PROJECT REPORT

AI-POWERED STUDENT PERFORMANCE CHATBOT

ILearn - Student Performance Analysis System

Submitted By:
1: Rafia Shahid (3351)

2:Alishba Ishtiaq (ADC-54)

3:Zeeshan Ahmed (3392)

4:Muhammad Abdullah Khan (3377)

Submitted To:
Sir Ahtesham

Course:
Advance Programming

Table of Contents

CHAPTER 1: INTRODUCTION

1.1 INTRODUCTION

In today's educational landscape, managing and analyzing student performance data has become increasingly complex. Educators need efficient tools to track student progress, identify areas of improvement, and provide personalized feedback. Traditional methods of student performance analysis often involve manual data processing and static reporting systems that lack flexibility and real-time insights.

ILearn is an innovative AI-powered web application designed to revolutionize how educators interact with student performance data. The system combines traditional database management with cutting-edge artificial intelligence to create an intelligent chatbot that can answer natural language questions about student performance. By leveraging Retrieval-Augmented Generation (RAG) technology, ILearn provides educators with instant, context-aware insights about individual students, their academic performance, attendance patterns, and areas requiring attention.

The system is built as a full-stack web application using ASP.NET Core 8.0 for the frontend interface and Python FastAPI for the AI-powered backend. It utilizes Supabase PostgreSQL with pgvector extension for efficient vector-based similarity search, enabling semantic understanding of student data. The integration of HuggingFace's DeepSeek language model allows the chatbot to engage in natural conversations and provide meaningful analysis of student performance metrics.

ILearn offers a dual functionality approach: traditional CRUD (Create, Read, Update, Delete) operations for student data management, and an intelligent conversational interface for analyzing and querying student information. This combination makes it a comprehensive tool for modern educational institutions seeking to leverage AI technology for better student outcomes.

1.2 Problem Statement

Educational institutions face several significant challenges in managing and analyzing student performance data effectively:

1. Manual Data Analysis: Teachers spend considerable time manually reviewing student records, calculating averages, and identifying patterns in performance data. This process is time-consuming and prone to human error, reducing the time available for actual teaching and student interaction.

2. Lack of Real-Time Insights: Traditional reporting systems generate static reports that quickly become outdated. Educators need immediate answers to questions like "Which students are struggling in Mathematics?" or "How is Ahmed's attendance compared to last month?" but current systems cannot provide instant, conversational responses.

3. Data Accessibility Issues: Student performance data is often stored in spreadsheets or disconnected database systems that require technical knowledge to query effectively. Non-technical educators struggle to extract meaningful insights without IT support, creating bottlenecks in decision-making.

4. Limited Contextual Understanding: Existing systems can show raw numbers and statistics but lack the ability to provide contextual analysis. For example, understanding that a student's declining performance might be related to poor attendance requires connecting multiple data points, which traditional systems handle poorly.

5. Inefficient Communication: When teachers need to discuss student performance with parents or administrators, gathering comprehensive information from multiple sources is cumbersome. There is no unified interface that provides a complete picture of student performance on demand.

6. Scalability Concerns: As student populations grow, managing individual student data and providing personalized attention becomes increasingly difficult. Manual methods do not scale well, leading to some students being overlooked.

These challenges result in delayed interventions, missed opportunities for student support, and inefficient use of educators' time. There is a clear need for an intelligent system that can provide instant, natural language access to student performance data while maintaining comprehensive data management capabilities.

1.3 Objectives

The primary objectives of the ILearn project are:

1. Develop an Intelligent Chatbot Interface: Create an AI-powered conversational interface that allows educators to ask questions about student performance in natural language and receive contextually relevant, accurate responses. The chatbot should understand various query formats and provide meaningful insights rather than just raw data.

2. Implement Comprehensive Student Data Management: Build a robust CRUD system that enables educators to efficiently manage student records, including personal information, subject-wise performance, attendance records, and assignment completion rates. The system should provide an intuitive interface for data entry and updates.

3. Integrate RAG Technology: Implement Retrieval-Augmented Generation technology to enable semantic search and context-aware responses. The system should convert student data into vector embeddings and use similarity search to retrieve relevant information before generating responses.

4. Ensure Real-Time Performance Analysis: Provide instant analysis of student performance metrics, including identification of strengths, weaknesses, and areas requiring improvement. The system should automatically categorize performance levels and provide actionable recommendations.

5. Create a Scalable Architecture: Design a system architecture that can handle growing amounts of student data without performance degradation. The use of cloud-based services and efficient database indexing should ensure scalability.

6. Maintain Data Security and Privacy: Implement proper security measures to protect sensitive student information, including environment-based configuration management, input validation, and secure API communication.

