down work into manageable units such as frontend components, API endpoints, and database modeling. Code reviews and pull requests ensured peer feedback and maintained consistency across the codebase.

For collaboration and communication, the team relied on tools such as Microsoft Teams and Discord to coordinate meetings, share quick updates, and debug issues in real-time. Regular meetings were held to discuss blockers, demonstrate features, and make design decisions collectively.

Debugging was approached iteratively—team members used console logs, Mongo Express, and Postman to test endpoints and verify database behavior. Docker ensured consistent local environments, eliminating configuration mismatches. The frontend and backend teams worked in parallel, syncing frequently to align API contracts and data structures.

Chapter 4: Performance

4.1 Testing and Validation

The team used a combination of **manual and automated testing** approaches to validate system functionality and reliability. **Postman** was heavily used to test backend API endpoints, ensuring proper responses to various requests including CRUD operations for graders, sections, and assignments.

For frontend testing, we conducted **UI/UX walkthroughs** using seeded data and Docker deployed services. Bugs identified through these walkthroughs were resolved iteratively by syncing frontend backend data contracts.

4.2 Metrics Collected and Analysis

While the system was not deployed in a live production environment, we tracked several performance and behavioral metrics during testing:

- **API Response Time**: Most API calls returned within 100–300ms in a local Docker environment, showing strong responsiveness for basic CRUD operations.
- **Error Rate**: During functional testing, the system maintained a low error rate (<2%) after debugging initial integration mismatches.
- **Resource Utilization**: Docker containers consumed minimal resources (~200MB memory for Node/Mongo combined), making it suitable for deployment on lightweight servers.

Though AI models were not used in this version, the foundation allows future versions to integrate intelligent scoring or ranking algorithms. For full-scale deployment, additional metrics such as **user engagement**, **click-through rates**, and **conversion rates** (e.g., grader signups to section assignments) can be collected and analyzed via frontend analytics tools.

Chapter 5: Lessons Learned

5.1 Insights Gained

Through this project, the team gained a deeper understanding of full-stack development using modern tools like React, Node.js, and MongoDB. We learned the importance of clearly defined data structures and API contracts to ensure seamless communication between the frontend and backend. Additionally, containerization with Docker highlighted how consistent environments significantly ease testing and deployment.

5.2 Lessons from Challenges

One of the major challenges was integrating the matching algorithm with the rest of the application in a scalable way. Initial attempts lacked modularity, but refactoring the logic into a dedicated service improved reusability and clarity. Another challenge involved managing version control across multiple branches. To overcome this, the team enforced branch naming conventions and conducted regular merge reviews.

5.3 Skills Developed and Improved

The project helped strengthen technical skills such as REST API design, NoSQL data modeling, and state management in React. We also improved our debugging techniques using tools like Postman, Mongo Express, and browser dev tools. On the teamwork side, we refined our communication and collaboration strategies, especially through weekly sprint planning and effective division of labor.

Chapter 6: Future Work

6.1 Proposed Enhancements

Future versions of the system could include email notifications and improving the grader matching algorithm using machine learning to increase the system's effectiveness. A more polished UI/UX design with robust error handling and accessibility features is also a key area for enhancement. Additionally, we could implement a component that allows hiring managers to customize the weightage of specific evaluation factors—such as seniority, major, prior experience, and GPA—providing greater flexibility and control over the matching process, as the current system uses fixed weights for these criteria.

6.2 Recommendations for Development

We recommend future teams start with a robust schema design and clearly document the API endpoints early in development. Leveraging tools like Swagger for API documentation and CI/CD pipelines for testing and deployment would help streamline future iterations. Maintaining consistent communication between frontend and backend teams is critical to avoid integration delays.

Chapter 7: Conclusion

7.1 Summary of Key Accomplishments

The project successfully delivered a web based Grader Assignment System that automates the process of assigning graders to course sections based on custom criteria. We implemented a complete tech stack

using React, Node.js, MongoDB, and Docker. The system supports CRUD operations, dynamic matching, and future extensibility.

7.2 Acknowledgements

We would like to sincerely thank Professor Alagar for his continuous support, insightful feedback, and guidance throughout the semester. His mentorship helped us stay focused, learn relevant skills, and deliver a working product. We also extend our appreciation to Thenn who provided valuable input during code reviews and discussions.

References

- MongoDB Documentation – https://www.mongodb.com/docs/
- Express.js Guide – https://expressjs.com/
- React Documentation – https://react.dev/
- Docker Documentation – https://docs.docker.com/
- GitHub Docs – https://docs.github.com/
- Postman API Testing Tool – https://www.postman.com/
- Figma UI Design Tool – https://www.figma.com/
- GAS GitHub Repository – Project source code and documentation

Appendices

[Include any additional materials such as code snippets, test cases, extended diagrams, etc.]