# EARLY INTERVENTION MANAGEMENT SYSTEM FOR HIGHER EDUCATION


Akash Wihaga Gunathilaka

p2783610

## Table of Contents

# Abstract

Universities now gather large amounts of data from online learning platforms about how students interact with course materials, submit assignments, and participate in classes. The data has clear potential for identifying students who are starting to struggle, while there is still time to help them. However, most research in learning analytics focuses on building prediction models, with far less attention given to how tutors could actually use these predictions in their daily work.

The projects aimed to bridge the gap by developing a complete early intervention system. Using the OULAD dataset, I trained a XGBoost model to predict students at risk of poor performance. A full web application was built around this model, using react with TypeScript for the frontend,bFastAPI for the backend and PostgreSQL with SQLAlchemy for the database.

The system was developed iteratively, starting with the machine learning model, followed by the backend and the database, and finished with the frontend. The model performed reasonably well on the dataset, and the complete system was evaluated in terms of usability, functionality, security and ethical considerations.

This project, demonstrates how predictive models can be turned into practical tools that university staff can actually use, rather than leaving them alone as standalone research experiments. Although this is only a prototype, it offers a strong foundation for future work using real institutional data, improved explainability, and better integration with existing learning management systems.

# 1. Introduction

Higher education institutions now generate large amounts of data through digital learning platforms. Every time students log in, watch lectures, submit assignments, or interact with course materials, that activity is recorded. This data creates a real opportunity to identify students who are starting to fall behind before it becomes a serious problem.

In the fields of learning analytics and educational data mining, researchers have spent years trying to predict student performance using behavioural and academic indicators. The goal is simple: catch problems early so universities can offer support when it can still make a difference. However, most existing studies focus heavily on building and testing prediction models in isolation. Many achieve impressive accuracy, but few actually turn those models into tools that lecturers and tutors can use in their day-to-day work.

This project tries to bridge that gap. I designed and built a full-stack web application that combines machine learning predictions with practical features academic staff can use such as student search, risk dashboards, detailed profiles, and the ability to record interventions.

## 1.1 Project aim

The aim of this project was to design and build a full-stack web application that uses machine learning to identify students who are at risk of poor academic performance, and to provide academic staff with practical tools to support them early.

Specifically, the aim is to develop an early intervention system that combines an XGBoost prediction model with a user-friendly web interface. The system will allow staff to search for students, view risk predictions along with their contributing factors, and record any interventions they make.

## 1.2 Project Objectives

To achieve this aim, the project will pursue several key objectives:

1. Conduct a literature review on learning analytics, student performance prediction, and early intervention systems in higher education
2. Analyze publicly available student datasets and prepare them for machine learning applications through preprocessing and feature engineering.

3. Develop and evaluate machine learning models capable of predicting student academic risk levels based on engagement and academic indicators.
4. Design and implement a backend system capable of handling data management, model predictions and application logic.
5. Develop a web-based dashboard that allows users to monitor student performance, visualize risk levels and explore engagement trends.
6. Implement basic rule-based intervention suggestions to assist academic staff in responding to identified risks.
7. Evaluate the system in terms of predictive performance, usability, ethical considerations, and practical limitations.

## 1.3 Motivation for the project

Student retention and success are major challenges for universities. Too often, students only get meaningful support after they have already failed an assignment, missed multiple deadlines, or disengaged completely. By that point, it is much harder to turn things around.

This issue became personal for me during my own time at university. I saw friends struggling quietly for weeks or even months before anyone noticed. Some barely passed their modules, while others ended up dropping out. These experiences made me realise how valuable and early warning system could be.

At the same time, I was learning about machine learning and saw its potential to analyze patterns in student engagement and assessment behavior. While there is already good research in this area, most of it stops at producing model outputs. I wanted to go further and build something that university staff can use.

## 1.4 Project Scope

The scope of this project focuses on the design and implementation of a prototype early intervention system that integrates machine learning predictions within a web-based application. It uses the publicly available OULAD dataset, the system allows staff to view student data, generate risk predictions, understand contributing factors, and log interventions.

The project will involve several components, including machine learning model development, backend system implementation, database integration, and frontend dashboard development. However, the system is intended as a prototype research project rather than a fully deployed institutional solution. The work will use publicly

available and synthetic datasets to ensure compliance with ethical and privacy considerations, and it will focus on demonstrating the feasibility and effectiveness of integrating predictive analytics within an educational support system.

## 1.5 Personal Development Objectives

Beyond the project itself, this work aligned well with my longer-term goal of working in machine learning AI. Through this project I can gain hands-on experience in:

- Data preprocessing and feature engineering
- Training and evaluating ML models (XGBoost)
- Building a full-stack application (React + FastAPI + PostgreSQL)
- Integrating machine learning models within a web application

It also pushed me to improve research, technical writing and project management skills which I believe will be valuable going forward.

Overall, this project provides an opportunity to combine research, software engineering, and machine learning development in a way that supports both academic achievement and long-term career aspirations in the machine learning field.

# 2. Literature Review

## 2.1 The Role of Technology in Improving Education

Over the past few years, higher education has changed a lot because of the rise of online learning platforms. Every time a student logs in , watches a lecture and submits an assignment the activity gets recorded. This has given universities a huge amount of data on how students actually behave and engage with their course.

This shift has moved universities away from the old way of doing things — waiting until a student fails an exam or stops showing up before offering help — towards something more proactive. Instead of reacting to problems, staff can now try to spot them early using data.

There are two main research areas looking at this: Educational Data Mining (EDM), which focuses more on the technical side of building models, and Learning Analytics (LA), which is more interested in how these insights can actually help teaching and learning in practice. This distinction is supported by Papamitsiou and Economides (2014), who describe LA and EDM as related data-driven approaches for prediction, feedback and educational decision-making.

A systemic review by Albreiki et al. (2021) showed that machine learning can produce some promising results for predicting student performance and dropout risk. However, they also pointed out a major problem in the field: lots of researchers focus heavily on improving model accuracy, but very few build systems that lecturers and tutors can actually use in their daily work. This gap is what made me want to build a complete working system rather than just another prediction model.

## 2.2 Early approaches: traditional machine learning and static data

The earlier studies were quite limited. For example, Kovacic (2010) tried using classification trees on basic enrolment data like demographics and course choices. The accuracy was only around 59-60%, and the conclusion was clear: background information alone is not enough. This shows the need for richer datasets and more advanced models to improve prediction and enable effective early intervention.

Later, researchers started trying more algorithms. Sekeroglu et al. (2019) tested models like neural networks and support vector machines. The results improved, but they were still mostly working with static features that didn't really capture how a student's behavior changes during the semester.

A more recent study by Ahmed (2024) reported very high accuracy (96%) using SVM, however it included the final result as an input feature, which creates data leakage and makes the results unrealistic.

These early studies demonstrate that machine learning can predict student performance, but they also reveal key limitations:

- Reliance on static data
- Limited feature engineering
- Potential methodological flaws such as data leakage

These limitations highlight the need for better approaches.

## 2.3 Evolution Towards Advanced Models

As research into student performance prediction developed, many researchers moved away from simple model towards more advanced ensemble methods.

Early studies typically used basic algorithms such as decision trees, Naïve Bayes, and logistic regression (Albreiki et al., 2021). While these provided a decent starting point, they often struggled with educational data because they relied heavily on static features and made quite strong simplifying assumptions (Fei and Yeung, 2015).

More recent work has focused on ensemble techniques like Random Forest, Gradient Boosting, and especially XGBoost. These approaches have consistently outperformed traditional single models in several studies (Jha et al., 2019). The main advantage is that they combine multiple weaker models to create a stronger overall prediction. Instead of relying on one model, they learn from previous errors and gradually improve accuracy (Chen and Guestrin, 2016).

This is particularly useful in education, where student behaviour is complex — factors like engagement, assessment scores, and login patterns all influence each other in non-linear ways. However, these gains in accuracy come with a downside. Ensemble models are often seen as ‘black boxes’, making them harder to interpret (Marquez-Vera et al., 2016). In an educational context, where lecturers need to understand why a student is flagged as at risk, this lack of transparency is an important limitation.

For this project, I chose XGBoost as the main model. It offers strong predictive performance while also providing feature importance scores, which help explain which factors are contributing most to a student’s risk level.

## 2.4 The Shift to Behavioral Data and Richer Datasets

Later in the literature, there was a shift away from relying mainly on demographic and static academic data towards the use of behavioral data, such as interaction logs from learning management systems.

Kloft et al. (2014) demonstrated that clickstream data, such as session frequency, page views, and activity patterns, can effectively predict student dropout. Importantly, this highlights the usefulness of behavioral data derived from student interactions. Similarly, Whitehill et al. (2017) found that a simple baseline model using only the number of days since a student's last interaction with the course achieved an average AUC of 87.33%, suggesting that recent engagement behavior is a strong indicator of dropout risk. However, Whitehill et al. also showed that models that use richer clickstream features achieved higher performance overall, suggesting that broader behavioral features should be combined for a more meaningful output.

A major development in this area is the introduction of the OULAD dataset by Kuzilek et al. (2017). This dataset includes over 32 000 students and more than 10 million interaction records, combining demographic, academic and behavioral data. Unlike earlier datasets, OULAD captures detailed student activity, which makes it an ideal dataset to build an early intervention system around.

Subsequent work using this dataset reinforces its value. Jha et al. (2019) applied various machine learning models to OULAD and found that behavioral features delivered the highest predictive performance, achieving AUC values approaching 0.9. Interestingly, adding demographic and static academic data provided only marginal improvements. This further adds to the conclusion that dynamic behavioral data is far more informative for predicting student risk than traditional static variables.

Overall, this shift from static data to behavioral data represents a key turning point, moving from static demographic data to dynamic behavioral data, which is more informative of student performance.

## 2.5 Temporal Modeling and the Importance of Early Prediction

A further milestone in this field is recognizing that student behavior is dynamic and changes over time, rather than remaining static throughout a course.

He et al. (2015) illustrates that student performance can be predicted relatively early, achieving an AUC of 0.87, as early as week 2 and improving to 0.94 by week five. This demonstrates that early prediction is feasible and provides a foundation for developing early-warning systems.

However, Kloft et al. (2014) noted that prediction accuracy tends to improve as the course progresses and more behavioural data becomes available. This creates a clear trade-off: while early predictions allow more time for intervention, they are generally less reliable due to limited data.

To better capture this temporal nature, Fei and Yeung (2015) proposed modelling student behavior as a sequence using LSTM (Long Short-Term Memory networks) which models student activity as sequences over time. Their results indicate that temporal models using dynamic data outperform traditional static models by capturing long-term behavioral patterns. Nevertheless, LSTM models come with notable drawbacks, they are computationally expensive and significantly more difficult to interpret, which can limit their effectiveness.

Márquez-Vera et al. (2016) offers a valuable perspective by arguing that the success of prediction models should be judged primarily on their usefulness for enabling timely interventions, rather than on accuracy metrics alone. Their work supports the idea that reliable predictions can be made within the first few weeks of the course

In general, the literature has evolved from purely static prediction models to more time-aware approaches. While early prediction is clearly valuable for intervention, the trade-off between timeliness and accuracy remains a challenge. This has increased interest in temporal modelling techniques that can better reflect the evolving nature of student behavior.

## 2.6 Evaluation Challenges, Class imbalance and Generalizability

Despite improvements in modelling techniques, there are several challenges that remain in evaluating student prediction models.

One of the most persistent issues is class imbalance. In most educational datasets, successful students greatly outnumber those who fail or drop out. This occurs because at-risk students often disengage early, leaving fewer interaction records and creating an imbalanced dataset (Albreiki et al., 2021). As a result, models can achieve misleadingly

high accuracy by simply predicting the majority class while performing poorly at identifying the students who need support.

He and Garcia (2009), rightly point out that accuracy can be a deceptive metric in such cases. They recommend using more suitable measures such as AUC, recall and F1-score. This is supported by several other studies as well including He et al. (2015) and Fei and Yeung (2015). Both of which used AUC as their primary evaluation metric.

Unfortunately, many studies still fail to address class imbalance, either by relying solely on accuracy or by not applying techniques such as resampling or class weighting. This is a significant methodological weakness, since the main purpose of these systems is to reliably detect at-risk students rather than maximize overall accuracy.

Another important concern is the reliability of the evaluation methods. Whitehill et al. (2017) showed that studies overestimate performance by training and testing on the same dataset, which produces overly optimistic results that would not hold up in real-world conditions.

Beyond evaluation methods, there is a broader issue of generalizability. A significant portion of existing studies are based on Massive Open Online Courses (MOOCs), where large-scale behavioral data is readily available and many of the influential studies (Kloft et al., 2014; He et al., 2015; Whitehill et al., 2017), are based on MOOC environments. These characteristics differ substantially from traditional higher education. MOOC environments typically feature self-pace learning, lower commitment levels, and very high dropout rates often exceeding 80% (Kloft et al., 2014; Whitehill et al., 2017). In contrast, university courses usually have a structured curriculum, formal assessments, and greater academic accountability.

Consequently, models developed using MOOC data may capture patterns that do not transfer well into conventional university settings. This raises serious questions about the real-world applicability of much of the existing research and highlights the need for more studies conducted in traditional higher education contexts.

## 2.7 Current State and Research Gap

While most of the studies focus on prediction, fewer address how these predictions can be used in practice.

One notable exception is the Course Signals system developed by Arnold and Pistilli (2012) at Purdue University. This system used predictive analytics to identify at risk-students and provided feedback using a traffic-light visualization. The project reported improvements in both student performance and retention. This clearly illustrates that the true value of machine learning in education lies not just in prediction accuracy, but in its ability to support decision making and timely intervention.

Nevertheless, most research stops at the modelling stage. Looking at the current state of the field, there has been clear progress – moving from simple models with static data to more advanced ensemble methods and richer behavioral datasets.

However, several gaps remain:

- Over reliance on MOOC datasets
- Limited generalizability to higher education settings
- Insufficient handling of class imbalance
- Lack of real-world system implementation

## 2.8 Project Positioning

This project aims to build on this existing research while addressing some of the key limitations in these studies.

It combines:

1. XGBoost, a high-performance ensemble method
2. Behavioral and academic data from the OULAD dataset
3. Proper evaluation techniques
4. A full-stack web application that connects prediction with practical intervention tools.

Unlike many previous studies that focus on prediction accuracy, this project places equal emphasis on usability. The system enables academic staff to search for students, view risk predictions with contributing factors, and record interventions. By integrating

machine learning into a functional web-based early intervention system, this work aims to bridge the gap between predictive analytics research and real-world application in higher education.

# 3. Aims, Objectives, Requirements

## 3.1 Global Aim of the Project

The main goal of this project is to design and implement a full-stack Early Intervention System designed for a higher education environment. The system applies machine learning to predict student academic risk and presents the results through a web application. It is designed to help staff identify students who may be struggling or disengaging early on and provide them with support.

Staff should be able to use the platform to see student engagement and performance metrics, generate risk predictions and get initial intervention recommendations. On top of this it also aims to provide tools to record and track interventions over time, enabling a more structured and hands on approach to helping students.

Additionally, the system provides administrative features for dataset management, model retraining and tunable risk thresholds that help support continuous improvement of prediction accuracy and adaptation to diverse institutional settings.

Overall the project aims to address the gap between machine learning models and real-world usability for embedding predictive analytics into a practical decision support system.

## 3.2 Objectives

- O1 – User account and access management

Implement secure and role based access to the system, allowing different types of users such as staff or administrators to interact with features intended for them while maintaining system security.

- O2 – Student data management

To provide access to student records, enabling users to view , search, filter, and manage student data in an efficient manner.

- O3 – Risk prediction and machine learning features

Integrate a machine learning model capable of generating risk predictions, along with interpretable outputs such as confidence scores and contributing factors.

- O4 – Dashboard and data visualisation

Develop dashboard and visualisation features that present key information clearly, including overall risk ditribution, engagement trends, and high risk student cases.

- O5 – Intervention support

Support academic staff to take action by providing suggested interventions based on the students risk level and enabling the tracking of those interventions.

- O6 – Ethical and transparency features

Ensuring responsible use of data and models by implementing transparency , explainability, and appropriate access controls.

- O7 – System administration

Allowing administrators to manage datasets, retrain models , and configure the system parameters to maintain and improve system performance.

## 3.3 Requirements

The requirements below were derived from the project objectives, they define what the system must do and how it should behave to meet the aim of this project. Each requirement is linked to an objective so that the system functionality can be traced back to the original goals.

