International Applicant Rating Algorithm

Project Solution Approach

International Programs

We bring WSU to the world, and the world to WSU.

WSU International

25-FA25-SP26-IAOWIP-GEN

Steven Bennett, Khushi Panchal

10/20/2025

Table Of Contents

I. Introduction

The International Applicant Rating Algorithm is a system developed for Washington State University (WSU) to enhance the evaluation of international student applications. Currently, admissions staff must rely heavily on manual review of data from Slate, including GPA, travel history, curriculum, applicant essays, and other risk factors. This manual process is time-consuming, susceptible to inconsistencies, and often delays decision-making, making it challenging to apply the admissions rubric fairly and efficiently across large applicant pools.

The primary goal of the International Applicant Rating Algorithm is to automate and standardize the applicant scoring process while maintaining fairness and transparency. The system calculates categorical scores and combines them into a composite rating, allowing admissions counselors to compare applicants more effectively. Essays are analyzed using AI, with alternative methods available in cases where AI services are inaccessible. The system generates detailed score breakdowns and exportable reports, ensuring that admission decisions are transparent and easily defensible.

Future iterations will expand integration with Slate, enhance AI-assisted document processing and fraud detection, and provide administrators with the ability to adjust scoring rules through configuration files rather than direct code changes. By combining automation with human oversight, WSU can increase efficiency, reduce bias, and build a scalable system to manage international admissions effectively.

II. System Overview

The International Applicant Rating Algorithm is designed to help WSU's admissions staff evaluate international student applications efficiently, accurately, and consistently. It integrates automated data processing, essay evaluation, and fraud detection tools within a single web-based platform.

The system follows a layered architecture consisting of three primary components: the Flask Web Interface, which handles user interactions and input; the Application Logic Layer, which performs scoring, essay analysis, and fraud detection; and the Data Layer, which manages local storage and report generation. Together, these layers ensure smooth data flow from user input to processed results and exported reports.

Specialized Python libraries and APIs are used to perform the system's analyses. Essays are evaluated using the Hugging Face API, which applies natural language processing models to assess tone, clarity, and structure. Fraud detection employs OCR and computer vision tools such as Tesseract, PyPDF, and OpenCV to identify anomalies in financial statements and transcripts.

The system is modular, with separate components for scoring, fraud checks, and report exporting, each operating independently while sharing data seamlessly. This modular design enables future integration with WSU's admissions system (Slate) and allows for the addition of more advanced AI-powered fraud detection using tools such as the OpenAI API.

III. Architecture Design

A. Overview

The International Applicant Rating Algorithm uses a layered architecture that separates responsibilities into distinct modules, making the system maintainable, scalable, and easy to update. This design allows components such as essay scoring, fraud detection, and applicant evaluation to function independently while supporting transparent, explainable, and modifiable results.

The system has three primary layers. The User Interface Layer, built with Flask, handles applicant data input, document uploads, and displays evaluation results, including scores and fraud assessments. The Application Logic Layer performs core functions, integrating scoring algorithms, AI-powered essay analysis, and fraud detection to produce reliable outputs. The Data Layer manages storage and retrieval of files and results, currently using local files and in-memory objects, with future integration planned for WSU's Slate system.

Each subsystem communicates through well-defined interfaces. The scoring service aggregates metrics such as GPA and essay quality, the essay analysis subsystem uses Hugging Face models to evaluate clarity and tone, and the fraud detection subsystem employs OCR and computer vision to identify document anomalies. The Export Service compiles results into structured reports for review or download.

This layered and modular architecture provides a robust, flexible foundation for automating complex admissions evaluations while maintaining transparency, adaptability, and room for future enhancements.

B. Subsystem Decomposition

1. Flask Web Interface

Description
The Flask Web Interface is the main user-facing layer of the system. It handles all interactions with admissions staff, including input of applicant data, essay submissions, and uploaded supporting documents. It also displays evaluation results such as applicant scores, essay analysis feedback, and fraud detection summaries.

Concepts and Algorithms Generated
The interface uses Flask routing to handle requests and validate incoming data. It communicates with backend services, such as scoring, essay analysis, and fraud detection, through internal function calls. JSON is used for structured data exchange between subsystems.

