[Grader Assignment System]

[Team Name] - CS 4485.0W1 Group Project

Haeun Kim,

Havish Chaganti,

Sanved Paladhi,

Yosep Ha

Supervisor: Professor Sridhar Alagar

Course Coordinator: Thennannamalai Malligarjunan

Erik Jonsson School of Engineering and Computer Science

University of Texas at Dallas

[5/2/2025]

Table of Contents

Chapter 1: Introduction

v

The Grader Assignment System (GAS) is a web-based application aimed at automating the grader selection process for university courses. The current process is manual, time-consuming, and lacks optimization, leading to inefficiencies in candidate matching. The proposed system will streamline the workflow by automating candidate filtering, optimizing grader-course assignments, and allowing hiring managers to make final adjustments. By reducing processing time and providing transparency in selection criteria, the system improves fairness and efficiency in grader hiring.

1.2 Objectives, Scope, and Goals Achieved

Objectives & Scope

1. Automate the grader selection process to reduce time and manual effort.
2. Use a rule-based matching algorithm to suggest the best grader for each course.
3. Make it easy to see how each match was made and why.
4. Deliver a system that supports file uploads, matching, and manual editing through a web interface.

Core Functionalities

- Automatically filter candidates based on key requirements (like GPA, major, etc.)
- Match graders to courses using a scoring system
- Show clear reasoning behind each match
- Support CSV, Excel, and PDF input/output
- Provide a web interface for uploading, reviewing, and editing assignments

Extra Features

- Let professors mark preferences for certain students
- Handle returning graders who've worked in past semesters

Goals Achieved

- Users can upload candidate resumes and course needs
- Automatic matching works and generates explanations
- Manual edits to assignments are supported
- Final results can be exported as Excel or CSV files
-

1.3 Key Highlights

- Feature 1: File Upload and Processing Interface
  Users can upload candidate resumes (as a ZIP), a list of applicants (Excel), and course-professor requirements (Excel) directly from the frontend.
  Once uploaded, the files are stored and parsed in the backend. A Start Matching button kicks off the matching process using the parsed data.

- Feature 2: Automatic Matching with Reasoning
  After processing, the system displays a table of recommended grader assignments.
  Each row shows the assigned grader, their major, email, and a justification generated by the algorithm (e.g., shared keywords, GPA match, experience).
  Matching is based on a combination of skills, course domains, resume sections, and professor preferences.

- Feature 3: Manual Editing of Assignments
  Users can click Edit on any row to open a popup where they can select a different student from a searchable dropdown.
  When a new grader is chosen, the justification is automatically updated to reflect the manual override (e.g., "Assigned by professor").
  All edits are saved back to the database in real time.

- Feature 4: Result Export and Download
  The final assignment results can be downloaded as a CSV or Excel file using the Download menu.
  This includes both automatically and manually assigned matches with all related fields.

- Feature 5: Responsive Frontend & Backend Integration
  The entire workflow is built using a React frontend and Django backend with MySQL.
  The interface is clean, responsive, and supports real-time updates with filters, sorting, and re-upload support.
-

1.4 Significance of the Project

The main point of this project was to build a system that actually makes the grader assignment process better. Instead of doing everything by hand, this system automates most of it while still letting professors stay in control. It makes the whole process faster, more fair, and more transparent. By clearly showing why a student was matched to a course, it helps take the guesswork out of the decision—and overall just makes things run more smoothly.