O1 – User account and access management

- R1.1 Authentication: The system shall allow users to log in and log out securely.

- R1.2 Authorisation (Role Based Acess Control) : The system shall restrict access to features based on user role (Staff , Admin).

- R1.3 Password Management: The system shall allow authenticated users to change their password

- R1.4 Password Reset: The system shall provide a password reset mechanism suitable for this prototype (token based reset flow)

- R1.5 Session management – The system shall enforce session expiry and log users out after a period of inactivity.

O2 – Student Data Management

- R2.1 Student listing: The system shall display a list of students

- R2.2 Search and filtering: the system shall allow users to search and filter students by attributes such as dataset, region, presentation and risk level.

- R2.3 Student profile: The system shall provide a detailed profile view for each individual student

- R2.4 Historical records: The system shall allow users to view historical feature snapshots and historical predictions for a student

- R2.5 Administrative record management: The system shall allow authorised admin users to edit and delete student records

O3 – Risk prediction and Machine Learning Features

- R3.1 Prediction Generation: The system shall generate a risk prediction for an individual student using the latest available feature snapshot and the active model.

- R3.2 Risk Score: the system shall output a numerical risk score for each prediction

- R3.3 Risk level categorization: The system shall categorize predictions into risk levels, for example low/medium/high based on the tunable threshold.

- R3.4 Confidence: The system shall provide a confidence score associated with the prediction output.

- R3.5 Explainability: The system shall provide a basic explanation of contributing factors influencing the prediction

- R3.6 Timestamping: The system shall store the timestamp of each prediction.

- R3.7 Prediction Comparison: The system shall allow users to compare the latest prediction with the previous prediction for the same student.

- R3.8 Model performance visibility (admin): The system shall allow admin users to view stored model performance metrics for the active model

O4 – Dashboard and data visualisation

- R4.1 Summary overview: The system shall display an overview including total students and risk distribution counts.

- R4.2 Risk distribution visualisation: The system shall visualise the distribution of student risk levels.

- R4.3 High-risk visibility: The system shall provide a view of recent or current high-risk students to support priortisation.

- R4.4 Usable navigation: The system shall provide an intuitive navigation structure to access dashboard and student views efficiently.

O5 – Intervention support

- R5.1 Intervention Suggestions: The system shall generate rule-based intervention suggestions based on student risk level and available indicators.

- R5.2 Intervention logging: The system shall allow staff to create intervention records

- R5.3 Intervention tracking: The system shall allow staff to update intervention status and review historical interventions for a student.

- R5.4 Traceability: The system shall link interventions to a prediction record to support traceability of decisions.

O6 – Ethical and transparency features

- R6.1 Data constraints: The system shall be developed and tesed using publicly available anonymised, or synthetic datasets.

- R6.2 Access control for sensitive views: The system shall restrict access to student analytics to authenticated users

- R6.3 Transparency: The system shall provide a clear visibility of how risk levels are determined and provide basic interpretability information.

- R6.3 Human Oversight – The system shall be designed as a decision-support tool where staff can apply judgement rather than acting on predictions alone.

O7 – System administration

- R7.1 Dataset Management: The system shall allow admin users to upload and manage datasets used for training and experimentation.

- R7.2 Retraining: The system shall allow admin users to retrain the model using an uploaded dataset and activate the new model.

- R7.3 Risk threshold configuration: The system shall allow admin users to configure risk thresholds used for risk categorization

- R7.4 Data import: The system shall allow admin users to import prepared datasets into the system database for application use

## 3.4 Non-Functional requirements

- NFR1 Security: Passwords shall be stored using strong hashing. Protected endpoints shall require authentication, admin operations shall require authorisation.

- NFR2 Usability: The user interface will present key information with minimal steps and provide clear feedback on actions.

- NFR3 Maintainability: The system shall be structured into separate layers to support iterative development, testing and future updates.

- NFR4 Reliability: The system shall handle missing data gracefully and provide a error message where required.

- NFR5 Reproducibility (ML) : Training outputs (model artefacts and metrics) shall be persisted so model performance can be reviewed and models can be reloaded for inference.

## 3.5 Development Methodology

This project followed an iterative development approach. As the system was not a simple stand-alone application and comprised of many parts including machine learning, backend development, database design etc., this was the right approach. Building the whole system in one attempt would have made the project difficult to manage so the work was divided into smaller stages.

The prediction model was the heart of the early-intervention system and therefore it formed the starting point for the project. It was necessary to understand the dataset, prepare the feature, train the model and confirm that the prediction process was appropriate to identify student risk before building the full application. This helped to ensure the system was built around a working predictive component rather than designing the application before confirming the machine learning component was feasible.

The second major step was developing the backend. It was built incrementally, starting with the database schema using PostgreSQL and SQLAlchemy , then gradually built the FASTAPI endpoints. This approach made it much easier to test dataflows and ensure each part of the system worked correctly before adding more complex features such as authentication, intervention tracking and administrative functions.

Once the core API routes were stable, I began developing the frontend in parallel with the backend. This allowed me to connect the React interface directly to real backend functions instead of working with a static prototype. Areas like the dashboard, student profile, prediction views and intervention pages were added itteratively, Afterwards the more coplex admin features were refined as the system grew more complete.

One of the clear advantages of the iterative method was that new improvements for the system only became clear after the parts of the system were already working. For example, after building the prediction generation feature, it became obvious there needs to be a baseline comparison with other students to enable comparison and make the results clear and interpretable. Testin was performed continously throughout the development using both manual and automated methods.

This methodology proved well-suited to the projects scope as a research prototype combining multiple technical domains.

# 4. System Design

## Why the design stage matters?

The design stage was important because the system combines multiple components, including the frontend, backend, database layer and the ML component. This section explains how everything was organised to support scalability, security and usability.

## 4.1 System architecture (high-level)

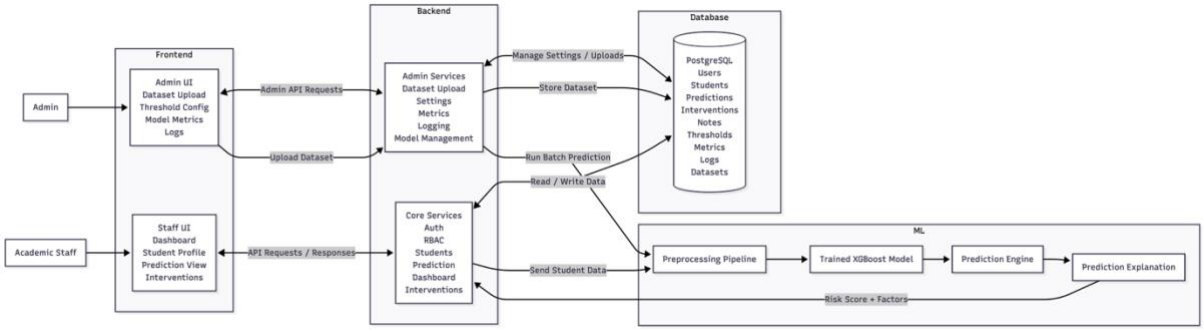



*Fig 4. 1*

The system uses a classic three-tier web architecture (presentation, application, and data layers) with the added ml component to separate concerns and improve maintainability as we can see in the figure 4.1.

### Presentation layer (Frontend)

The frontend was built using React with TypeScript. React was selected for its component-based structure and efficient stage management capabilities. For the dashboard in particular this is crucial as it helps risk scores, student lists, and other data to update dynamically without needing to reload the page. For academic staff who need to explore student data efficiently react provides a more responsive and interactive user experience.

### Application layer (Backend)

The backend was implemented using FastAPI, a modern Python web framework. FastAPI was chosen because of its strong performance, built-in data validation with Pydantic, and support for asynchronous operations.

This layer is reponsible for:

- Authentication and role-based access control
- Managing student data and historical records
- Generating machine learning predictions
- Mapping risk scores to categories
- Storing interventions and tracking actions

- Supporting administrative operations such as dataset management and model retraining

Because of its lightweight nature and better suitability for building RESTful APIs that integrate with machine learning components, FastAPI was preferred over heavier frameworks like Django.

### Data layer

PostgreSQL was selected as the relational database management system. A relational database was chosen over NoSQL solutions because the system requires strong data consistency with well-defined relationships between entites (students, predictions, interventions etc), and reliable transaction support

In addition to storing application data, trained machine learning models and associated artefacts are saved on disk and referenced through the database. This approach supports model versioning and allows different models to be activated without modifying any backend code.

## 4.2 System Data Flow

The interaction between the system components follows a structured flow:

1. The user interacts with the frontend (requests a student profile or prediction).
2. The request is sent to the FastAPI backend via HTTP/
3. The backend retrieves the relevant information from the PostgreSQL database.
4. If a prediction is required the backend loads the active machine learning model.
5. The model generates a risk score based on the latest feature snapshot.
6. The backend maps the score to a risk category (low, medium, high).\
7. The result is stored in the database and returned to the frontend.
8. The frontend displays the result through the dashboard.

This flow seperates data processing , business logic and presentation.

## 4.3 Use Case Design

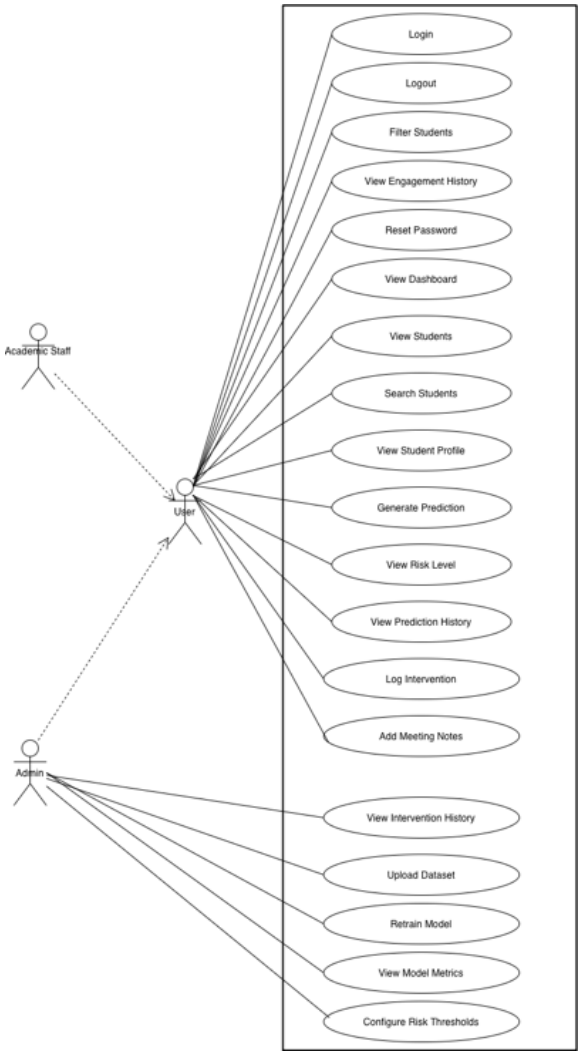



*Fig 4. 2*

As shown in Figure 4.2, Staff Users form the primary users of the system and can perform the following key actions:

- Log in and manage their account

- View the dashboard and high risk students

- Search and filter students

- View detailed student profiles and engagement history

- Generate risk predictions and view contributing factors

- Compare current and historical predictions

- Log and track interventions

Admins have all the privileges of staff users plus additional system management capabilities:

- Uploading and managing datasets
- Retraining and activating new machine learning models
- Tuning risk thresholds
- managing model versions

The clear seperation of roles restricts sensitive operations to authorised admins only and helps ensure the system security and maintain appropriate boundaries, addressing requirement R1.2 (Role-Based Access Control)

The use case design allows for a solid foundation for implementing a secure user experience and aligns well to meet objectives (O3 , O5 and O7).

## 4.4 Database Design (ER diagram)

A relational database schema was designed to support the full lifecycle of student monitoring, prediction and intervention. The complete strucrure is illustrated in Figure 4.3 below.

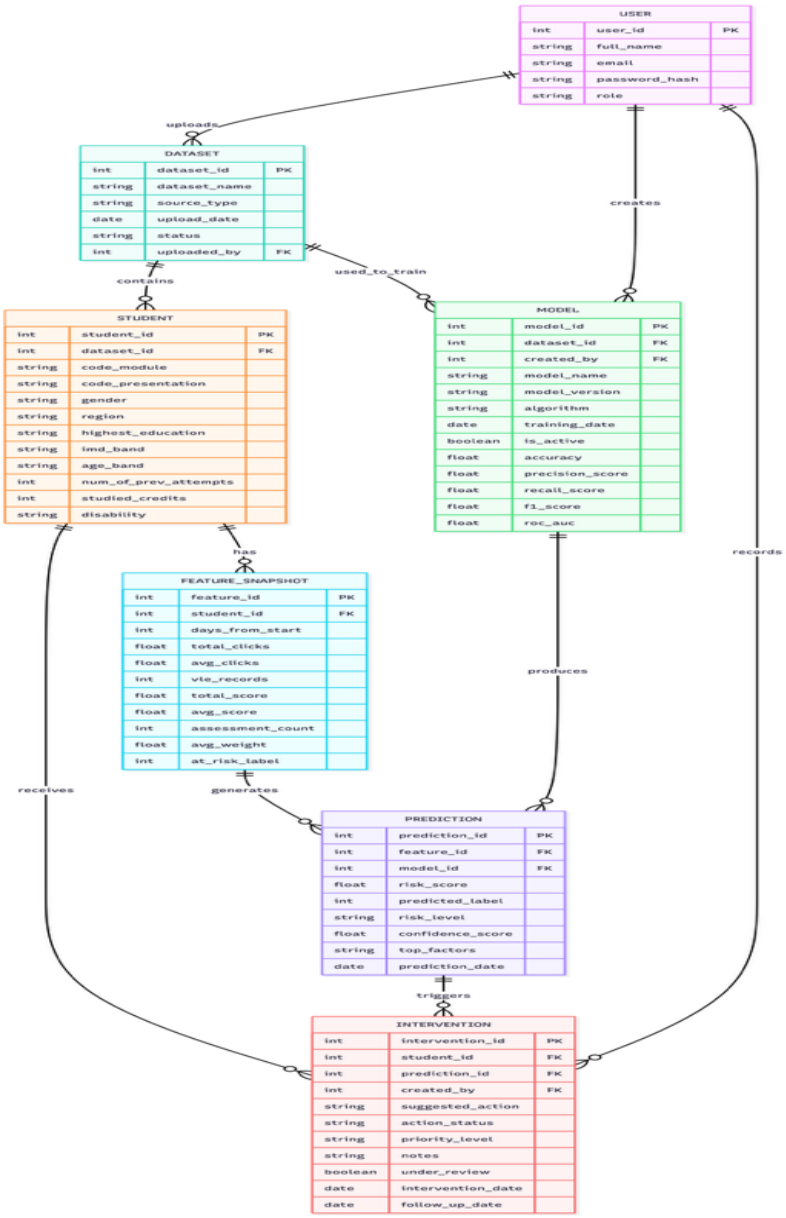



*Fig 4. 3*

As shown in Figure 4.3, the database consists of two main groups of entites

Core entities

- Users: stores authentication and role information to support secure access control.
- Students: Stores student identifiers and attributes, linked to datasets
- Feature snapshots: Stores engineered features at specific time points, which enables temporal tracking of engagement.
- Predictions: Stores model output, allowing comparison of risk levels over time.
- Interventions: Stores actions taken by staff , supporting traceability and evaluation of decisions.

Administrative entities

- Datasets: represents uploaded datasets used for training or analysis
- Model records: Stores metadata about trained models, including versioning and active status
- Risk thresholds: defines boundaries for categorizing risk scores into levels.

This schema estabilished clear relationships between entities, particularly linking predictions to both feature snapshots and interventions. Such connections are essential for maintaining traceability, such as showing how a students risk level changed over time and what interventions were applied in response.

## 4.5 Design Decisions and Trade-Offs

Several design decisions were made to balance performance, interpretability and system complexity:

- XGBoost vs Deep learning

While Deep learning models can capture temporal patterns more effectively, they were not selected because of high computational costs and limited interpretability. XGBoost was chosen as the primary model because it provides a strong predictive performance while allowing us to see the feature importance scores.

- Relational Database vs NoSQL

A relational database was selected due to the need for structured relationships and consistency. Although NoSQL systems offer flexibility, they are less suited for enforcing relationships between entities like predictions and interventions.