7. Provide Multi-Technology Integration: Successfully integrate multiple technologies including .NET for frontend, Python for AI processing, PostgreSQL for data storage, and cloud-based AI models to create a cohesive system.

8. Enable Context-Aware Conversations: Allow the chatbot to maintain conversation history and provide follow-up responses that build on previous interactions, creating a more natural and efficient user experience.

1.4 Project Scope

In Scope:

Functional Requirements:

1. **Student Management Module:**
   - Create, read, update, and delete student records
   - Store student information including ID, name, semester, subjects with marks, attendance percentage, and assignment completion rates
   - Display student data in list and detailed views
   - Validate input data to ensure accuracy
2. **AI Chatbot Module:**

    - Natural language question-answering interface
    - Student-specific performance queries
    - Comparative analysis between subjects
    - Attendance and assignment tracking queries
    - Context-aware follow-up questions
    - Conversation history management
    - Quick suggestion buttons for common queries
3. **RAG Pipeline:**
    - Automatic generation of vector embeddings for student data
    - Semantic similarity search using cosine distance
    - Context retrieval for AI responses
    - Integration with HuggingFace models for embeddings and language generation
4. **Database Management:**
    - PostgreSQL database with pgvector extension
    - Separate tables for business data and AI embeddings
    - Efficient indexing for vector search operations
    - Cloud-hosted database on Supabase
5. **User Interface:**
    - Responsive web interface built with ASP.NET Core MVC
    - Clean and intuitive design
    - Real-time status indicators
    - Progress bars for performance visualization

Technical Requirements:

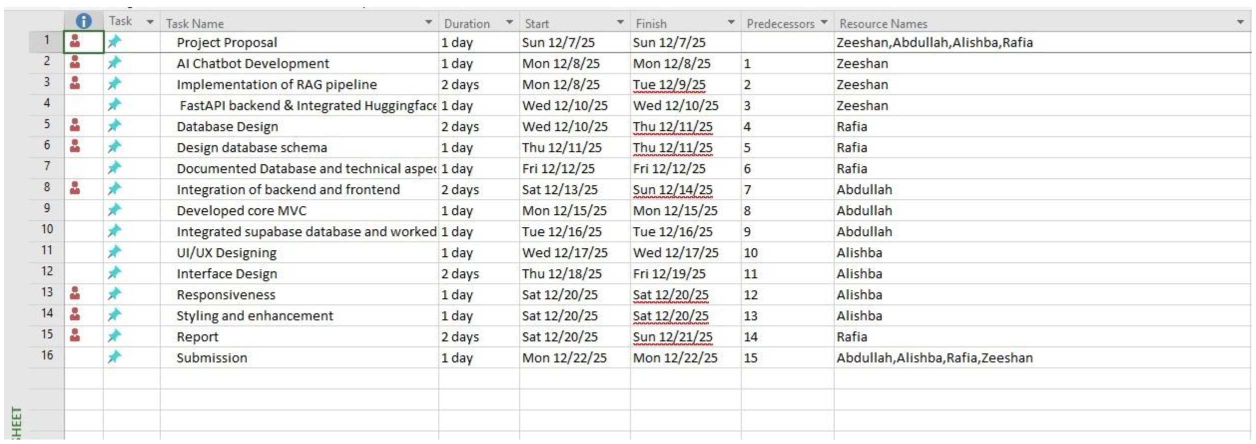
1. Frontend: ASP.NET Core 8.0 MVC framework
2. Backend API: Python FastAPI for AI processing
3. Database: Supabase PostgreSQL with pgvector
4. AI Models: HuggingFace DeepSeek-V3.2 and BAAI/bge-small-en-v1.5
5. Deployment: Local development environment with cloud database
6. Security: Environment variables for API keys and secrets

Limitations:

1. The system is currently configured for demonstration with 4 sample students
2. AI response quality depends on the accuracy of input data
3. Internet connection required for AI model access
4. Response time for chatbot varies from 2-5 seconds depending on query complexity
5. Vector search accuracy depends on the quality of embeddings