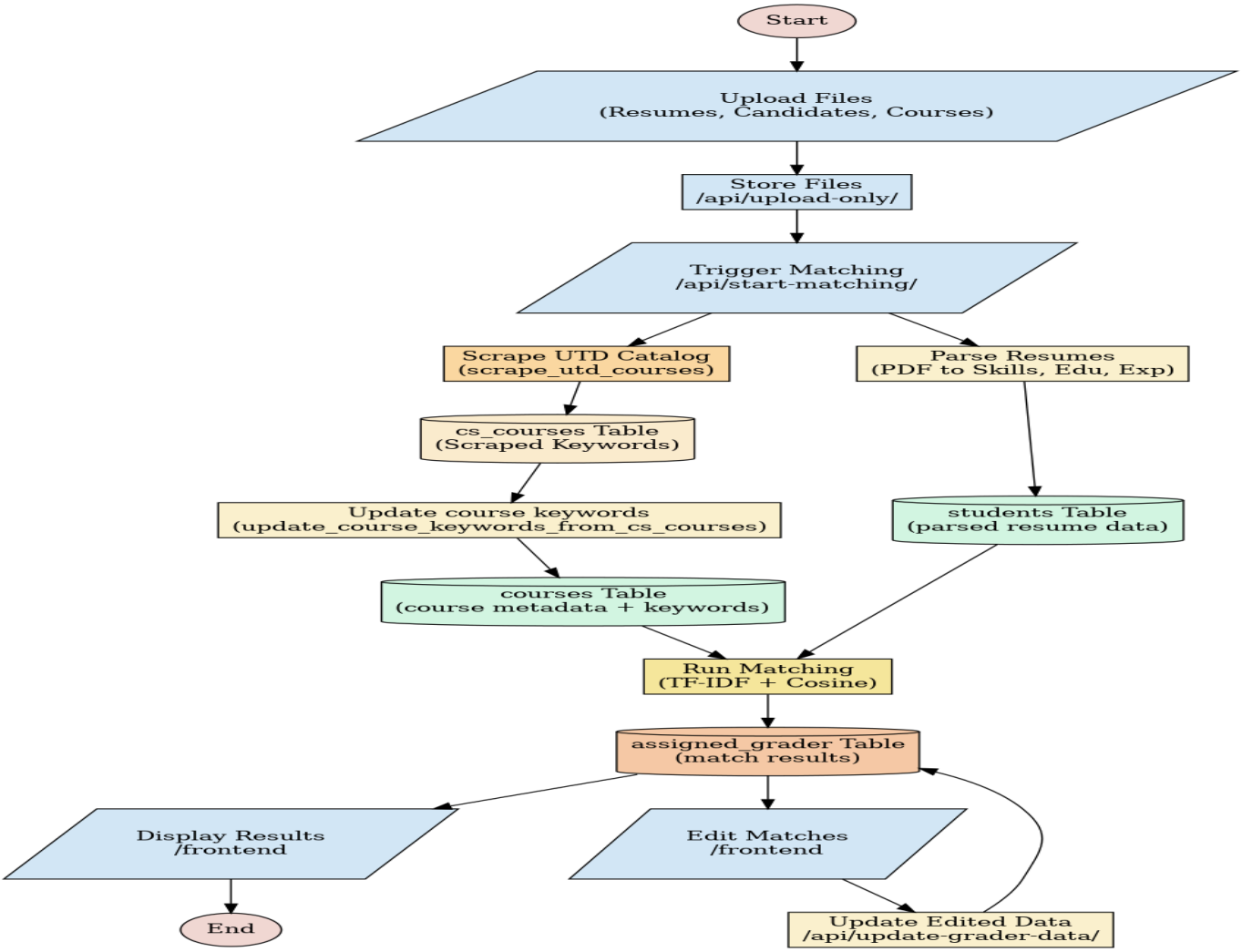
Chapter 2: High-Level Design

2.1 System Overview & Proposed Solution

API specifications

Endpoint	Method	Purpose
/api/process/	POST	Uploads candidate file, course file, and resume ZIP file to the server.
/api/start-matching/	POST	Triggers resume parsing, course keyword enrichment, and grader assignment logic.
/api/match-results/	GET	Retrieves the current grader-course assignments from the database.
/api/update-grader-data/	POST	Sends edited rows (from Manual Edit) back to update the database.
/api/upload-only/	POST	Saves uploaded files (resume/courses/candidates) without processing.
/api/unassigned-graders/	GET	Returns a list of graders who are not currently assigned to any course.
/api/scrape-courses/ *(optional)*	GET or POST	Triggers UTD catalog scraping to update course keywords in the DB.