- FastAPI vs Django

FastAPI was selected because of its lightweight design and performance advantages. While Django is feature-rich, it introduces overhead for an API focussed system.

- Prototype Scope vs Full Deployment

The system is designed as a prototype rather than a production system. It focusses on core functionality while acknowledging limitations in scalability and real world deployment.

## 4.6 API Design

The backend was designed as a RESTful API using FastAPI. This allows the frontend to communicate with the backend through HTTP requests, with responses returned in JSON format. Seperating the frontend and backend improves the maintainability and makes the system easier to develop and test.

To keep the codebase clean and avoid different paerts of the logic from becoming tangled, the API was organised by functional area. Separate groups were created for authentication, student management, predictions, interventions, dashboards, and admin operations.

Authentication is handled using JSON Web Tokens (JWT). After a sucessful login, the backend issues a token which the frontend then sends with every protected request. The backend validates the token and enforces role-based access control. Staff users have access to monitoring and intervention features, while administrative functions such as dataset management, model retraining, and threshold configuration are restricted to admins only. These checks are enforced server-side for proper security.

One of the most important endpoints is the prediction route. When called, the backend fetches the student's latest feature snapshot, passes it to the active XGBoost model, stores the prediction result (including the risk score, confidence and contributing factors), and returns it to the frontend.

The intervention API supports the action stage of the system. After a prediction has been generated, staff can view suuggested interventions and record the action taken. Allowing the system support a more practical workflow rather than being a simple risk displayer.

Furthermore, the API was also designed to use appropriate HTTP status codes to make errors easier to handle. This includes authentication errors , not-found responses etc. This helps the frontend give clearer feedback to users making enhancing reliability.


Users
GET /users/ Get Users
POST /users/ Create User
Students
GET /students/ Get Students
POST /students/ Create Student
POST /students/create-with-features Create Student With Features
GET /students/{student_id}/feature-averages Get Feature Averages For Student
GET /students/search Search Students
GET /students/{student_id} Get Student
PUT /students/{student_id} Update Student
DELETE /students/{student_id} Delete Student
GET /students/{student_id}/profile Get Student Profile


Feature Snapshots
GET /feature-snapshots/ Get Feature Snapshots
POST /feature-snapshots/ Create Feature Snapshot
GET /feature-snapshots/{feature_id} Get Feature Snapshot
GET /feature-snapshots/student/{student_id} Get Feature Snapshots For Student
Predictions
POST /predictions/generate/{student_id} Create Prediction
POST /predictions/generate-dataset/{dataset_id} Generate Predictions For Dataset
GET /predictions/ Get Predictions
GET /predictions/{prediction_id} Get Prediction
GET /predictions/student/{student_id} Get Predictions For Student
GET /predictions/compare/{student_id} Compare Latest Predictions

## Interventions

- GET `/interventions/suggestions/{student_id}` Get Intervention Suggestions
- GET `/interventions/` Get Interventions
- POST `/interventions/` Create Intervention
- GET `/interventions/student/{student_id}` Get Interventions For Student
- GET `/interventions/{intervention_id}` Get Intervention
- PUT `/interventions/{intervention_id}` Update Intervention

## Admin - Risk Thresholds

- GET `/admin/risk-thresholds/` Get Thresholds
- PUT `/admin/risk-thresholds/` Update Thresholds

## Authentication

- POST `/auth/login` Login
- POST `/auth/logout` Logout
- GET `/auth/me` Read Me
- POST `/auth/change-password` Change Password
- POST `/auth/request-password-reset` Request Password Reset
- POST `/auth/reset-password` Reset Password

## Admin - Models

- GET `/admin/models/` List Models
- GET `/admin/models/active` Get Active Model
- PUT `/admin/models/activate/{model_id}` Activate Model
- DELETE `/admin/models/{model_id}` Delete Model
- GET `/admin/models/{model_id}/metrics-file` Get Model Metrics File

## Admin - ML

- POST `/admin/ml/upload-dataset` Upload Dataset
- POST `/admin/ml/retrain` Retrain Model
- POST `/admin/ml/retrain-oulad` Retrain Model Oulad

## Dashboard

- GET `/dashboard/summary` Get Dashboard Summary
- GET `/dashboard/risk-distribution` Get Risk Distribution
- GET `/dashboard/recent-high-risk` Get Recent High Risk
- GET `/dashboard/intervention-status` Get Intervention Status

## Admin - Data

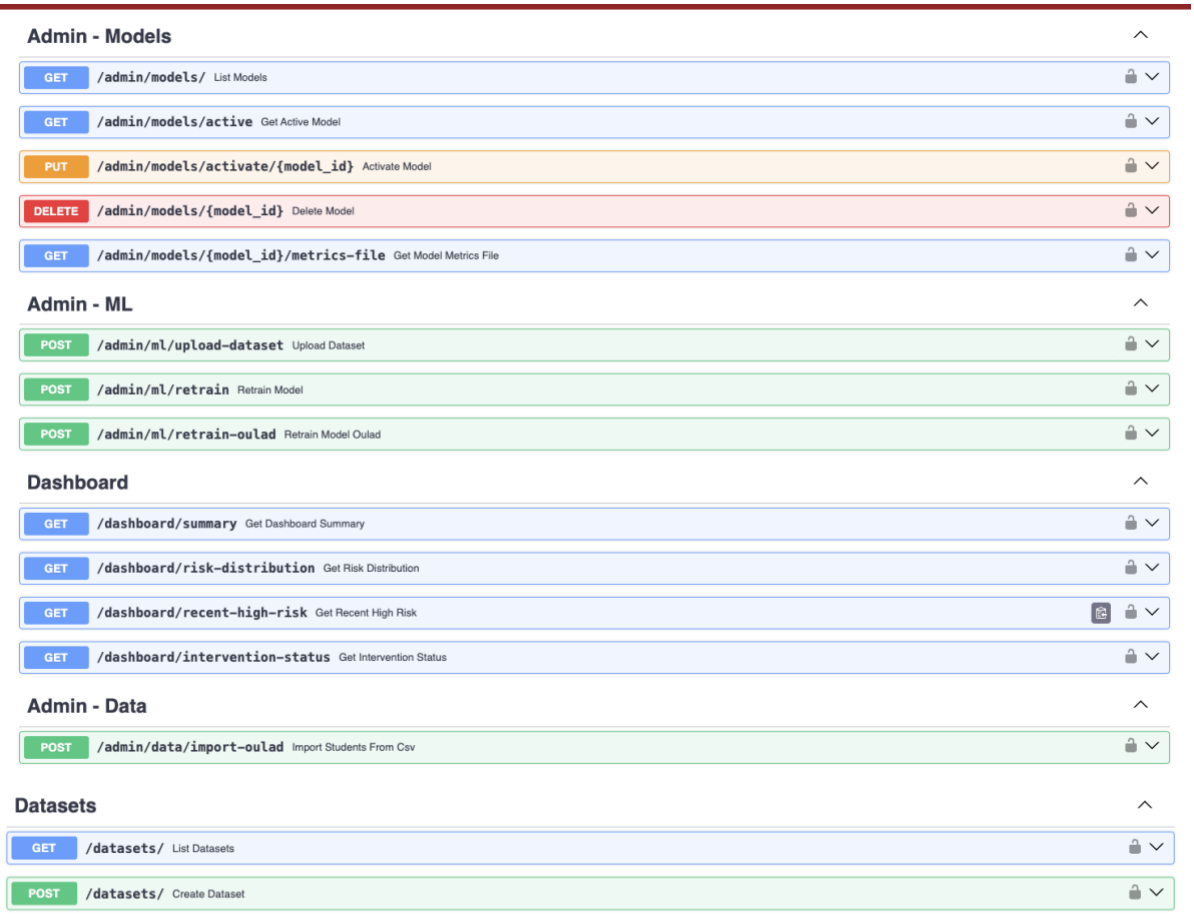



*Figure 4.4*

## 4.7 Security Design

Security was considered from the early planning stages of the project. Although the system uses publicly available and anonymised data, it was designed with core security principles, as it deals with sensitive information.

All users must first log into the application to make sure that only authorized staff can view protected resources and access system features. Satisfying requirements R1.1 AND R6.2 in the aims and requirements section.

Role-based access control was implemented to differentiate between staff and administrator privileges. Staff users are limited to core functions (see figure 4.2). Admins hace additional privileges. These restrictions are enforced on the backend and not just hidden in the frontend, to provide proper security.

Access from an unattended device was another key concern, to reduce this risk, both backend token expiry and frontend timeouts were implemented. This will automatically log the user out after a period of inactivity.

Finally, all passwords are stored using strong hashing algorithms. This ensures that even in the event of a database breach, user credentials will remain protected.

## 4.8 Scalability and Maintainability

The system was designed with future development and potential real-world use in mind. Particular attention to scalability and maintainability was given through several choices.

1. The three-tier architecture allows the frontend, backend, and machine learning components to be updated independantly without affecting the rest of the system.

2. Including model retraining allows administrators to upload new datasets and train updated models as more data becomes available. This will allow for continous improvement in prediction accuracy over time.

3. Finally, the dataset management features allow the system to adapt to different educational context by importing new or institution-specific data.

Together these design decisions, ensure that the current prototype provides a solid foundation that can evolve beyond the initial scope.

# 5. Implementation

## 5.1 Overview

Implementation followed the iterative methodology described in Section 3.5. The different components were developed and integrated in stages, with regular testing at each step to ensure they worked correctly and connected smoothly with one another.

## 5.2 Tech Stack

The system was built using a full-stack approach with the following technologies:

- Frontend: React with TypeScript and Vite, providing a responsive and component-based user interface.
- Backend: FastAPI paired with Pydantic for building the Rest API and performing request/response validation
- Database – PostgreSQL with SQLAlchemy ORM to manage relational data and maintain data integrity.
- Machine learning – XGBoost and scikit-learn, with trained models and associated artefacts saved to disk for efficient run time predictions.

## 5.3 Backend Implementation

### Project structure

The backend was structured as described in section 4.6 (API Design), with code organised by responsibility into distinct layers:

- Database models
- API routes
- Business logic services
- Dependancies
- Schemas
- machine learning utilities

Database models were created for all core entities. API routes were grouped together by domain like authentication, admin functions etc. with each group placed in its own file for clarity. Business logic was seperated from route handlers where appropriate, and machine elarning utilities were kept in dedicated modules (fig 5.3.1).

this organisation reduced duplication between files and made the backend easier to upgrade.

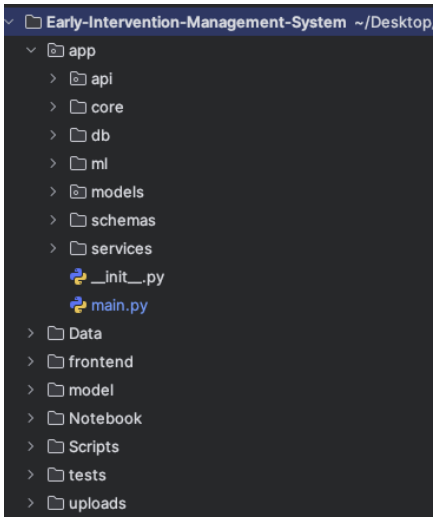



*fig 5.3. 1*

## Authentication and Role-Based Access Control

Authentication was implemented using JSWON Web Tokens in the FastAPI backend. The login endpoint (POST /auth/login) validates the user credentials and returns an access token with the users identity and expiration time (fig 5.3.2). Protected endpoints decode and validate the tokens before allowing access to data (fig 5.3.3).

```
@router.post( path: "/login", response_model=Token)  Akash Gunathilaka
def login(
    form_data: OAuth2PasswordRequestForm = Depends(),
    db: Session = Depends(get_db),
):
    user = db.query(User).filter(User.email == form_data.username).first()

    if not user or not verify_password(form_data.password, user.password_hash):
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Incorrect email or password",
            headers={"WWW-Authenticate": "Bearer"},
        )

    if not user.is_active:
        raise HTTPException(status_code=403, detail="Inactive user")

    access_token = create_access_token(data={"sub": user.email})

    return {
        "access_token": access_token,
        "token_type": "bearer",
    }
```

*fig 5.3. 2*

```python
def get_current_user(  1 usage  Akash Gunathilaka *
    token: str = Depends(oauth2_scheme),
    db: Session = Depends(get_db),
):
    try:
        # This decode step enforces expiry:
        # if `exp` is in the past or signature is invalid, we raise an invalid token error
        payload = jwt.decode(token, SECRET_KEY, algorithms=[ALGORITHM])
        email = payload.get("sub")
        if email is None:
            raise credentials_exception
    except InvalidTokenError:
        raise credentials_exception

    # Token only contains the user identifier we still load the user from the database
    # so we can check current role and avoid trusting stale token data.
    user = db.query(User).filter(User.email == email).first()
    if user is None:
        raise credentials_exception

    return user
```

*fig 5.3. 3*

Although the frontend hides admin pages for better user experience, the actual access control is handled in the backend. Role-Based Acess Control was enforced on the server side using a dependency (require_admin, fig 5.3.4). This ensures that admin-only routes are strictly restricted.

```python
def require_admin(  26 usages  Akash Gunathilaka *
    current_user: User = Depends(get_current_active_user),
):
    # Server-side RBAC: this is the real enforcement (UI hiding alone is not security).
    if current_user.role.lower() != "admin":
        raise HTTPException(status_code=403, detail="Admin access required")
    return current_user
```

*fig 5.3. 4*

Password security was handled using pwdlib for strong hashing, so passwords are never stored in plaintext in the database (fig 5.3.5). A simple prototype password reset flow was also implemented using time-limited reset tokens, allowing the users to reset their passwords, without requiring email or sms (fig 5.3.6).

```python
password_hash = PasswordHash.recommended()


def verify_password(plain_password: str, hashed_password: str) -> bool:  3 usages  Akash Gunathilaka
    return password_hash.verify(plain_password, hashed_password)


def get_password_hash(password: str) -> str:  5 usages  Akash Gunathilaka
    return password_hash.hash(password)
```

*fig 5.3. 5*

```python
def create_password_reset_token(email: str, expires_minutes: int = 15) -> str:  2 usages  Akash Guna
    """
    Token-based password reset
    """
    expire = datetime.now(timezone.utc) + timedelta(minutes=expires_minutes)
    payload = {"sub": email, "purpose": "password_reset", "exp": expire}
    return jwt.encode(payload, SECRET_KEY, algorithm=ALGORITHM)


def decode_password_reset_token(token: str) -> str:  2 usages  Akash Gunathilaka
    payload = jwt.decode(token, SECRET_KEY, algorithms=[ALGORITHM])
    if payload.get("purpose") != "password_reset":
        raise ValueError("Invalid token purpose")
    email = payload.get("sub")
    if not email:
        raise ValueError("Missing subject")
    return str(email)
```

*fig 5.3. 6*

Session management uses a two-layer approach.

Backend access tokens expire after a configurable time period (fig 5.3.7):

```python
# Frontend also has an idle timeout, but the backend enforces expiry
ACCESS_TOKEN_EXPIRE_MINUTES = int(os.getenv("ACCESS_TOKEN_EXPIRE_MINUTES", "60"))
```

```python
def create_access_token(data: dict, expires_delta: timedelta | None = None) -> str:  2 usages
    # `data` is the payload. We always add an expiry (`exp`) so tokens are time-limited.
    to_encode = data.copy()
    expire = datetime.now(timezone.utc) + (
        expires_delta or timedelta(minutes=ACCESS_TOKEN_EXPIRE_MINUTES)
    )
    to_encode.update({"exp": expire})
    return jwt.encode(to_encode, SECRET_KEY, algorithm=ALGORITHM)
```

*fig 5.3. 7*

and the frontend implements an inactivity timeout that automatically logs the user out after a set period of no activity (fig 5.3.8):

```javascript
useEffect(() : (() => void) | undefined  => {
  // Idle timeout (front-end safety net).
  // Backend tokens already expire, but this helps if someone leaves the app open
  // on a shared/unattended machine: after no interaction we clear the token and redirect back to login.
  const token : string | null  = getToken()
  if (!token) return

  const timeoutMinutes : number  = Number(import.meta.env.VITE_IDLE_TIMEOUT_MINUTES ?? 30)
  const timeoutMs : number  = Math.max( ...values: 1, timeoutMinutes) * 60_000

  const markActive : () => void  = () : void  => {  Show usages  Akash Gunathilaka *
    // Avoid spamming localStorage on high-frequency events like mousemove.
    // We only persist activity at most once every ~5 seconds.
    const now : number  = Date.now()
    const last : number  = Number(localStorage.getItem(LAST_ACTIVE_KEY) ?? '0')
    if (!last || now - last > 5000) localStorage.setItem(LAST_ACTIVE_KEY, String(now))
  }

  // Treat initial page load as activity so users don't get logged out immediately
  // if they refresh after a long pause.
  markActive()

  // Any user interaction counts as "activity".
  const events: Array<keyof WindowEventMap> = ['mousemove', 'mousedown', 'keydown', 'scroll', 'touchstart']
  for (const ev of events) window.addEventListener(ev, markActive, { passive: true })

  const interval : number  = window.setInterval(() : void  => {
    const now : number  = Date.now()
    const last : number  = Number(localStorage.getItem(LAST_ACTIVE_KEY) ?? '0')
    const idleFor : number  = now - (last || 0)
    if (idleFor >= timeoutMs) {
      logout()
```

*fig 5.3. 8*

## Student Management

Student management functionality was implemented through a dedicated router and its database models.