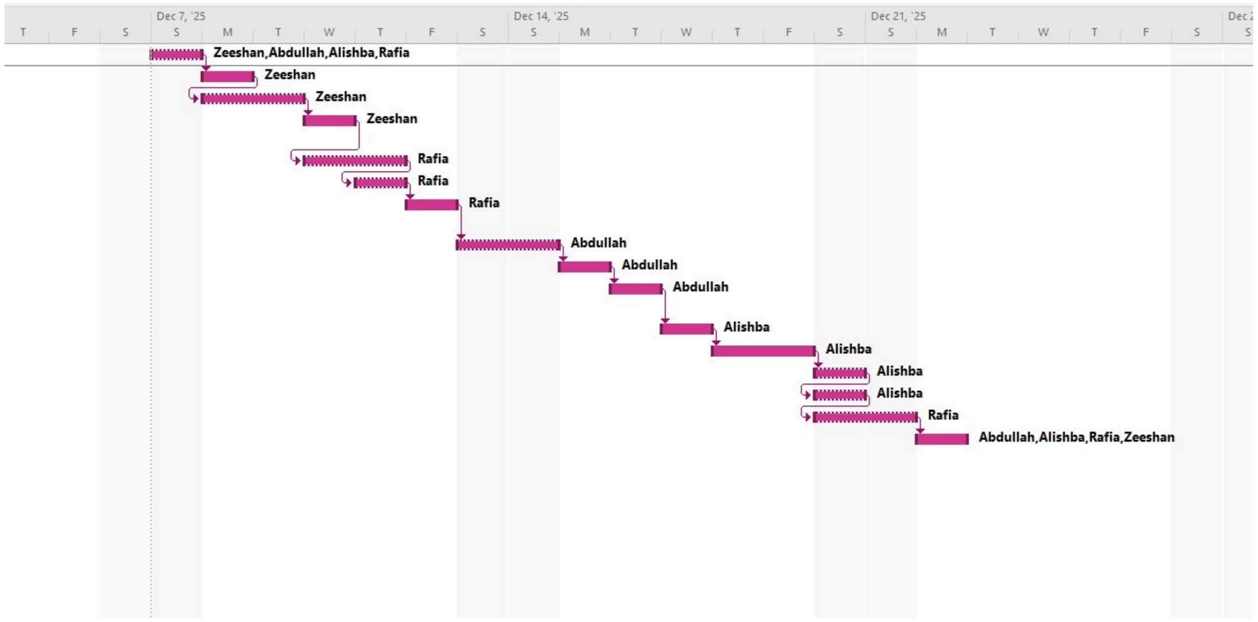
1.5 Implementation Timeline

Task Sheet

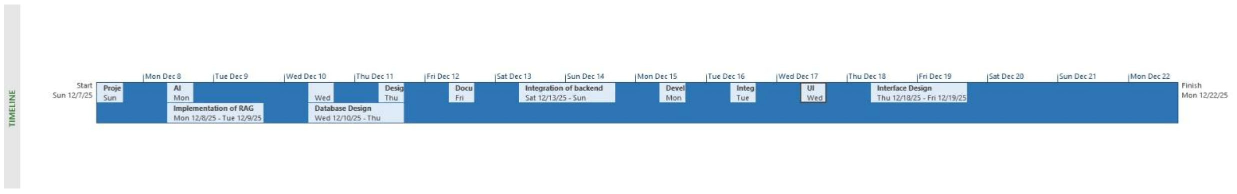


		Task	Task Name	Duration	Start	Finish	Predecessors	Resource Names
1			Project Proposal	1 day	Sun 12/7/25	Sun 12/7/25		Zeeshan,Abdullah,Alishba,Rafia
2			AI Chatbot Development	1 day	Mon 12/8/25	Mon 12/8/25	1	Zeeshan
3			Implementation of RAG pipeline	2 days	Mon 12/8/25	Tue 12/9/25	2	Zeeshan
4			FastAPI backend & Integrated Huggingface	1 day	Wed 12/10/25	Wed 12/10/25	3	Zeeshan
5			Database Design	2 days	Wed 12/10/25	Thu 12/11/25	4	Rafia
6			Design database schema	1 day	Thu 12/11/25	Thu 12/11/25	5	Rafia
7			Documented Database and technical aspec	1 day	Fri 12/12/25	Fri 12/12/25	6	Rafia
8			Integration of backend and frontend	2 days	Sat 12/13/25	Sun 12/14/25	7	Abdullah
9			Developed core MVC	1 day	Mon 12/15/25	Mon 12/15/25	8	Abdullah
10			Integrated supabase database and worked	1 day	Tue 12/16/25	Tue 12/16/25	9	Abdullah
11			UI/UX Designing	1 day	Wed 12/17/25	Wed 12/17/25	10	Alishba
12			Interface Design	2 days	Thu 12/18/25	Fri 12/19/25	11	Alishba
13			Responsiveness	1 day	Sat 12/20/25	Sat 12/20/25	12	Alishba
14			Styling and enhancement	1 day	Sat 12/20/25	Sat 12/20/25	13	Alishba
15			Report	2 days	Sat 12/20/25	Sun 12/21/25	14	Rafia
16			Submission	1 day	Mon 12/22/25	Mon 12/22/25	15	Abdullah,Alishba,Rafia,Zeeshan

Gantt Chart:



Timeline



CHAPTER 2: SYSTEM ANALYSIS AND DESIGN

2.1 Literature Review

The development of ILearn is based on modern web development practices using the ASP.NET Core framework and associated technologies. This review focuses on key elements of the .NET ecosystem and web application architecture that shaped the project's design.