Interface Description
Services Provided:

Service Name	Provided To	Description
Data Upload	Application Logic Layer	Accepts applicant information, essays, and documents through forms or API endpoints
Result Display	End User	Shows processed evaluation reports, fraud summaries, and downloadable results

Services Required:

Service Name	Service Provided
Scoring Service	Application Logic Layer
Essay Analysis Service	Application Logic Layer
Fraud Detection Service	Application Logic Layer
Export Service	Application Logic Layer

2. Scoring Service

Description
The Scoring Service evaluates applicant data to generate an overall rating. It considers GPA, essay results, and other academic or behavioral metrics, producing a standardized numerical score.

Concepts and Algorithms Generated
A rule-based scoring algorithm calculates weighted metrics and normalizes them to produce a score between 0 and 100. This structure allows future integration of AI-based scoring models for more adaptive evaluation.

Interface Description
Services Provided:

Service Name	Provided To	Description
Applicant Scoring	Flask Interface	Computes overall applicant scores and returns them to the user interface

Services Required:

Service Name	Service Provided
StudentAnalyzer (Essay Analysis)	Essay Analysis Service
Data Layer	Local storage and retrieval

3. Essay Analysis Service

Description
The Essay Analysis Service evaluates applicant essays using machine learning models hosted on **Hugging Face**. It assesses tone, clarity, coherence, and relevance, contributing to the overall applicant rating.

Concepts and Algorithms Generated
The subsystem sends essay text to the Hugging Face Inference API. Pretrained NLP models generate embeddings and confidence scores across linguistic dimensions such as sentiment, grammar, and structure. These results are normalized and weighted to calculate a composite essay score, without the need for in-house model training.

Interface Description
Services Provided:

Service Name	Provided To	Description
Essay Evaluation	Scoring Service	Processes essay text and returns normalized sub-scores for clarity, structure, and relevance

Services Required:

Service Name	Service Provided
Hugging Face API	External API for text inference
API key configuration	Internal authentication

4. Fraud Detection Service

Description
The Fraud Detection Service identifies potentially fraudulent or manipulated documents, such as financial statements or academic transcripts. It analyzes PDFs and images for inconsistencies, missing metadata, abnormal layouts, or suspicious patterns.

Concepts and Algorithms Generated
This subsystem combines rule-based checks with OCR and computer vision. Libraries such as PyPDF, pdfplumber, PIL, pytesseract, and OpenCV are used to extract text, detect anomalies, and assign a fraud likelihood score. While currently local, future versions may integrate OpenAI's multimodal API for enhanced fraud detection.

Interface Description
Services Provided:

Service Name	Provided To	Description
Document Analysis	Flask Interface	Performs OCR and structural analysis on uploaded documents, returning fraud likelihood scores

Services Required:

Service Name	Service Provided
PyPDF, pdfplumber, PIL, pytesseract, OpenCV	Local libraries
OpenAI API (Optional)	Advanced document verification

5. Export Service

Description
The Export Service organizes and saves system outputs in structured formats such as CSV or XLSX for review, archiving, or sharing. It ensures that scores, essay evaluations, and fraud detection results can be easily exported.

Concepts and Algorithms Generated
The service collects results from other components, formats them into structured reports, and saves them locally. Future integration with WSU's Slate system will enable direct uploading of applicant evaluations for real-time review.

Interface Description
Services Provided:

Service Name	Provided To	Description
Result Export	Flask Interface	Combines scoring, essay, and (future) fraud results into downloadable reports with applicant IDs and timestamps

Services Required:

Service Name	Service Provided
Scoring Service	Application Logic Layer
Data Layer	Local storage
(Future) Fraud Detection Service	Application Logic Layer

6. Data Layer

Description
The Data Layer manages file storage and retrieval for the system. Currently, it uses local files and in-memory objects to handle essays, transcripts, financial documents, and processed results. It provides a centralized interface for reading, writing, and exporting data across subsystems.

Concepts and Algorithms Generated
Data is temporarily stored in memory for aggregation and then written into structured formats (JSON, CSV, XLSX) for persistence. The layer ensures smooth communication between processing services and the export system.

Interface Description
Services Provided:

Service Name	Provided To	Description
Data Read/Write	Essay Analysis, Fraud Detection, Application Logic Layer	Handles opening, reading, and writing of local files (PDFs, images, text) for intermediate and final outputs
Export Handling	Application Logic Layer	Saves processed results and reports in portable formats for user access

Services Required:

Service Name	Service Provided
Fraud Detection and Essay Analysis	Internal modules
Data Layer	os, json, io, etc.