The main listing and search features are provided by the GET /students and GET /students/search endpoints. The search endpoint supports server side filtering by attributes (fig 5.3.9). These filters are exposed through backend query parameters so the frontend can request the records.

```
/students/search
@router.get( path: "/search", response_model=list[StudentSearchResult])  Akash Gunathilaka
def search_students(
    dataset_id: int | None = Query(default=None, description="Filter by dataset_id"),
    risk_level: RiskLevel | None = Query(default=None, description="Filter by latest prediction risk level"),
    code_presentation: str | None = Query(default=None, description="Filter by code_presentation"),
    region: str | None = Query(default=None, description="Filter by region"),
    limit: int = Query(default=50, ge=1, le=500, description="Max students returned (before risk filter)"),
    db: Session = Depends(get_db),
    current_user: User = Depends(get_current_active_user),
):
    q = db.query(Student)

    if dataset_id is not None:
        q = q.filter(Student.dataset_id == dataset_id)
    if code_presentation is not None:
        q = q.filter(Student.code_presentation == code_presentation)
    if region is not None:
        q = q.filter(Student.region == region)
```

*fig 5.3. 9*

To improve performance and simplify the frontend logic, a combined student profile endpoint was implemented. Instead of making multiple separate requests, this single endpoint returns the students basic information, the latest feature snaphot, current prediction, and intervention history in one response (fig 5.3.10). This reduces the number of separate API calls required by the frontend and keeps the profile data consistent.

```
@router.get( path: "/{student_id}/profile", response_model=StudentProfileResponse)  Akash Gunathilaka
def get_student_profile(
    student_id: int,
    db: Session = Depends(get_db),
    current_user: User = Depends(get_current_active_user),
):
    student = db.query(Student).filter(Student.student_id == student_id).first()
    if not student:
        raise HTTPException(status_code=404, detail="Student not found")

    latest_snapshot = (
        db.query(FeatureSnapshot)
        .filter(FeatureSnapshot.student_id == student_id)
        .order_by(FeatureSnapshot.feature_id.desc())
        .first()
    )

    latest_prediction = (
        db.query(Prediction)
        .filter(Prediction.student_id == student_id)
        .order_by(Prediction.prediction_id.desc())
        .first()
    )

    interventions = (
        db.query(Intervention)
        .filter(Intervention.student_id == student_id)
        .order_by(Intervention.intervention_id.desc())
        .all()
    )

    return {
        "student": student,
        "latest_feature_snapshot": latest_snapshot,
        "latest_prediction": latest_prediction,
        "interventions": interventions,
```

*fig 5.3. 10*

Administrative operations such as updating and deleting student records are available through PUT/students/{student_id} and DELETE /students/{student_id}. These endpoints are restricted to admin users via require_admin (fig 5.3.11).

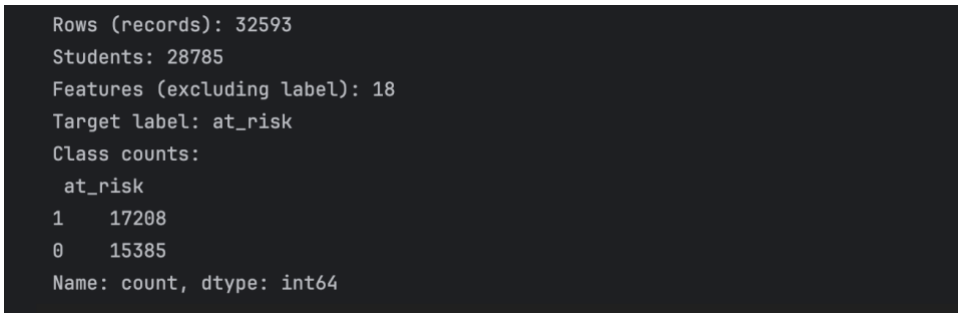
```python
@router.put( path: "/{student_id}", response_model=StudentResponse)  Akash Gunathilaka
def update_student(
    student_id: int,
    payload: StudentUpdate,
    db: Session = Depends(get_db),
    current_user: User = Depends(require_admin),
):
    student = db.query(Student).filter(Student.student_id == student_id).first()
    if not student:
        raise HTTPException(status_code=404, detail="Student not found")

    data = payload.model_dump(exclude_unset=True)
    for k, v in data.items():
        setattr(student, k, v)
    db.commit()
    db.refresh(student)
    return student


</students/{student_id}
@router.delete("/{student_id}")  Akash Gunathilaka
def delete_student(
    student_id: int,
    db: Session = Depends(get_db),
    current_user: User = Depends(require_admin),
):
    student = db.query(Student).filter(Student.student_id == student_id).first()
    if not student:
        raise HTTPException(status_code=404, detail="Student not found")

    db.delete(student)
    db.commit()
    return {"message": "Student deleted"}
```

*fig 5.3. 11*

Overall, the student management endpoints provide the data access layer needed for listing, filtering, profile and admin record management, while keeping the proper access control boundaries

## Machine learning Preprocessing and Training Pipeline

The public and anonymized OULAD dataset was used to train the model. The prediction task was framed as a binary clasification problem, students were labelled as either at risk or not at risk based on the information available on their outcome (fig 5.3.12).



```
Rows (records): 32593
Students: 28785
Features (excluding label): 18
Target label: at_risk
Class counts:
 at_risk
1    17208
0    15385
Name: count, dtype: int64
```

*Fig 5.3.12 OULAD dataset summary*

Before training, the dataset was preprocessed so that the model could use only the relevant engagement and academic indicators. This started by preparing behavioural

activity features, assessment-related features, and removing nulls and encoding categorical features, subsequently the final_result column was dropped to prevent leakage (Figs 5.3.13-5.3.15) and the dataset was then split into training and testing sets so that the model could be evaluated on records it had not seen during training (Fig 5.3.16).

```
assessment_features = (
    early_assessments
    .groupby(['id_student', 'code_module', 'code_presentation'])
    .agg(
        avg_score=('score', 'mean'),
        total_score=('score', 'sum'),
        assessment_count=('score', 'count'),
        avg_weight=('weight', 'mean'),
    )
    .reset_index()
)
```

*Fig 5.3.13*

```
#handling nulls
print('Before Handling' ,df.isnull().sum())

#fill nulls with 0 where it makes sense
feature_cols = [
    'total_clicks',
    'avg_clicks',
    'vle_records',
    'avg_score',
    'total_score',
    'assessment_count',
    'avg_weight',
]

#fill all numerical columns with 0
df[feature_cols] = df[feature_cols].fillna(0)
#fill imd_band which is a categorical column with 'Unknown'
df['imd_band'] = df['imd_band'].fillna('Unknown')

print('\nAfter Handling',df.isnull().sum())
```

*Fig 5.3.14*

```
#dropping final column to ensure no data leakage, student id is kept in place for grouped splitting
df = df.drop(columns=['final_result'])


#So i am creating two datasets for comparison between two models one with the module code and one without the module code
df_with_module = df.copy()
df_without_module = df.drop(columns=['code_module'])
```

*Fig 5.3.15*

```python
# turned the training and evaluation into one reusable function so i can compare models with and without certain features
def train_and_evaluate(df_input, model, model_name):
    x = df_input.drop(columns=['at_risk'])
    y = df_input['at_risk']

    #Using student id for grouped splitting
    groups = df_input['id_student']

    gss = GroupShuffleSplit(n_splits=1, test_size=0.2, random_state=42)
    train_idx, test_idx = next(gss.split(x, y, groups=groups))

    x_train = x.iloc[train_idx].copy()
    x_test = x.iloc[test_idx].copy()
    y_train = y.iloc[train_idx].copy()
    y_test = y.iloc[test_idx].copy()

    #dropping student id after splitting so it is not used as a feature
    x_train = x_train.drop(columns=['id_student'])
    x_test = x_test.drop(columns=['id_student'])

    #encoding after splitting
    x_train = pd.get_dummies(x_train, drop_first=True)
    x_test = pd.get_dummies(x_test, drop_first=True)

    #align columns
    x_train, x_test = x_train.align(x_test, join='left', axis=1, fill_value=0)

    #clean column names for XGBoost compatability
    x_train.columns = x_train.columns.str.replace(r'[\[\]<]', '', regex=True)
    x_test.columns = x_test.columns.str.replace(r'[\[\]<]', '', regex=True)

    #handle class imbalance (only applies to XGBoost)
    if isinstance(model, XGBClassifier):
        pos = int((y_train == 1).sum())
        neg = int((y_train == 0).sum())
        if pos > 0:
            model.set_params(scale_pos_weight=neg / pos)
```

*Fig 5.3.16*

Additionally class distribution was checked before training. The dataset was relatively balanced with 17,208 at risk records and 15,385 records labelled as not at risk (Fig 5.3.17), so accuracy was more meaningful than it would be in a highly imbalanced dataset. However, additional metrics such as precision, recall, F1-score and ROC-AUC were also recorded (Fig 5.3.18).

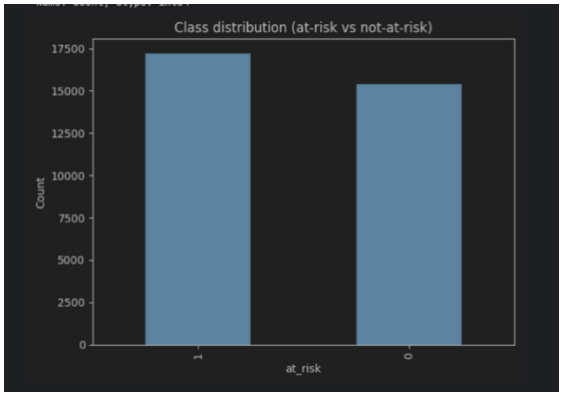



*Fig 5.3.17*

```
#metrics
print(f"\n{model_name}")
print("Accuracy:", accuracy_score(y_test, y_pred))
print("ROC-AUC:", roc_auc_score(y_test, y_prob))
print("PR-AUC:", average_precision_score(y_test, y_prob))
print("Precision:", precision_score(y_test, y_pred))
print("Recall:", recall_score(y_test, y_pred))
print("F1:", f1_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

*Fig 5.3.18*

The final trained model, feature column structure and performance metrics were saved as artefacts. Allowing the backend to load the same model and feature structure when predicting live this helped lower the risk of mismatched training and inference inputs while improving reproducibility (Fig 5.3.19).

```
(out_dir / 'final_metrics.json').write_text(json.dumps(metrics, indent=2), encoding='utf-8')
(out_dir / 'final_master_metrics.json').write_text(json.dumps(metrics, indent=2), encoding='utf-8')

print('Saved:', [
    str(out_dir / p)
    for p in [
        'final_xgb_model.pkl',
        'final_feature_columns.pkl',
        'raw_input_columns.pkl',
        'final_metrics.json',
        'final_master_model.pkl',
        'final_master_feature_columns.pkl',
        'final_master_raw_input_columns.pkl',
        'final_master_metrics.json',
    ]
])
```

*Fig 5.3.19*

## Feature selection and final model choice

A significant concern here was that, feature importance analysis showed code_module as one of the most important features. This conveyed that the model could be learning module difficulty rather than student engagement and assessment behavior. So to experiment, two models were trained with and without the code modules (Fig 5.3.20).

The final implemented model removed code_module to make sure the predictions were based more directly on student engagement and assessment rather than module difficulty.

```
train_and_evaluate(df_with_module, lr_model, "Logistic Regression WITH module")
train_and_evaluate(df_with_module, xgb_model, "XGBoost WITH module")

train_and_evaluate(df_without_module, lr_model, "Logistic Regression WITHOUT module")
train_and_evaluate(df_without_module, xgb_model, "XGBoost WITHOUT module")
```

*Fig 5.3.20*

## Prediction Pipeline and Runtime Inference

The prediction pipeline was implemented to enable risk predictions to be generated directly from within the web application. When a request is made to the endpoint , the backend first retrieves the students most recent feature snapshot from the database containing the engineered engagement and assessment features used as input for the model (fig 5.3.21).

```
def predict_for_student(student_id: int, db: Session, *, explain: bool = True) -> Prediction:
    feature_snapshot = (
        db.query(FeatureSnapshot)
        .filter(FeatureSnapshot.student_id == student_id)
        .order_by(FeatureSnapshot.feature_id.desc())
        .first()
    )

    if feature_snapshot is None:
        raise HTTPException(
            status_code=404,
            detail="No feature snapshot found for this student",
        )
```

*fig 5.3.21*

The backend then identifies the currently active model by querying the ModelRecord table for the version marked active. This model and associated feature column definition are loaded from the disk (fig 5.3.22). To ensure compatability, a single-row Pandas DataFrame is constructed with the exact feature structure used during training. This prevents errors due to mismatched features between training and inference time (fig 5.3.23).

```
active_model = (
    db.query(ModelRecord)
    .filter(ModelRecord.is_active == True)
    .first()
)

if active_model is None:
    raise HTTPException(status_code=404, detail="No active model found")

if not active_model.model_path or not active_model.feature_columns_path:
    raise HTTPException(
        status_code=500,
    detail="Active model artifact paths are missing",
    )

prediction_result = generate_prediction(
    feature_snapshot,
    model_path=active_model.model_path,
    feature_columns_path=active_model.feature_columns_path,
)
```

*fig 5.3.22*

```python
def build_feature_dataframe(feature_snapshot, feature_columns) -> pd.DataFrame:
    data = {
        "total_clicks": feature_snapshot.total_clicks,
        "avg_clicks": feature_snapshot.avg_clicks,
        "vle_records": feature_snapshot.vle_records,
        "avg_score": feature_snapshot.avg_score,
        "total_score": feature_snapshot.total_score,
        "assessment_count": feature_snapshot.assessment_count,
        "avg_weight": feature_snapshot.avg_weight,
    }

    df = pd.DataFrame([data])

    for col in feature_columns:
        if col not in df.columns:
            df[col] = 0

    return df[feature_columns]
```

*fig 5.3.23*

Afterwards, the XGBoost model returns a probability score for the at-risk class, which is saved as the student's risk score (fig 5.3.24). The score is then categorized into a risk level using tunable thresholds stored in the RiskThreshold table (fig 5.3.25).

```python
def generate_prediction(feature_snapshot, model_path: str | None = None, feature_columns_path: str | None = None):
    model = load_model_from_path(model_path) if model_path else load_model()
    feature_columns = (
        load_feature_columns_from_path(feature_columns_path)
        if feature_columns_path
        else load_feature_columns()
    )

    df = build_feature_dataframe(feature_snapshot, feature_columns)

    probs = model.predict_proba(df)[0]
    risk_score = float(probs[1])
    predicted_label = int(model.predict(df)[0])
    confidence_score = float(max(probs))
    risk_level = get_risk_level(risk_score)

    top_factors = _explain_top_factors(model=model, df=df, cache_key=_get_model_cache_key(model_path))

    return {
    "risk_score": risk_score,
    "predicted_label": predicted_label,
    "risk_level": risk_level,
    "confidence_score": confidence_score,
    "top_factors": top_factors,
    }
```

*fig 5.3.24*

```python
threshold = db.query(RiskThreshold).first()
high_threshold = threshold.high_threshold if threshold else 0.7
medium_threshold = threshold.medium_threshold if threshold else 0.4

risk_score = prediction_result["risk_score"]
if risk_score >= high_threshold:
    risk_level = "High"
elif risk_score >= medium_threshold:
    risk_level = "Medium"
else:
    risk_level = "Low"
```

*fig 5.3.25*

Once generated , the full prediction record is saved to the database (fig 5.3.26), which also gives the system the ability to compare predictions over time. This makes sure that the system does not treat predictions as temporary outputs and allows for traceability between model output, risk classification and follow-up records.