2.2 Design and Architecture Diagrams [Include system architecture, flow diagrams, and database schemas, ER diagrams.]



Chapter 3: Implementation Details

3.1 Technologies Used

Category	Technology	Purpose / Usage
Frontend	React.js	Building interactive UI (UploadPage, DisplayPage, ManualEditPage)

	Tailwind CSS	Styling UI components with utility-first CSS framework
	Framer Motion	Adding animations and transitions to UI
	XLSX.js	Reading/writing Excel files in the browser
	Ngrok	Exposing local frontend server for remote testing
Backend	Django	Serving REST API endpoints and handling core logic
	Python	Main language for backend development
	PyMuPDF	Parsing PDF resumes to extract structured information
	scikit-learn	TF-IDF vectorization + cosine similarity scoring for matching
Database	MySQL	Storing students, courses, and assignment data
Web Scraping	Selenium	Scraping CS course descriptions from UTD catalog
Development	Visual Studio Code	Code editing and project development
Tools	Postman	API testing during backend development
	Git & GitHub	Version control and code collaboration
	pip + npm	Managing Python and JavaScript dependencies

3.2 Code Documentation

The system works in two main steps. First, the user uploads three files from the frontend — a candidate list, a course-professor list, and a ZIP of resumes. These are sent to the /api/process/ endpoint and just get saved in the backend for now. Once the user clicks the "Start Matching" button, it calls /api/start-matching/, and that's when the actual logic runs. The backend parses all the resumes using PyMuPDF to extract fields like skills, experience, education, and projects. It stores everything into the MySQL database. Courses are also saved, and if keyword data is missing, it scrapes from the UTD catalog using Selenium.

Then it runs the matching algorithm. It compares each course's keywords with the parsed resume data using TF-IDF and cosine similarity. If a professor already recommended a student, it uses that; otherwise,

it assigns the best match to the one who hasn't already been picked. It writes the final assignment, name, major, email, and reasoning, into the assigned grader table. The frontend pulls that data from /api/match-results/ and shows it on the DisplayPage. Manual edits can be made through ManualEditPage and saved back to the database using /api/update-grader-data/.

GitHub Repo: [https://github.com/Hani-124/8-Grader-Assignment-System.git]

3.3 Development Process

We followed a sprint-based Agile approach to keep things organized and iterative. Each sprint focused on a specific goal. For example, setting up file uploads, building the matching logic, or integrating database updates. We tracked tasks using a github project board and broke everything down into manageable chunks like frontend features, backend endpoints, and database setup. Regular check-ins helped us stay aligned and adjust priorities as needed.

For collaboration, we used GitHub for version control and pull requests to review code before merging. Debugging was mostly handled in real time using browser dev tools for the frontend and print/log-based debugging for the backend. Postman was used to test API endpoints quickly. We also used Ngrok to expose the local server when testing frontend-backend integration remotely, which helped a lot during development and demo prep.

Chapter 4: Performance

4.1 Testing and Validation

Integration testing was conducted using Postman to validate API responses and our website to simulate frontend-backend interactions. End-to-end testing was performed by running the frontend and backend on separate machines, connected via ngrok, to simulate real-world usage and test the entire workflow from file uploads to grader assignments.

For debugging, we relied on Chrome DevTools for frontend issues, such as inspecting JavaScript and the DOM, and the VSCode Debugger for backend Python code, particularly for file processing. These tools helped us efficiently identify and resolve issues, ensuring the system worked seamlessly.

4.2 Metrics Collected and Analysis

During the development of the Grader Assignment System, we collected several key performance metrics to ensure the platform's efficiency and reliability. Metrics such as page load time, API response time, and error rate were continuously monitored throughout the testing phase. The backend operations, including file uploads and resume parsing, achieved swift response times, while system stability was maintained by keeping the error rate low through effective exception handling and validations. We also tracked resource utilization metrics, such as CPU and memory usage, during high-load scenarios to confirm that the system could handle concurrent file processing without performance degradation.

On the user experience front, we concentrated on monitoring interaction patterns and the usability of core features, including uploading resumes, viewing assignment results, and editing grader details. Although no formal user analytics were implemented, we conducted consistent testing using our web interface, which validated that the workflows were intuitive and functional. These insights informed our design decisions and confirmed that the interface provided a seamless end-to-end experience for users interacting with both frontend and backend components.

Chapter 5: Lessons Learned

5.1 Insights Gained

Through the development of our project, we gained a deeper understanding of the full software development lifecycle, particularly how backend logic, frontend design, and database management must work cohesively. We also realized the importance of clear communication and modular design to ensure maintainability and scalability.

5.2 Lessons from Challenges

One of the key challenges we faced was integrating frontend and backend components across different environments using tools like Ngrok. We also encountered parsing inconsistencies with varied resume formats and had to iterate on our algorithm to ensure accurate student-course matching. These challenges were addressed through debugging sessions, improved error handling, and code modularization for cleaner architecture.