IV. Data design

The current version of the International Applicant Rating Algorithm uses a file-based data design rather than a traditional database. This approach simplifies early development and allows local testing without requiring database setup or network connectivity. Data is processed and stored using local files and in-memory Python objects, with standardized formats to ensure consistency and portability.

The system's data is primarily handled by two components: Local File Storage and In-Memory Objects. The details of each component are explained below.

Local File Storage
Local File Storage is used to store applicant documents and processed results. Each file corresponds to a specific type of data and is organized into structured directories:

- /data/financial/authentic – authentic financial documents
- /data/financial/fraudulent – potentially fraudulent financial documents
- /data/applicants/essays – submitted essays

Processed outputs from these files are saved as structured JSON reports. For example, the Fraud Detection Service generates fraud_eval_report.json, which contains scores and metadata for each document analyzed, such as missing information, layout anomalies, and fraud likelihood.

In-Memory Objects
In-memory Python objects are used to temporarily store and manipulate applicant data during processing. For instance, the Essay Analysis Service creates dictionaries containing essay text and normalized scores from the Hugging Face API. These objects are passed to the Application Logic Layer, which aggregates essay results, applicant data, and future fraud scores into a unified evaluation report.

Data Flow and Export
Once processing is complete, the combined results can be exported as .csv or .xlsx files for review by admissions staff. Each exported file includes applicant identifiers, scores, essay analysis results, and (in future versions) fraud evaluations.

Scalability and Future Database Integration
Although the current design uses files and memory, it is structured to support future migration to a database-backed system. For example, outputs could be mapped directly to database tables with fields for applicants, essays, and fraud results. This design ensures a smooth transition when integrating with WSU's Slate system, enabling automated storage and retrieval of applicant data. By combining local files and in-memory data structures, the system ensures modularity, clarity, and portability, while also laying the groundwork for future database integration.

V. User Interface Design

The International Applicant Rating Algorithm provides a clean and intuitive web-based interface built using Flask, designed specifically for WSU admissions staff. The interface focuses on simplicity, clear navigation, and quick access to automated evaluation results. The design emphasizes minimal text input, structured workflows, and immediate visibility of scores and reports, making it easy for staff to upload applicant data, run analyses, and review results efficiently.

The main pages of the interface include the Login Page, Student Scoring Page, Analytics Dashboard, and Batch Processing Page. The Login Page ensures secure access, restricting functionality to authorized admissions staff. The Student Scoring Page acts as the core workspace, allowing staff to enter applicant data, including GPA, essays, and supporting documents—and immediately view automated scoring results. The Analytics Dashboard summarizes trends across applicants, such as average scores, subcategory analysis, and potential risk factors. The Batch Processing Page supports uploading multiple applicant records at once via .csv files, with the system generating consolidated .xlsx or .csv output files for easy review.

The interface incorporates accessibility and usability best practices, including clearly labeled buttons for actions such as Upload, Analyze, and Export, strong color contrast, and structured layouts. This minimizes errors, reduces training time, and ensures that even large batches of applicants can be processed efficiently.

Use cases are directly integrated into the interface:

- **Input Student Profile Data:** Users enter demographic, academic, and behavioral information via text fields and dropdowns.
- **Analyze Student:** Clicking the Analyze Student button triggers automated scoring and essay evaluation using the backend services.
- **View Essay Statistics and Preview Essays:** Users can review essay metrics (word count, readability) and preview formatted essay text.
- **Clear Essay Text:** A Clear button resets fields for new input.
- **Select Risk Factors:** Checkboxes allow counselors to flag financial or behavioral issues, which affect scoring calculations.
- **Verify Document Authenticity (Future Integration):** The interface will allow uploading of financial or transcript documents for fraud detection, displaying likelihood scores alongside other evaluation results.

The design is modular and scalable, with future updates planned to integrate the Fraud Detection subsystem and enhance batch processing and reporting. The current interface prioritizes ease of use, clarity, and immediate feedback for admissions staff, while supporting future expansion and advanced AI-assisted workflows

VI. Constraints

The development of IARA faced several technical, ethical, and resource-based constraints that shaped its final design. Each limitation required trade-offs to maintain system functionality, fairness, and usability within the project's timeline and scope.