```python
new_prediction = Prediction(
    student_id=feature_snapshot.student_id,
    feature_id=feature_snapshot.feature_id,
    model_id=active_model.model_id,
    risk_score=prediction_result["risk_score"],
    predicted_label=prediction_result["predicted_label"],
    risk_level=risk_level,
    confidence_score=prediction_result["confidence_score"],
    top_factors=prediction_result["top_factors"] if explain else None,
)

db.add(new_prediction)
db.commit()
db.refresh(new_prediction)
```

*fig 5.3.26*

## Interventions and suggestions

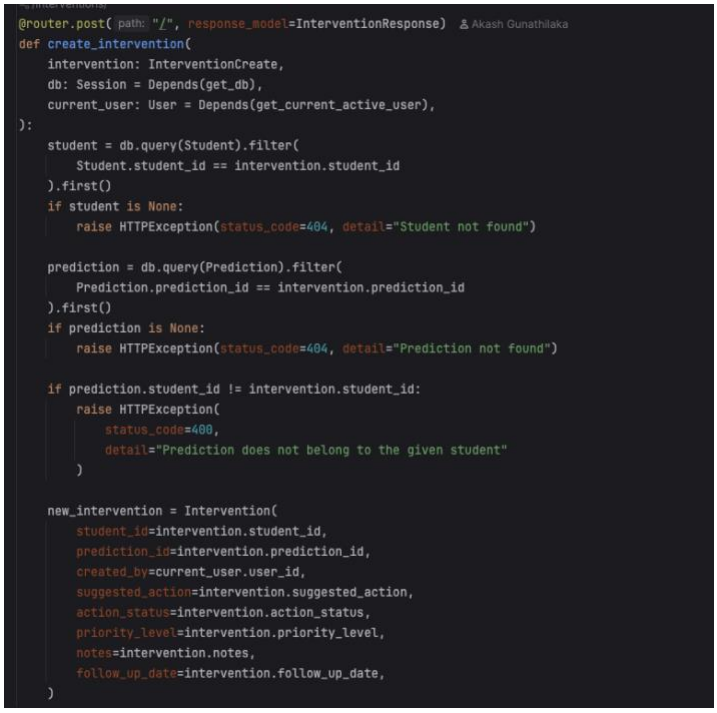
Intervention support was implemented through an API endpoint for generating rule-based suggestions and storing intervention records . A rule based suggestion endpoint GET /interventions/suggestions/{student_id} provides recommended actions based on the students current risk level using if-then rules. (fig 5.3.27)

```python
suggestions: list[str] = []
priority = "low"

# Simple rule-based suggestions
if latest_prediction.risk_level == "High":
    priority = "high"
    suggestions.extend([
        "Schedule an urgent 1:1 meeting within 7 days.",
        "Review recent assessment performance and missed submissions.",
        "Create a short weekly engagement plan and monitor VLE activity.",
        "Offer support resources (study skills, wellbeing, tutoring).",
    ])
elif latest_prediction.risk_level == "Medium":
    priority = "medium"
    suggestions.extend([
        "Send a check-in message and recommend office hours.",
        "Monitor engagement weekly and re-run prediction after new activity.",
        "Suggest targeted study resources for weak areas.",
    ])
else:
    priority = "low"
    suggestions.extend([
        "No immediate intervention required; continue routine monitoring.",
        "Encourage consistent engagement with learning materials.",
    ])

# Feature-driven if snapshot exists
if latest_snapshot is not None:
    if (latest_snapshot.avg_score or 0) < 40:
        suggestions.append("Low average score detected: recommend revision plan and assessment support.")
    if (latest_snapshot.total_clicks or 0) < 50:
        suggestions.append("Low engagement detected: encourage regular VLE activity and set weekly goals.")
    if (latest_snapshot.assessment_count or 0) == 0:
        suggestions.append("No assessments submitted in early period: check for access/issues and provide guidance.")
```

*fig 5.3.27*

The staff can create and manage intervention records through the endpoints POST /interventions. And PUT /interventions/{intervention_id}. Each record includes the suggested action, priority level, current status and follow-up date. Allowing for proper

documentation and progress tracking. All interventions are linked to both the student and the specific prediction that triggered them for traceability, allowing the system to show what actions were taken and how the case evolved (fig 5.3.28).



```python
@router.post( path: "/", response_model=InterventionResponse)  Akash Gunathilaka
def create_intervention(
    intervention: InterventionCreate,
    db: Session = Depends(get_db),
    current_user: User = Depends(get_current_active_user),
):
    student = db.query(Student).filter(
        Student.student_id == intervention.student_id
    ).first()
    if student is None:
        raise HTTPException(status_code=404, detail="Student not found")

    prediction = db.query(Prediction).filter(
        Prediction.prediction_id == intervention.prediction_id
    ).first()
    if prediction is None:
        raise HTTPException(status_code=404, detail="Prediction not found")

    if prediction.student_id != intervention.student_id:
        raise HTTPException(
            status_code=400,
            detail="Prediction does not belong to the given student"
        )

    new_intervention = Intervention(
        student_id=intervention.student_id,
        prediction_id=intervention.prediction_id,
        created_by=current_user.user_id,
        suggested_action=intervention.suggested_action,
        action_status=intervention.action_status,
        priority_level=intervention.priority_level,
        notes=intervention.notes,
        follow_up_date=intervention.follow_up_date,
    )
```

*fig 5.3.28*

## Administrative Features

Administrative functionality was implemented to allow the system to be maintained and improved without modifying the source code. All admin operations are protected by require_admin.

The admin router includes endpoints for the following operations.

- Import new data into the database