5.3 Skills Developed and Improved

Throughout the project, we improved our technical skills in React, Django, MySQL, and data processing techniques such as TF-IDF and cosine similarity. On the team side, we developed better collaboration

habits using GitHub for version control, conducted integration testing effectively, and learned how to manage tasks and roles within a group setting.

Chapter 6: Future Work

6.1 Proposed Enhancements

There are a bunch of features we didn't have time to build but would definitely make the system better:

- **Smarter Matching**: Right now, we're using a TF-IDF based approach that works well, but it could be improved by adding NLP models like BERT or even training something lightweight to understand resumes more contextually.

- **Better Resume Parsing**: The current parser handles most PDFs okay, but it struggles with inconsistent formats. In the future, we could use layout-aware parsing or improve section detection to pull out more detailed experience/project info.

- **Admin Dashboard**: A real admin view would be helpful for reviewing uploads, tracking system status, and managing data from one place.

- **User Permissions**: Right now, everyone has full access. Adding roles (admin, viewer, professor, etc.) would prevent accidental edits and keep things cleaner.

- **File Validation & Error Messages**: Uploading the wrong file or format can break stuff. More helpful error messages and validation would make the system more user-friendly.

- **Email Notifications**: Sending automatic emails to assigned students or professors when results are finalized would help close the loop.

6.2 Recommendations for Development

If someone picks this up in the future, here's what I'd recommend:

- **Keep It Modular**: Breaking up code into logical pieces (e.g., parsing, matching, database functions) made it way easier to debug and test. Stick with that structure.

- **Avoid Hardcoding**: Use environment variables and config files instead of hardcoding things like file paths or database names. It'll save you a lot of pain later.

- **Add API Docs**: Swagger or even a Postman collection would help anyone trying to work with the backend without digging through all the views.
- **Make the UI Smoother**: The core functionality works, but there's room to improve things like making uploads more intuitive or adding inline editing instead of popups.
- **Think About Scaling**: If other departments ever want to use this, we'll need support for multiple roles, filtered data views, and possibly multiple pipelines.

Chapter 7: Conclusion

7.1 Summary of Key Accomplishments

Chapter 7: Conclusion

7.1 Summary of Key Accomplishments

Over the course of this project, we were able to build a fully working system that automates the grader assignment process from start to finish. Some of the main accomplishments include:

- Built a full-stack web app with a React frontend and Django backend
- Enabled uploading of resumes, candidate lists, and course info directly from the browser
- Parsed and processed resumes to extract skills, experience, and education
- Matched candidates to courses using a keyword-based scoring algorithm
- Generated justifications for each match to provide transparency
- Created a manual edit feature so professors can override matches easily
- Supported exporting the final results in Excel/CSV formats
- Successfully stored all relevant data in a MySQL database with clean integration across all components

Overall, we were able to meet our goals and deliver a system that works smoothly end-to-end.

7.2 Acknowledgements

We'd like to sincerely thank Dr. Alagar for guiding us throughout the course and project. His support and clear structure gave us a solid foundation to work from and helped keep us on track.

A special thank you goes to our TA, Malligarjunan Thennannamalai, who helped us at every step — from debugging tricky issues to reviewing our implementation and making sure we stayed focused on the project goals. His feedback, responsiveness, and technical help were absolutely essential, and we really appreciate the time he dedicated to supporting us.

References

1. Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). *ImageNet classification with deep convolutional neural networks*. Advances in neural information processing systems, 25.

2. Scikit-learn Documentation. *TF-IDF Vectorizer*. https://scikit-learn.org

3. PyMuPDF (fitz) Documentation. *PDF parsing and text extraction*. https://pymupdf.readthedocs.io

4. Django REST Framework Docs. https://www.django-rest-framework.org

5. MySQL Documentation. https://dev.mysql.com/doc/

6. ReactJS Official Docs. https://reactjs.org

7. Tailwind CSS Docs. https://tailwindcss.com

8. UTD Course Catalog – Computer Science. https://catalog.utdallas.edu

9. NLTK Stopword Filtering. https://www.nltk.org

10. OpenAI ChatGPT — Used for code assistance and explanation guidance throughout the project.