Constraint Type	**Specific Limitation**	**Design Trade-off / Adaptation**
Time / Schedule	Limited development time within the semester	Focused on core pipeline (data input, rating, visualization) and postponed advanced features like ML model tuning
Data Privacy	Sensitive applicant and academic data	Enforced anonymized records and role-based access for authorized counselors only
Ethical / Bias Control	Avoiding algorithmic bias in applicant rating	Incorporated manual review checkpoints and transparent scoring logic
Performance	Need for real-time feedback during applicant rating	Optimized backend processes and caching to reduce latency
Accessibility	Platform must be usable by all staff members	Followed WCAG 2.2 standards to improve text readability and keyboard navigation
Budget	No funding for premium APIs or hosting	Used open-source libraries and free-tier hosting options
Maintainability	System must support future updates	Used modular architecture for easier integration of future AI models

VII. Glossary

Flask: A Python web framework used to build the International Applicant Rating Algorithm web interface and route user input to backend logic.

OCR (Optical Character Recognition): A technology that converts scanned or image-based text into machine-readable data.

OpenAI API: A future integration for fraud detection, providing multimodal AI models capable of analyzing text and images.

Hugging Face API: A cloud-based platform providing pre-trained machine learning models for natural language processing tasks like essay scoring.

JSON: A text-based format used to store and exchange structured data between components

CSV/XLSX: File formats for tabular data exports used to share applicant reports

API Key: Credential used to authenticate and authorize API requests.

PyTesseract: Python library that performs OCR by wrapping Google's Tesseract engine.

OpenCV: Open-source computer vision library used for detecting visual document anomalies

Slate: WSU's admissions management system, planned for future integration.

VIII. References

[1] S. Hoffstaetter and M. Lee, *"pytesseract 0.3.13 – Python-tesseract: A Python wrapper for Google's Tesseract-OCR engine,"* Python Package Index (PyPI), Aug. 15, 2024. [Online]. Available: https://pypi.org/project/pytesseract/

[2] *"OpenCV-Python Tutorials: Introduction to OpenCV,"* OpenCV Documentation, version 4.13.0-dev, Oct. 20, 2025. [Online]. Available: https://docs.opencv.org/4.x/d6/d00/tutorial_py_root.html

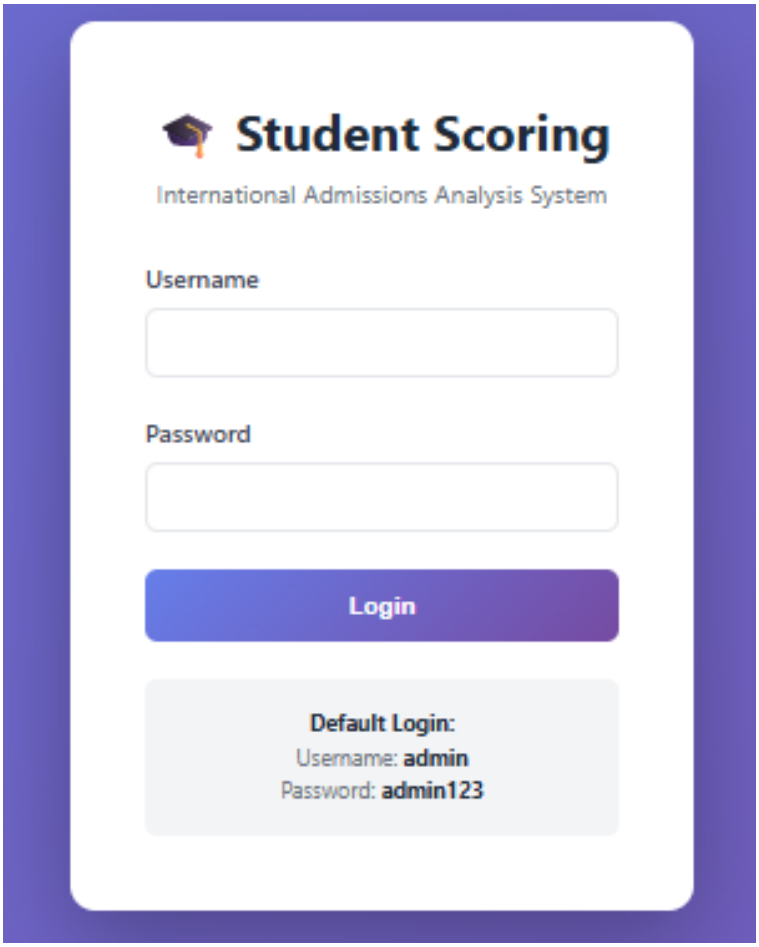
[3] F. Nurfikri and B. Whitfield, *"How to Build Optical Character Recognition (OCR) in Python,"* Built In, Apr. 9, 2025. [Online]. Available: https://builtin.com/data-science/python-ocr