```python
</admin/data/import-oulad
@router.post("/import-oulad")  Akash Gunathilaka
def import_students_from_csv(
    dataset_id: int,
    csv_path: str,
    generate_predictions: bool = True,
    upsert_students: bool = True,
    db: Session = Depends(get_db),
    current_user: User = Depends(require_admin),
):
```

*fig 5.3.29*

- Retrain models


*fig 5.3.30*

- Activate different models


*fig 5.3.31*

- Configure risk thresholds (see fig 5.3.25)

When a model is retrained, the new artefacts are saved to disk and registered as a new ModelRecord entry. This creates a model registry that aids versioning and safe activation of different models. Risk thresholds are stored in the database and retrieved during prediction, allowing threshold values to be updated without changing the code. Additional safeguards prevent deletion of active models and ensure the backend always uses the correct model for predictions.

## Request Validation and Error handling

Request and response schemas were defined using pydantic mdodels. This ensured that all incoming data was validated early, with the correct structure, required fields, and data types, before reaching the main business logic. Schemas were useful for endpoints handling login requests, intervention creation and admin operations. (fig 5.3.32)

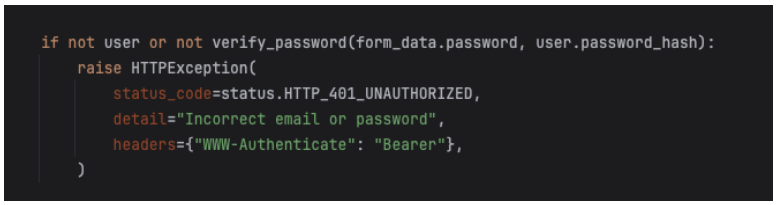



*fig 5.3.32*

The backend was also configured to return clear and appropriate HTTP status codes for common error scenarios — such as invalid credentials (401), unauthorised access (403), missing student records (404), missing feature snapshots, or when no active model is available. This approach made debugging and testing much easier, and

allowed the frontend to show more meaningful error messages to users when something went wrong.

## 5.4 Frontend Implementation

The frontend implementation focused on turning the backend functionality into a clear workflow for staff and administrators. This section focuses on how the interface was organised , how data was presented and how users interact with the main features of the system.

### Routing and Layout

Routing was used to separate the main areas of the application like the login page, dashboard, student list etc. To keep the  interface organised and made it easier to develop each part seperately.

The frontend routes were organised around the staff and admin workflows described earlier. Staff pages focus on student monitoring and interventions, while admin pages are kept separate.

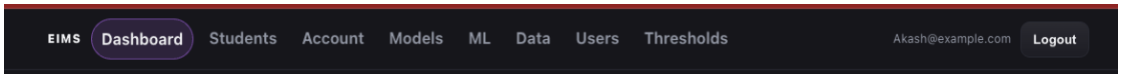



*fig 5.4.1 (admin pages)*


*fig 5.4.2 (staff pages)*

The layout was designed to support the main workflow of the system. Staff users can move from the dashboard to student search, then into a student profile, and finally to prediction and intervention actions. As the system was intended to support practical decision-making rather than simply displaying prediction results this flow was important. (fig 5.4.1, 5.4.2)

### Dashboard and Student Overview

The dashboard was implemented as the entry point for monitoring student risk. Instead of requiring users to inspect individual records, the dashboard presents summary information and risk distribution directly. This gives staff a quick overview of the current student population and helps them decide where to focus their attention. (fig 5.4.3)

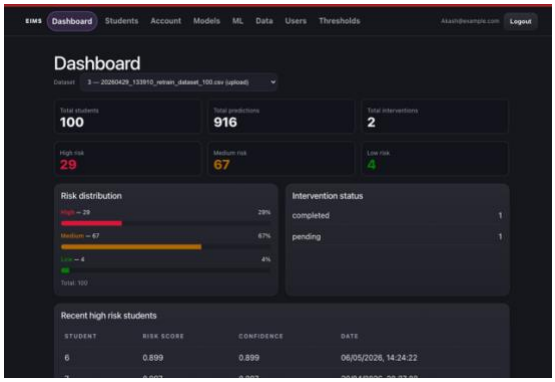



*fig 5.4.3*

Next, the student list page allows users to search and filter records. From a frontend perspective, this was important because the system may contain many student records, and staff would need a practical way to narrow the list down and select a particular student. (fig 5.4.4)

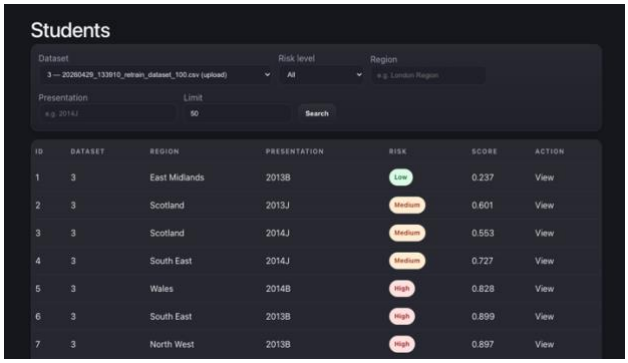



*fig 5.4.4*

## Student Profile Interface

Following this, the student profile page was designed as the main decision-support view. It brings together student details, engagement indicators, prediction results, comparison information and intervention history in one place. This reduces the need for users to move between multiple pages when reviewing an individual student. (figs 5.4.5)

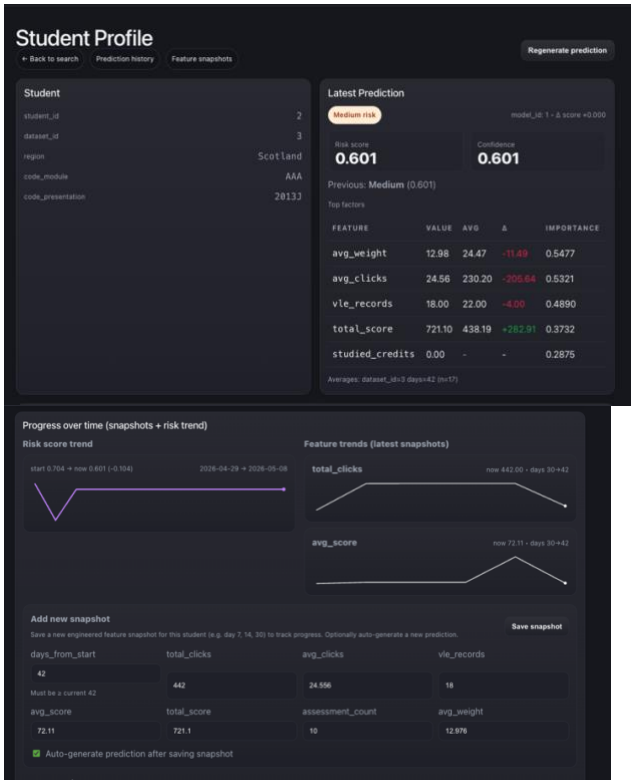



*fig 5.4.5*

Prediction results are presented in a clear format rather than raw model output. The interface also displays the risk score , risk level, confidence value, and contributing factors together. Where available, previous predictions and baseline comparisons with low-risk students are also shown to give additional context when interpreting the result. (fig 5.4.5, 5.4.6)

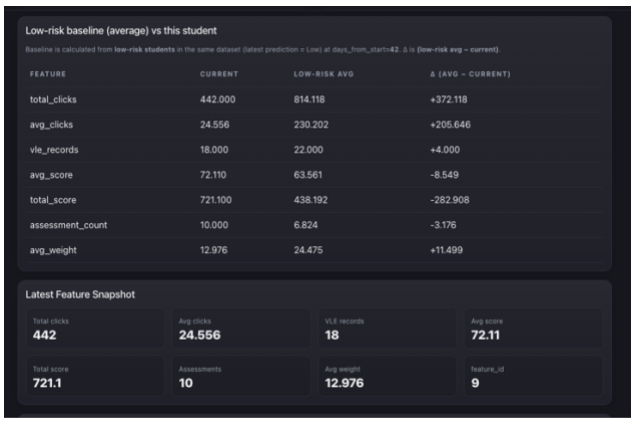



*fig 5.4.6*

## Forms, Actions and Feedback

Subsequently, forms were implemented for key actions such as logging/updating new interventions, and performing administrative takss. These forms were kept simple and focused only asking for the information acually needed for each task to reduce user confusion and minimise errors (Fig 5.4.7).

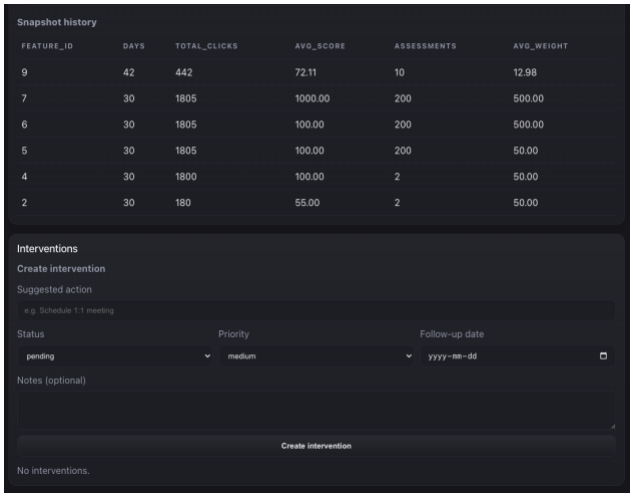



*Fig 5.4.7*

After completing actions like generating a prediction or saving an intervention, the interface automatically refreshes the relevant sections so staff can immediately see the outcome. Basic success messages and clear error feedback were added to improve usability when something goes wrong (fig 5.4.8 – 5.4.10).

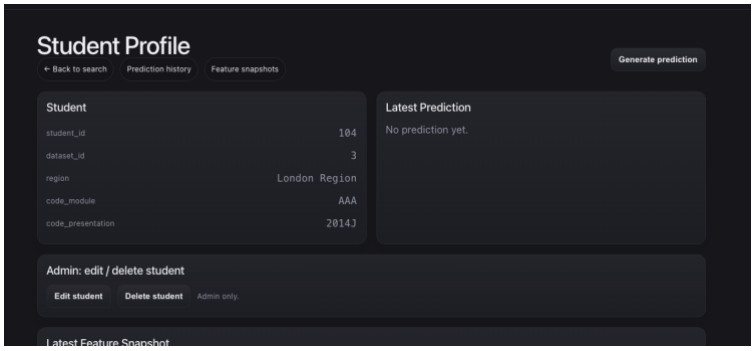



*fig 5.4.8 (before prediction)*

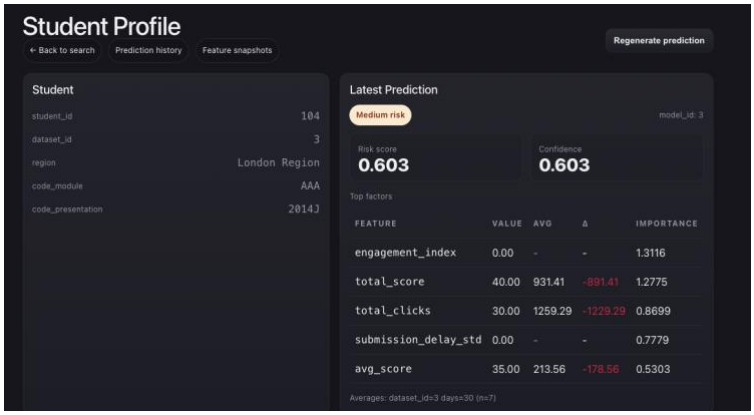



*fig 5.4.9 (after prediction)*

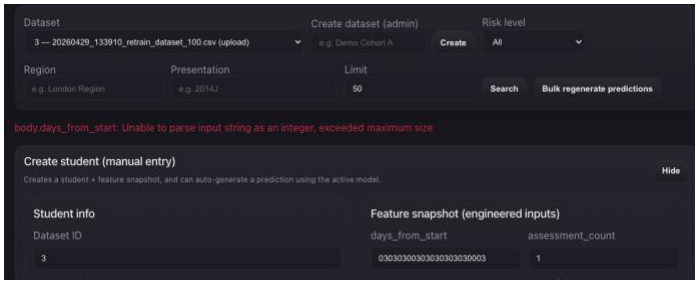
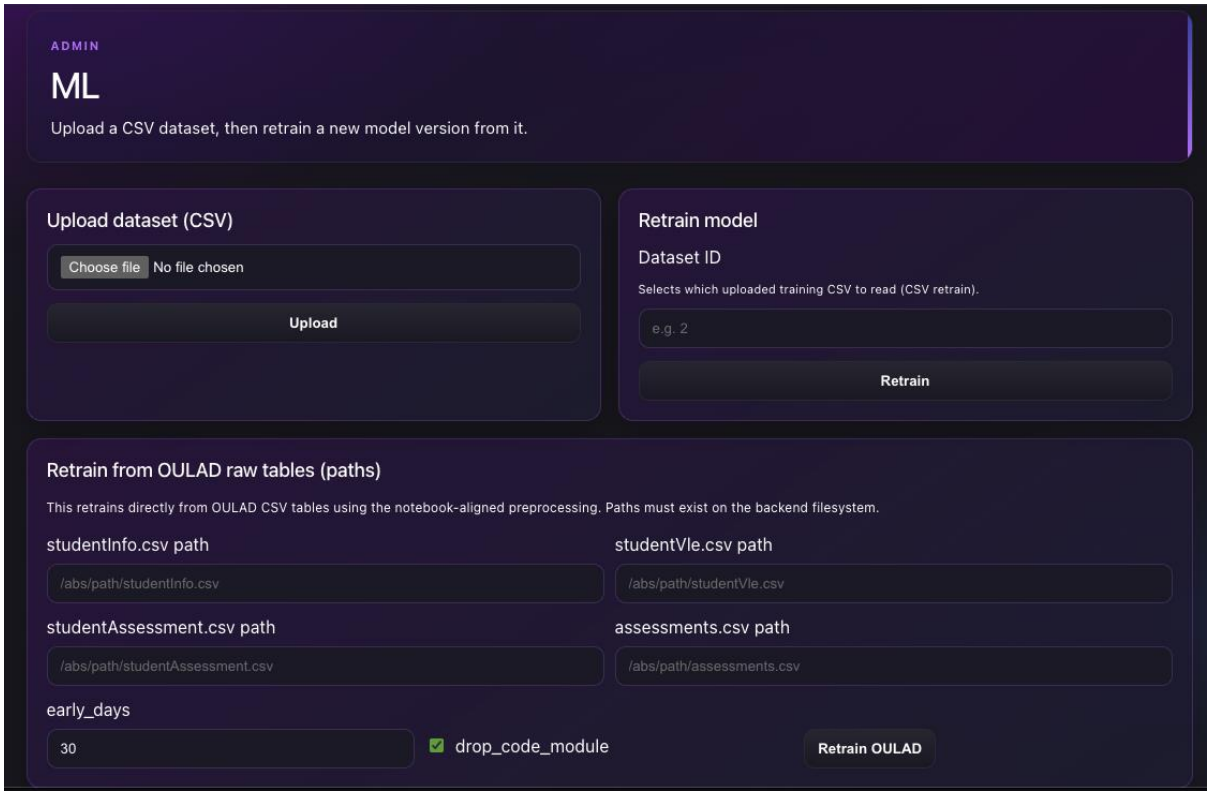
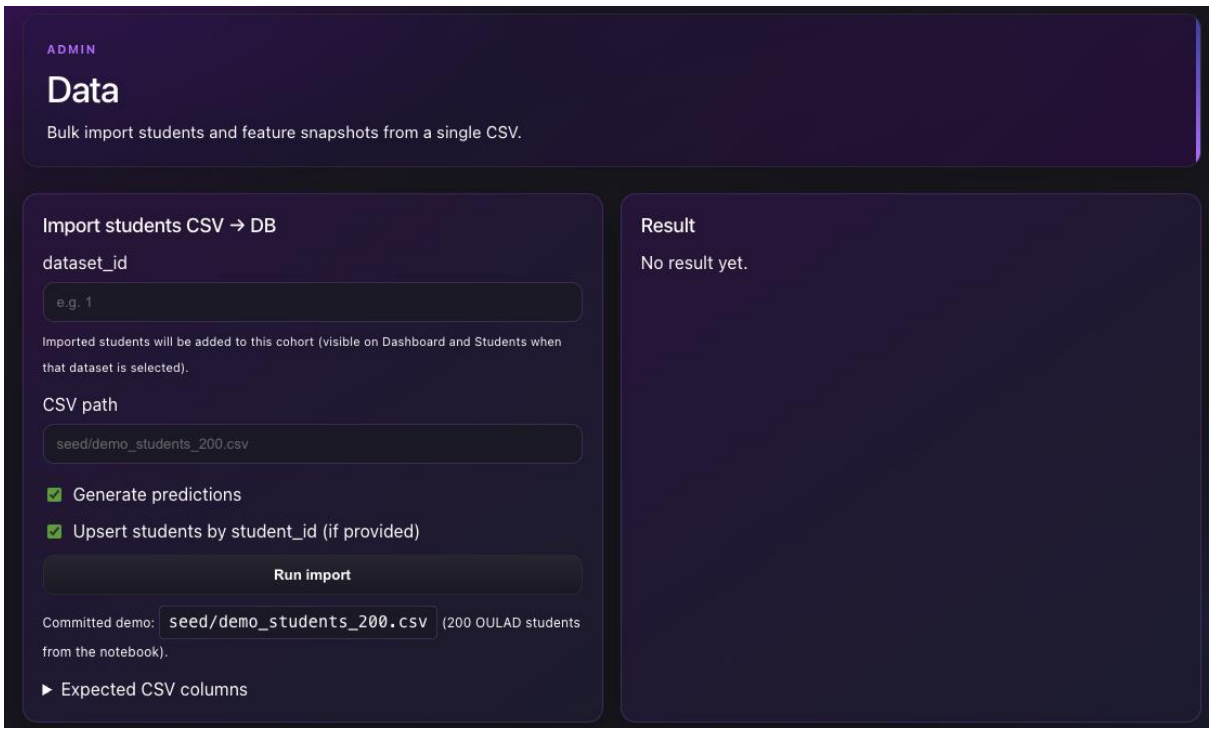



*fig 5.4.10*

## Admin Interface

The admin interface was built as a separate section , distinct from the main staff workflow. It includes pages for dataset upload and management, model retraining, risk threholds configuration and student record maintenance. This functionality is hidden from users logged in as staff to avoid confusion (see figs 5.4.1 , 5.4.2). These pages mainly act as frontend entry points for backend administrative operations, so the frontend implementation focuses on providing clear forms and feedback messages (figs 5.4.11)


ADMIN
ML
Upload a CSV dataset, then retrain a new model version from it.
Upload dataset (CSV)
Choose file
No file chosen
Upload
Retrain model
Dataset ID
Selects which uploaded training CSV to read (CSV retrain).
e.g. 2
Retrain
Retrain from OULAD raw tables (paths)
This retrains directly from OULAD CSV tables using the notebook-aligned preprocessing. Paths must exist on the backend filesystem.
studentInfo.csv path
/abs/path/studentInfo.csv
studentVle.csv path
/abs/path/studentVle.csv
studentAssessment.csv path
/abs/path/studentAssessment.csv
assessments.csv path
/abs/path/assessments.csv
early_days
30
drop_code_module
Retrain OULAD


ADMIN
Data
Bulk import students and feature snapshots from a single CSV.
Import students CSV → DB
dataset_id
e.g. 1
Imported students will be added to this cohort (visible on Dashboard and Students when that dataset is selected).
CSV path
seed/demo_students_200.csv
Generate predictions
Upsert students by student_id (if provided)
Run import
Committed demo:
seed/demo_students_200.csv
(200 OULAD students from the notebook).
Expected CSV columns
Result
No result yet.

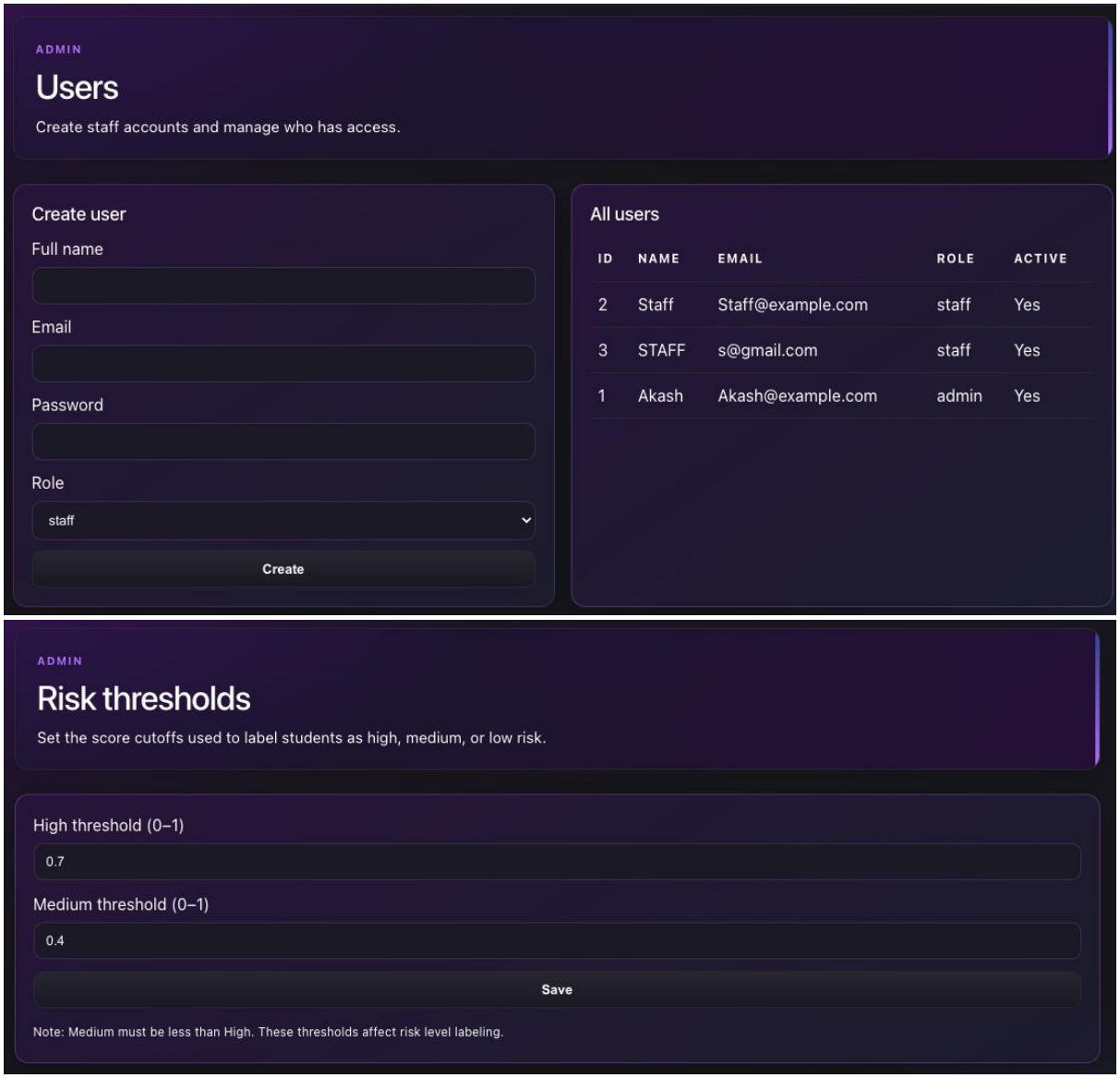



*Figure 5.4.11*

# 6. Evaluation

## 6.1 Evaluation Aims

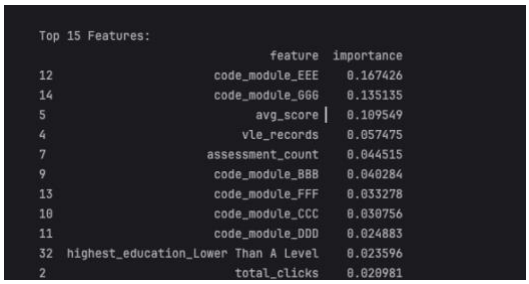
The goal of this evaluation was to assess whether the system met the project aims. Since the project is a web application with ML. The evaluation considered factors like predictive performance, requirement coverage, usability, security, ethical issues and practical limitations.

## 6.2 Predictive Model Evaluation

This section evaluates whether the model was suitable for use in the system. The model was assessed using the test set created during implementation and predictive performance, baseline comparison and practical limitations.

### Feature Selection Evaluation

Feature Selection was evaluated in depth because the first version of the model showed code_module as one of the most important features. This was a concern as this signalls that the model may have been learning module difficulty rather than student engagement patterns.

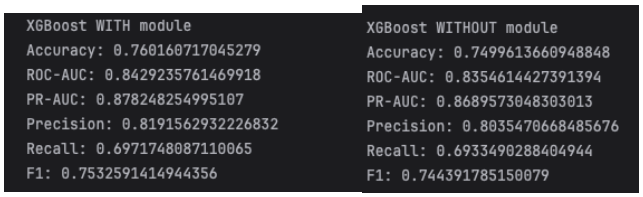


Top 15 Features:

| | feature | importance |
|---|---|---|
| 12 | code_module_EEE | 0.167426 |
| 14 | code_module_GGG | 0.135135 |
| 5 | avg_score | 0.109549 |
| 4 | vle_records | 0.057475 |
| 7 | assessment_count | 0.044515 |
| 9 | code_module_BBB | 0.040284 |
| 13 | code_module_FFF | 0.033278 |
| 10 | code_module_CCC | 0.030756 |
| 11 | code_module_DDD | 0.024883 |
| 32 | highest_education_Lower Than A Level | 0.023596 |
| 2 | total_clicks | 0.020981 |

*fig 6.1*

To test this, the model was trained and compared both with and without code_module. Removing this only caused a small decresase in performance, but it reduced the risk of the model depending too much on the code module. Because of this the final model was trained without the code_module. Making sure the model was not learning patterns that are not directly related to students.



XGBoost WITH module
Accuracy: 0.760160717045279
ROC-AUC: 0.8429235761469918
PR-AUC: 0.878248254995107
Precision: 0.8191562932226832
Recall: 0.6971748087110065
F1: 0.7532591414944356

XGBoost WITHOUT module
Accuracy: 0.7499613660948848
ROC-AUC: 0.8354614427391394
PR-AUC: 0.8689573048303013
Precision: 0.8035470668485676
Recall: 0.6933490288404944
F1: 0.744391785150079

*fig 6.2*

### Model Performance Metrics

The model was evaluated using metrics such as accuracy, precison, recall, F1-score, ROC-AUC and the confusion matrix. These metrics were chosen because they show performance from different perspectives. Accuracy gives an overall measure, while recall is especially important because it shows how many actual at-risk students were correctly identified. Precision shows how many predicted at-risk students were truly at risk, helping assess false alerts. F1-score balances precision and recall, ROC-AUC

shows how well the model seperates the two classes across thresholds, and the confusion matrix gives a clearer view of false positives and false negatives. (Figs 6.2, 6.3)

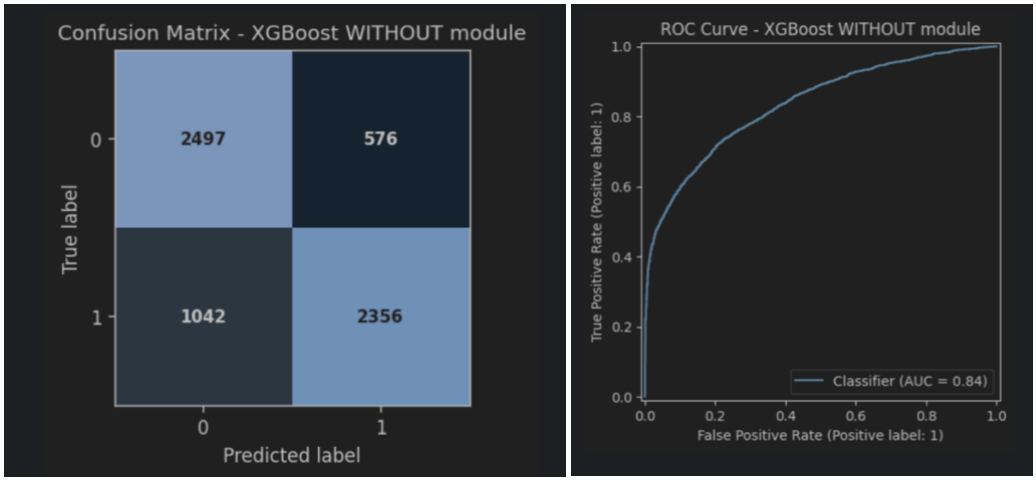



*fig 6.3*

These results combined suggest that the model is suitable for use in the system. However, the metrics were not treated as proof the model is ready for real deployment. Since the dataset and context are limited, we still need further testing and real institutional data before it can be trusted for live deployment.

## Baseline Comparison

The model was also compared with a Logistic Regression baseline model. This gave me a simple reference point to check whether the more advanced model was adding value. The XGBoost model performed better overall (fig 6.4), which further supported my decision for choosing this as the final model.