[4] *"OpenAI API Reference: Introduction,"* OpenAI Documentation, 2025. [Online]. Available: https://platform.openai.com/docs/api-reference/introduction?lang=python

VIII. Appendix A- User Interface Screenshots

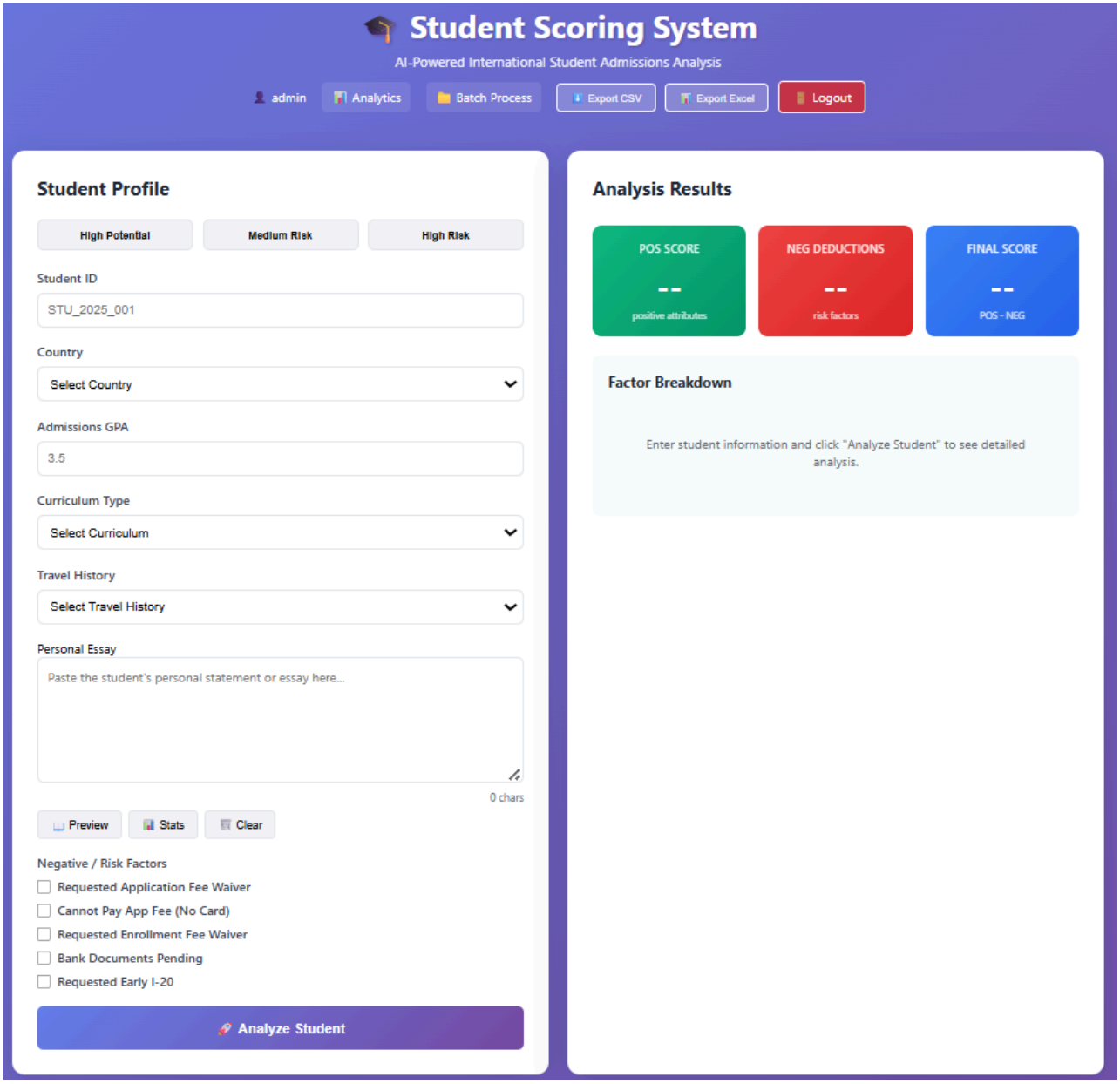
A. Login Interface

Figure 1: Allows administrative users to securely log into the International Applicant Rating Algorithm system before accessing any tools or applicant data.