```
XGBoost WITHOUT module
Accuracy: 0.7499613660948848
ROC-AUC: 0.8354614427391394
PR-AUC: 0.8689573048303013
Precision: 0.8035470668485676
Recall: 0.6933490288404944
F1: 0.744391785150079
```

```
Logistic Regression WITHOUT module
Accuracy: 0.7141091021480451
ROC-AUC: 0.8015819971817805
PR-AUC: 0.8339535789054251
Precision: 0.727781047675103
Recall: 0.727781047675103
F1: 0.727781047675103
```

*fig 6.4*

## Threshold and Risk Evaluation

The system does not only return a raw prediction score. It converts the score into a risk level. This makes the output easier for staff to understand because a risk category is easier to understand than a number. (Fig 6.5)

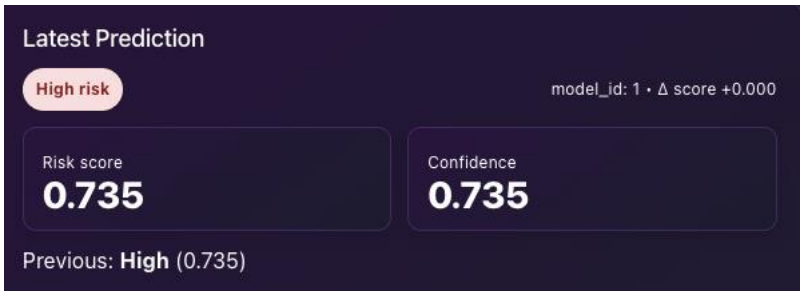



*Fig 6.5*

The risk thresholds were made configurable by admins. Because different institutions may identify risks differently. For example, one institution may want to flag more students early, while another may only want to flag students with a very high risk score due to limited support resources. (Fig 6.6)

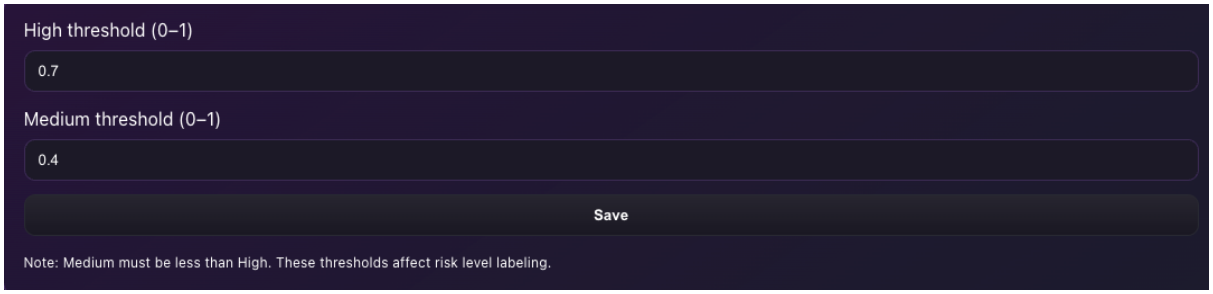



*Fig 6.6*

However the configurable risk thresholds have a trade-off. Lowering the threshold may help identify more at-risk students, but it may also increase false positives. Raising the threshold may reduce false positives but increase the number of at risk students missed. Because of this, the thresholds should not be treated as fixed or perfect.

## Error Analysis

It is important to recognize where the model could make wrong predictions. One possible issue is a borderline case. If a students risk score is close to a threshold, a small change in the input data could move them from one category to another, so staff should be careful when choosing medium risk or high risk predictions.

Furthermore, incomplete or noisy data could mean the model does not have enough reliable information to make a strong prediction, this is especially the case early in a course, when thete may not be enough behavioural data to clearly identify risk

There is also a risk that the model may rely on proxy features. For example, low engagement may be linked to academic risk, but it does not always mean a student is disengaged or failing. A student may have low activity because of illness and a number of other things, these factors are not visible in the dataset. Because of this the prediction should just be treated as a support signal rather than a final judgement.

### Reproducibility of the model

Reproducibility was crucial because the backend needed to use the same model and feature structure that were created during training. Saving the trained model artefacts, feature columns and model metrics allowed the same model to be loaded again for runtime prediction.

This improved reliability because predictions were linked to saved model records rather than temporary training outputs. It also made the system easier to review, since the information can still be accessed after training.

However, this was still limited as the model was trained on a fixed public dataset. It would need stronger experiment tracking and version control for datasets for it to be suited as a production ready system.

### Requirements Coverage

The system was evaluated against the objectives and requirements defined earlier in the report. The main requirements were covered well by the prototype. The admin requirements were also met, these features highlight that the system can be adapted and maintained rather than being a fixed demonstration

Howerver, some requirements should be understood within the limits of the prototype. For example the system demonstrate an early intervention workflow but it is not connected to a real university system. The explainability features are also basic and should not be treated as complete explanations of student behavior.

## Verification and Validation

Verification and validation were carried out using both automated and manual testing. Automated testing was used to check important backend behavior. While manual scenario testing was used to confirm that the full system workflow works through the interface.

Backend smoke tests were creating using pytest and FastAPIs TestClient. These tests checked important flows such as authentication protect, login , student search and intervention suggestions to name a few. This gave confidence the backend endpoints behaved as expected. (Fig 6.5)

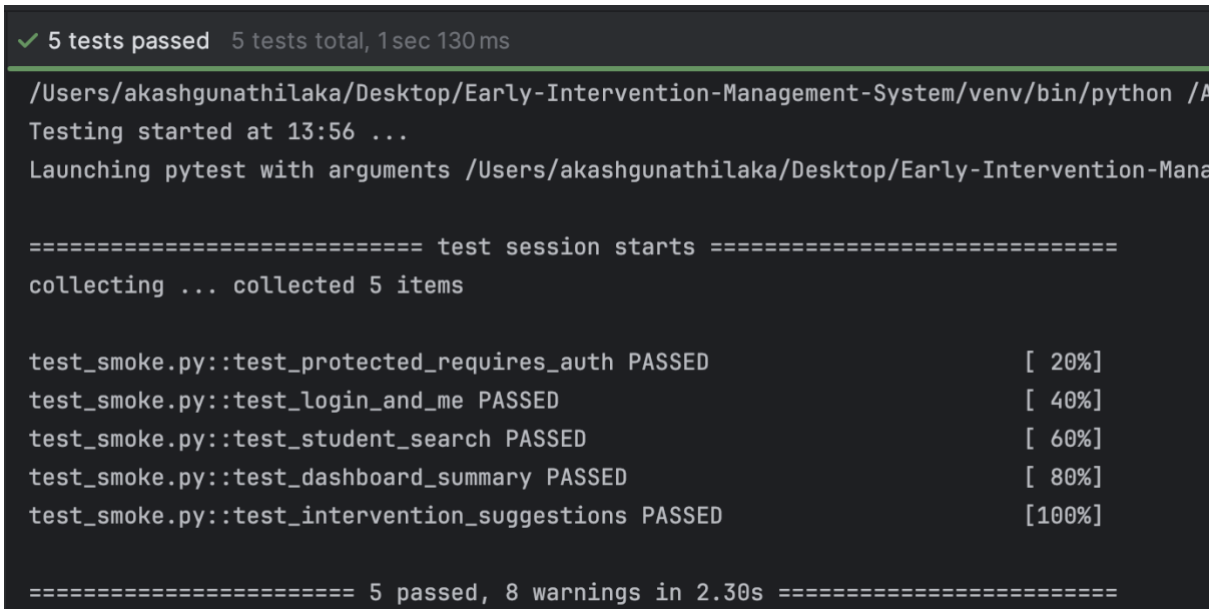



*Fig 6.5*

Manual testing was done as the system was a full-stack application. The staff workflow was tested from login through to intervention and updating. The admin workflow was also tested which included features like data handling , model retraining, threshold configuration and student management. (Fig 6.6)

| Test Plan | | | | |
|---|---|---|---|---|
| **Test Case** | **Steps** | **Expected Result** | **Actual Result** | **Status** |
| TC-01 Protected route requires auth | Call GET /students/ without token | Request rejected (401/403) | Request rejected (401/40 | Passed |
| TC-02 Login issues token | Submit valid credentials to POST /auth/login | 200 + JWT returned | 200 + JWT returned | Passed |
| TC-03 Get current user | Call GET /auth/me with token | 200 + user object (email, role, is_ | 200 + user object (email, | Passed |
| TC-04 RBAC blocks non-admin | Log in as staff → access /admin/models (or call an admin endpoint) | Access denied (403) and/or redir | Access denied (403) and/ | Passed |
| TC-05 Change password | Logged-in user → POST /auth/change-password | Password updated; can login wit | Password updated; can l | Passed |
| TC-06 Password reset (prototype) | POST /auth/request-password-reset → use token → POST /auth/reset-pas | Password reset succeeds; login | Password reset succeeds | Passed |
| TC-07 Session expiry (JWT) | Use an expired/invalid token on any protected endpoint | 401 returned; frontend logs out | 401 returned; frontend log | Passed |
| TC-08 Idle timeout (frontend) | Log in → remain inactive past VITE_IDLE_TIMEOUT_MINUTES | Auto logout + redirect to /login | Auto logout + redirect to / | Passed |
| TC-09 Student list | GET /students/ | 200 + list of students | 200 + list of students | Passed |
| TC-10 Student search basic | GET /students/search?limit=5 | 200 + list of 5 | 200 + list of 5 | Passed |
| TC-11 Student search filters | Call /students/search with dataset_id, region, code_presentation | Results match filters | Results match filters | Passed |
| TC-12 Filter by risk level | GET /students/search?risk_level=High | Only High-risk students returned | Only High-risk students r | Passed |
| TC-13 Student profile aggregation | GET /students/{id}/profile | Returns student + latest snapsh | Returns student + latest s | Passed |
| TC-14 Feature snapshot history | GET /feature-snapshots/student/{id} | Returns snapshot history list | Returns snapshot history | Passed |
| TC-15 Prediction history | GET /predictions/student/{id} | Returns prediction history list | Returns prediction history | Passed |
| TC-16 Generate prediction (single student) | POST /predictions/generate/{id} | New prediction created with risk | New prediction created w | Passed |
| TC-17 Compare predictions | Ensure ≥2 predictions → GET /predictions/compare/{id} | Latest + previous + delta returne | Latest + previous + delta | Passed |
| TC-18 Low-risk baseline averages | GET /students/{id}/feature-averages | Returns low-risk baseline averag | Returns low-risk baseline | Passed |
| TC-19 Intervention suggestions | After prediction → GET /interventions/suggestions/{id} | 200 + suggestions array | 200 + suggestions array | Passed |
| TC-20 Create intervention | POST /interventions/ with required fields | Intervention created and appears | Intervention created and a | Passed |
| TC-21 Update intervention | PUT /interventions/{intervention_id} status/notes/follow-up | Intervention updated and persist | Intervention updated and | Passed |
| TC-22 Admin edit student (RBAC) | Admin → PUT /students/{id} | Student updated | Student updated | Passed |
| TC-23 Admin delete student (RBAC + cascade) | Admin → DELETE /students/{id} | Student deleted; related snapsh | Student deleted; related s | Passed |
| TC-24 Upload dataset (admin) | POST /admin/ml/upload-dataset with CSV | Dataset file saved in uploads/ + ( | Dataset file saved in uplo | Passed |
| TC-25 Retrain model (admin) | POST /admin/ml/retrain?dataset_id=... | New model record created; pred | New model record create | Passed |
| TC-26 Activate model (admin) | PUT /admin/models/activate/{model_id} | Model becomes active; active m | Model becomes active; a | Passed |
| TC-27 View model metrics (admin) | GET /admin/models/{model_id}/metrics-file | Returns metrics JSON (or clear 4 | Returns metrics JSON (or | Passed |
| TC-28 Configure risk thresholds (admin) | PUT /admin/risk-thresholds/ then regenerate prediction | Thresholds saved; categorisatior | Thresholds saved; categc | Passed |
| TC-29 Bulk generate predictions (admin) | POST /predictions/generate-dataset/{dataset_id} | Returns attempted/generated/fai | Returns attempted/gener | Passed |
| TC-30 Error handling (not found) | Request missing student GET /students/999999/profile | 404 with clear message | 404 with clear message | Passed |

*Figure 6.6*

## 6.3 Usability Evaluation

Here we focussed on whether the system supports a clear workflow for staff and administrators. The evaluation was task based, the aim was to check whether the main actions could be completed without confusion and whether the system presented the information in a useful way.

The staff workflow was tested by moving through the main sequence of actions, logging in, viewing the dashboard, finding the student, opening their profile, generating a prediction and then reviewing the risk information and recording an intervention. This confirmed that the core workflow of the project was functional and the prediction results were connected to a follow up action rather than being an isolated output.

The administrator workflow was tested seperately, this included checking whether dataset management, model management could be accessed and properly used In the admin pages. Making sure that all complex admin funcionality worked as needed.

The main usability strength was that the system connects prediction and intervention in one workflow. However, the usability evaluation was limited because it was not carried out with a large number of real academic staff. A more complete evaluation would be needed to accurately state the usability of the system.

## 6.4 Security Evaluation

The security evaluation focused on checking whether the main access control requirements were met in practice. The key tests were whether unauthenticated users

were blocked from protected areas and whether staff users were prevented from admin only functions.

Attempts to access protected backend routes without a valid token were rejected, and admin only operations were not available to staff users. This was confirmation that the security restrictions were enforeced properly rather than just being hidden (Fig 6.7).

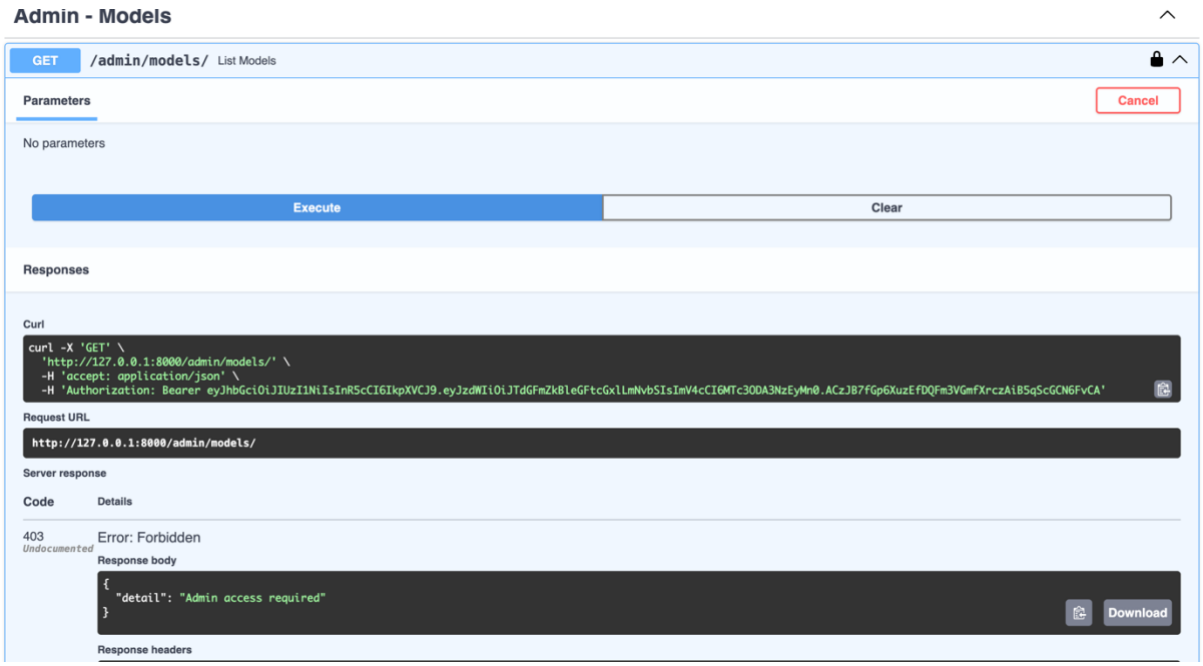



*Fig 6.7*

The system also met the basic requirements for password protection as we discussed previously on the report by password hashing, and session management (Fig 6.8)

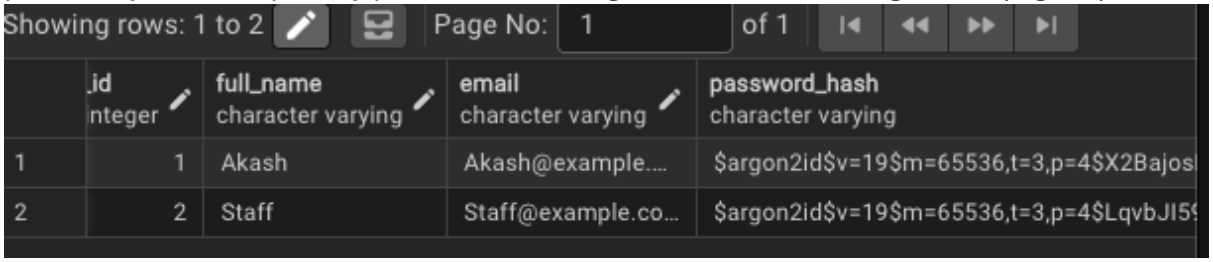



*Fig 6.8*

However, there are security limitations. JWT tokens are stored in localStorage, which has risks if the frontend is subjected to cross-site scripting. The system also does not currently include refresh tokens or server-side token revocation. This means the authentication design would need to be improved before any real deployment.

## 6.5 Ethical and Practical Evaluation

Ethical considerations are important because this system deals with student risk prediction. Even though the project uses public data, the system is based on a real-world problem where predictions could affect how staff respond to students.

Privacy was addressed by making sure there were no identifiable data to help remove the risk of exposing student information, this was already done with the OULAD dataset and, any other data used in the system was synthetic.

Another key concern with prediction projects is fairness and bias. A model may learn patterns from the data that do not fully represent the reasons behind student behaviour. For example, low engagement may be linked to academic risk, but it may also be caused by personal circumstances that the model cannot see. This is a key reason why the system should be viewed as a decision support tool rather than a final decision maker.

The system was designed to keep a human in the loop. Staff users are expected to review the risk score, confidence value, contributing factors, baseline comparisons, and student context before deciding what action to take. The system provides suggestions, but it does not automatically contact students or apply consequences.

There is still a risk of harm from false positives and false negatives. A false positive may cause unnecessary concern or extra staff workload, while a false negative may mean a student who needs help is missed. The system tries to reduce this risk by presenting predictions as decision-support information and by allowing staff to record interventions rather than treating the model output as a final decision.

## 6.6 Limitations

The evaluation shows that the prototype meets the main aims of the project, but there are still several limitations.

The first limitation is model generalisability. The model was developed using public, anonymised, or synthetic data, so it cannot fully represent the complexity of a real university environment. Real institutional data may include different behaviour patterns, course structures, assessment schedules. Because of this, the model would need to be retrained and evaluated using real institutional data before deployment.

Another limitation is explainability. The system provides contributing factors and baseline comparisons, but these are still basic explanations. They help staff understand the prediction, but they do not prove why a student is at risk. Future work could use stronger explainability methods and clearer staff-facing explanations.

The system is also limited as a prototype application.It does not use live data pipelines, and does not send automatic alerts when a student becomes high risk. These features would be important in a production version.