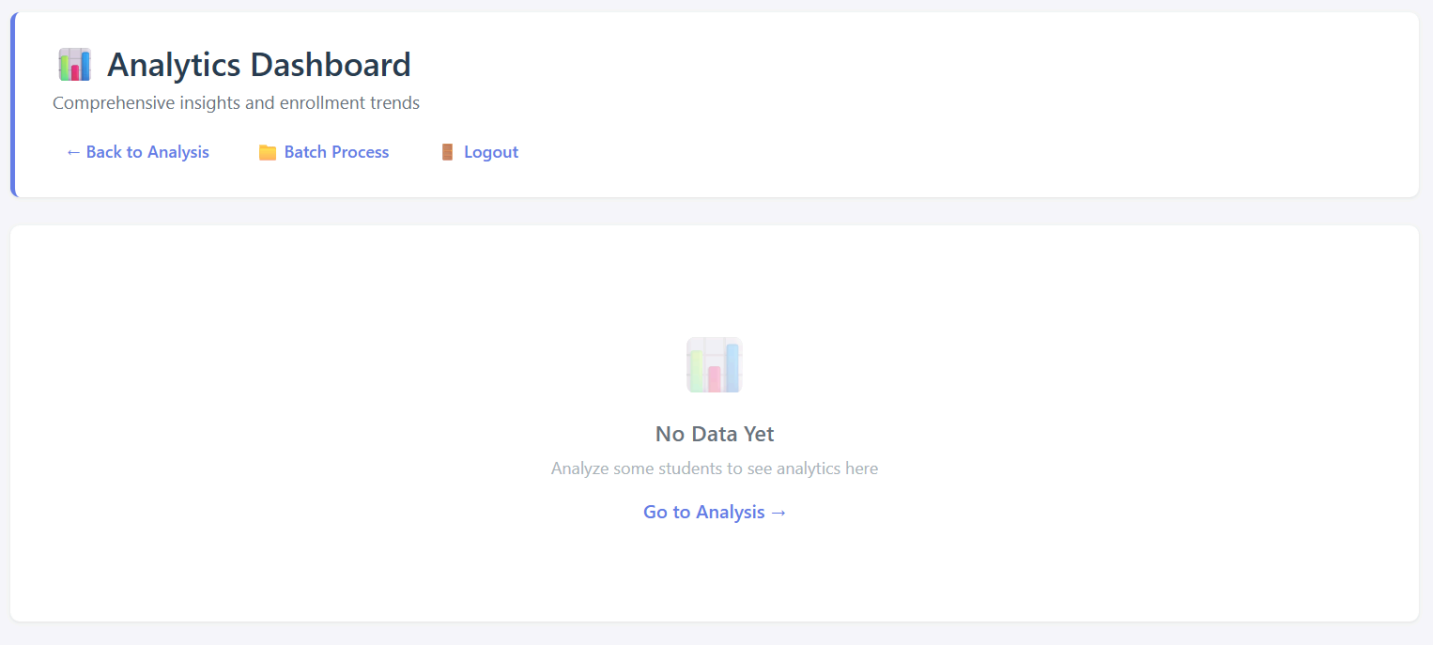
B. Student Scoring Page

Figure 2: The primary workspace where admissions staff enter applicant information, including GPA, essays, and supporting documents. The interface displays real-time scoring results, sub-scores, and composite ratings.



C. Analytics Dashboard

Figure 3: Provides a summary of trends and insights across multiple applicants, including average ratings, risk factor summaries, and overall comparisons within the applicant pool.



D. Batch Processing Interface

Figure 4: Supports bulk operations, allowing staff to upload .csv files containing multiple applicants. The system processes all entries, calculates scores, and outputs consolidated .xlsx or .csv reports for download.