Another area that could be improved is security. A real deployment would need stronger authentication handling, refresh tokens, server-side token revocation, audit logs, and more detailed monitoring of admin actions.

Future improvements should include stronger model evaluation, such as cross-validation, threshold analysis, calibration checks, confusion matrix analysis, and subgroup fairness checks. The system should also be tested with real academic staff to understand whether the dashboard, prediction explanations, and intervention workflow are clear and useful in practice

# 7. Critical Reflection and lessons Learned

Looking back at this project, I am proud that I managed to build more than just another machine learning model. Early on, it would have been easier to focus only on trained a model and reporting its accuracy. However, as the work progressed, I realised that a prediction by itself has limited value unless it is embedded in a system that staff can acually understand and use. This realisation shifted the project from purely a technical modelling project to a full early intervention platform.

This decision made the project more challenging that I initially anticipated. Integrating machine learning with a full stack web application and comprehensive documentation into one cohesive system was demanding. In hindsight, I understimated the time required to properly document the work as I went along. The implementation moved faster than the report, which mean I had to revisit earlier decisions and add explanations later. One clear lesses is that for projects of this scale, documentation should develop with the system rather being left behind.

## 7.1 Reflection on Technical Decisions

Several technical choices worked well for the scope of the project. Using XGBoost proved to a good decision, it delivered strong performance while offering feature importance scores that support basic explainability. I had considered more complex temporal models such as LSTMs, but they would have increased development time and reduced transparency, which was not useful for a staff facing tool.

The choice of a relational database also became increasingly more important. It started of as a standard decision and turned out to be crucial for maintaining traceability between students, predictions and interventions. On the other hand, some decisions had clear weaknesses. At the time I priortised getting a working authentication flow completed, but looking back I would have planned security more carefully from the start. A better version of the system would use stronger token handling , server-side revocation and audit logging.

## 7.2 Reflection on Scope and Project Management

The biggest challenge in the project was scope control. I wanted the system to feel realistic, so I kept adding features that made sense for a early interevention platform. This increased the amount of work significantly.

While many of these additions were valuable, I should have defined a stricter minimum viable product earlier as I also needed to focus on evaluation and writing. This does not

mean the features were unnecessary, but it showed the importance of balancing ambition with available time. I understand that a technically strong project can still lose marks if the report does not explain the work clearly enough.

The project also taught me that implementation progress can create a false sense of completion. When the system started working, it felt like the project was nearly finished. However the academic part of the project still required explanation, justification testing and more to be truly complete.

## 7.3 Reflection on evaluation

This is one area I think the project could have done stronger.While I included multiple metrics and considered class imbalance I understand that metrics only show part of the picture. For an early intervention system, it is not enough to know whether the model performs better overall, it is also important to understand what happens at different risk thresholds and how many at risk students may be missed.

To fix this, I would have added more detailed threshold analysis. This would have made the evaluation more meaningful because the system uses configurable risk thresholds in practice. I would also have liked to include subgroup analysis to check whether the model performs differently for different types of students. I believe this would have stregnthened the project as a whole.

## 7.4 Lessons learned and future improvements

One of the main lessons I learned from this project is that building a useful machine learning system is very different from simply training a good model. I realised that the model only becomes truly valuable when it is connected to a practical workflow where users can understand the results and take action. This changed my view of the entire project. Features like the dashboard, student profile and intervention logging became just as important as the accuracy of the model itself.

If I were to do this project again, I would define the core scope more strictly from the beginning. The essential workflow should have been prioritised first: viewing students, generating predictions, interpreting risk, and recording interventions. Features such as model retraining, threshold configuration, and advanced admin controls were useful, but they also increased the workload significantly. In future projects, I would separate

essential features from optional improvements more clearly so that the core system and evaluation are completed before expanding the scope.

Another important lesson was the value of writing the report continuously alongside development. In this project, the implementation moved ahead faster than the documentation, which meant that some technical decisions and testing evidence had to be added later. Although the system became stronger, the reporting process became harder because I had to remember and explain decisions after they had already been implemented. Next time, I would document each major feature as soon as it was completed, including things like design reasoning, implementation details, and testing evidence.

The project also helped me understand the importance of traceability in decision-support systems. Storing predictions, interventions, model records, and thresholds made the system more professional because it created a record of what happened and why. This is especially important in an educational context, where prediction outputs may influence how staff respond to students. In future work, I would extend this further by adding audit logs so that important actions, especially admin actions, could be reviewed more clearly.

In regards to machine learning, I now understand that evaluation needs to go beyond general performance metrics. Accuracy, precision, recall, F1-score, and ROC-AUC are useful, but they do not fully show how the system behaves in practice. If I were improving the project, I would include cross-validation, confusion matrices, threshold analysis, calibration checks, and subgroup performance analysis. This would provide a clearer understanding of how reliable the model is and whether it performs fairly across different groups of students.

Security needs to be planned early rather than added only when required. The current authentication system is suitable for a prototype, but a production version would need stronger token handling, refresh tokens, server-side token invalidation, and more detailed monitoring of sensitive actions. These improvements would be necessary before the system could be used in a real institutional environment.

Finally, I learned that user feedback is essential for a system like this. Although I tested the system myself and checked that the workflows functioned correctly, this is not the same as testing with real academic staff. A real user might interpret risk scores differently, misunderstand explanations, or need clearer guidance before recording an intervention. In future, I would involve users earlier and conduct a small usability study to improve the dashboard, prediction explanations, and intervention workflow.

Overall, this project improved my skills in full-stack development, machine learning integration, database design, testing, and ethical thinking. More importantly, it changed how I think about AI systems. I now understand that technical performance is only one part of the problem. For systems used in education, transparency, usability, fairness, traceability, and human judgement are just as important as prediction accuracy.

# 8. Conclusion

The project set out to address a clear gap in the field of learning analytics: the lack of practical systems that connect strong prediction models with tools academic staff can actually use. I believe the developed prototype successfully achieves this goal, by integrating a XGBoost model with a full-stack web application, the system allows staff to search for students, view risk predictions with contributing factors and record interventions in one connected workflow.

The work demonstrates that it is possible to go beyond a standalone prediction model and create a functional early intervention platform. Features such as historical prediction tracking, intervention logging, tunable risk thresholds and retraining models give the system the potential for ongoing improvement and adaptation to different institutions. Although this reamins a prototype and has not been tested on a live university setting, it provides a solid and practical foundation for future devlopment.

Overall, this project has been a valuable learning journey. Significantly stregnthening my skills in full-stack development, machine learning integration, database design and AI system building. More importantly, it changed how I think about applying technology in education. Technical performance is important, but transparency, usability, fairness, and human judgfement are equally essential when creating systems that can influence real student outcomes.

I look forward to building on this foundation in future work, particularly by incorporating real institutional data, enhancing explainability and strengthening the security with proper user testing from academic staff. This project has showed me the impact that AI tools can have on the real world.

# 9. References


1. *Albreiki, B., Zaki, N. and Alashwal, H. (2021) 'A systematic literature review of student's performance prediction using machine learning techniques',* Education Sciences*, 11(9), article 552. Available at: https://doi.org/10.3390/educsci11090552*

2. *Papamitsiou, Z. and Economides, A.A. (2014) 'Learning analytics and educational data mining in practice: A systematic literature review of empirical evidence',* Educational Technology & Society*, 17(4), pp. 49–64. Available at: https://www.researchgate.net/publication/267510046_Learning_Analytics_and_Educational_Data_Mining_in_Practice_A_Systematic_Literature_Review_of_Empirical_Evidence*

3. *Kovačić, Z.J. (2010) 'Early prediction of student success: Mining students' enrolment data',* Proceedings of the Informing Science & IT Education Conference (InSITE 2010)*, pp. 647–665. Available at: https://doi.org/10.28945/1281*

4. *Sekeroglu, B., Dimililer, K. and Tuncal, K. (2019) 'Student performance prediction and classification using machine learning algorithms', in* Proceedings of the 2019 8th International Conference on Educational and Information Technology. *New York: ACM, pp. 7–11. Available at: https://doi.org/10.1145/3318396.3318419*

5. *Ahmed, E. (2024) 'Student performance prediction using machine learning algorithms',* Applied Computational Intelligence and Soft Computing*, 2024, article ID 4067721, pp. 1–15. Available at: https://doi.org/10.1155/2024/4067721*

6. *Kloft, M., Stiehler, F., Zheng, Z. and Pinkwart, N. (2014) 'Predicting MOOC dropout over weeks using machine learning methods', in* Proceedings of the EMNLP 2014 Workshop on Analysis of Large Scale Social Interaction in MOOCs. *Doha: Association for Computational Linguistics, pp. 60–65. Available at: https://doi.org/10.3115/v1/W14-4111*

7. *He, J., Bailey, J., Rubinstein, B.I.P. and Zhang, R. (2015) 'Identifying at-risk students in massive open online courses', in* Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence. *Austin, TX: AAAI Press, pp. 1749–1755. Available at: https://doi.org/10.1609/aaai.v29i1.9471*

8. *Whitehill, J., Mohan, K., Seaton, D., Rosen, Y. and Tingley, D. (2017) 'Delving deeper into MOOC student dropout prediction',* arXiv preprint*, arXiv:1702.06404. Available at: https://doi.org/10.48550/arXiv.1702.06404*

9. *Fei, M. and Yeung, D.Y. (2015) 'Temporal models for predicting student dropout in massive open online courses', in* 2015 IEEE International Conference on Data Mining Workshop (ICDMW). *Atlantic City, NJ: IEEE, pp. 256–263. Available at: https://doi.org/10.1109/ICDMW.2015.174*

10. *Kuzilek, J., Hlosta, M. and Zdrahal, Z. (2017) 'Open University Learning Analytics dataset',* Scientific Data*, 4, article 170171. Available at: https://doi.org/10.1038/sdata.2017.171*

11. *Jha, N.I., Ghergulescu, I. and Moldovan, A.N. (2019) 'OULAD MOOC dropout and result prediction using ensemble, deep learning and regression techniques', in* Proceedings of the 11th International Conference on Computer Supported Education (CSEDU 2019). *Heraklion: SCITEPRESS, pp. 154–164. Available at: https://doi.org/10.5220/0007767901540164*

12. *Márquez-Vera, C., Cano, A., Romero, C., Noaman, A.Y.M., Fardoun, H.M. and Ventura, S. (2016) 'Early dropout prediction using data mining: A case study with high school students',* Expert Systems*, 33(1), pp. 107–124. Available at: https://doi.org/10.1111/exsy.12135*

13. *Arnold, K.E. and Pistilli, M.D. (2012) 'Course Signals at Purdue: Using learning analytics to increase student success', in* Proceedings of the 2nd International Conference on Learning Analytics and Knowledge. *New York: ACM, pp. 267–270. Available at: https://doi.org/10.1145/2330601.2330666*

14. *Chen, T. and Guestrin, C. (2016) 'XGBoost: A scalable tree boosting system', in* Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. *New York: ACM, pp. 785–794. Available at: https://doi.org/10.1145/2939672.2939785*

15. *He, H. and Garcia, E.A. (2009) 'Learning from imbalanced data',* IEEE Transactions on Knowledge and Data Engineering*, 21(9), pp. 1263–1284. Available at: https://doi.org/10.1109/TKDE.2008.239*