2.1.1 ASP.NET Core MVC Framework and C# with .NET 8.0

ASP.NET Core 8.0 is Microsoft's cross-platform, high-performance framework for building web applications, representing a modular rewrite of the original ASP.NET. It employs the Model-View-Controller (MVC) pattern to enforce separation of concerns: Models handle data and business logic, Views manage the user interface using Razor syntax, and Controllers process requests, interact with models, and return responses.

Key built-in features include model binding (mapping request data to parameters), validation via data annotations, dependency injection, middleware pipelines, tag helpers, and asynchronous programming support for efficient concurrency.

C#, the primary language, is a strongly-typed, object-oriented language with features like garbage collection, async/await for non-blocking operations, LINQ for data querying, nullable reference types for null safety, and records for immutable objects. .NET 8.0, a long-term support release, offers optimized performance, cross-platform support, and cloud-native capabilities.

In ILearn, ASP.NET Core 8.0 and C# form the core, enabling maintainable, testable, and scalable code through professional practices such as encapsulation, exception handling, and standard naming conventions.

2.1.2 RESTful API Integration, HttpClient, and Database Access

RESTful APIs use HTTP methods (GET, POST, PUT, DELETE) for resource operations, typically with JSON data exchange. ASP.NET Core's HttpClient, managed via IHttpClientFactory and dependency injection, supports efficient requests with connection pooling and error handling.

ILearn's ChatService uses injected HttpClient to communicate with a Python FastAPI backend, managing JSON serialization/deserialization, timeouts, and graceful error handling (e.g., try-catch and logging).

For database access, while Entity Framework Core (EF Core) is a powerful ORM for object-relational mapping, ILearn employs direct API calls to Supabase via a dedicated SupabaseService layer. This service encapsulates CRUD operations, asynchronous methods, connection management, exception handling, and data validation, promoting separation of concerns, testability, and maintainability.

2.1.3 Razor Views, Frontend Development, and Model Validation

Razor is ASP.NET Core's view engine, blending C# with HTML for dynamic rendering, loops, conditionals, and reusable partial views. It provides compile-time checks, strong typing, and features like tag helpers, layouts for consistency, and Bootstrap integration for responsive design.

ILearn's UI relies entirely on Razor views, utilizing @model directives, HTML/tag helpers, ViewData/ViewBag, form handling with anti-forgery tokens, and client-side validation.

Model validation uses data annotations (e.g., [Required], [StringLength], [Range], [RegularExpression]) for rules on properties. Validation runs automatically during binding, with errors stored in ModelState and displayed via tag helpers. ILearn applies this to student forms (e.g., required names/IDs, range checks for marks), including custom and client-side validation, preserving user input on failures.

2.1.4 *Dependency Injection, Configuration, and Security Best Practices*

ASP.NET Core's built-in dependency injection (DI) container supports Singleton, Scoped, and Transient lifetimes, promoting loose coupling and testability. In ILearn, services (e.g., SupabaseService, ChatService) are registered in Program.cs and constructor-injected into controllers, avoiding direct instantiation.

Configuration is flexible, sourcing from appsettings.json, environment-specific files, variables, and secrets. ILearn uses this for Supabase credentials and endpoints, with .gitignore protection against committing sensitive data, and supports typed binding to classes.

Security follows secure-by-default principles: HTTPS enforcement, anti-forgery tokens against CSRF, input/output validation against injection/XSS, CORS control, and isolated data access. ILearn implements defense-in-depth with validation, secure config, error handling without leakage, and no hardcoded credentials.

2.1.5 Asynchronous Programming and Performance Optimization

Asynchronous programming with async/await allows non-blocking I/O, freeing threads for better scalability. ILearn applies this extensively in controllers, services, HttpClient calls, and database operations.

Additional optimizations include appropriate DI lifetimes, System.Text.Json serialization, connection pooling, and reduced allocations, ensuring efficient handling of concurrent users.

2.2 System Architecture

2.2.1 Technology Stack

The ILearn system is built using a modern multi-tier architecture that combines the strengths of different technologies. The following table summarizes the technology stack:

Component	Technology	Purpose
Frontend	ASP.NET Core 8.0 MVC	User interface and business logic
Backend API	Python FastAPI	AI/ML processing and RAG pipeline
Database	Supabase PostgreSQL + pgvector	Data storage with vector search capabilities
AI Model	HuggingFace DeepSeek-V3.2	Large Language Model for conversational chat
Embeddings	BAAI/bge-small-en-v1.5	Text-to-vector conversion (384 dimensions)
Deployment	Local Development	Localhost testing environment

2.2.2 Architecture Overview

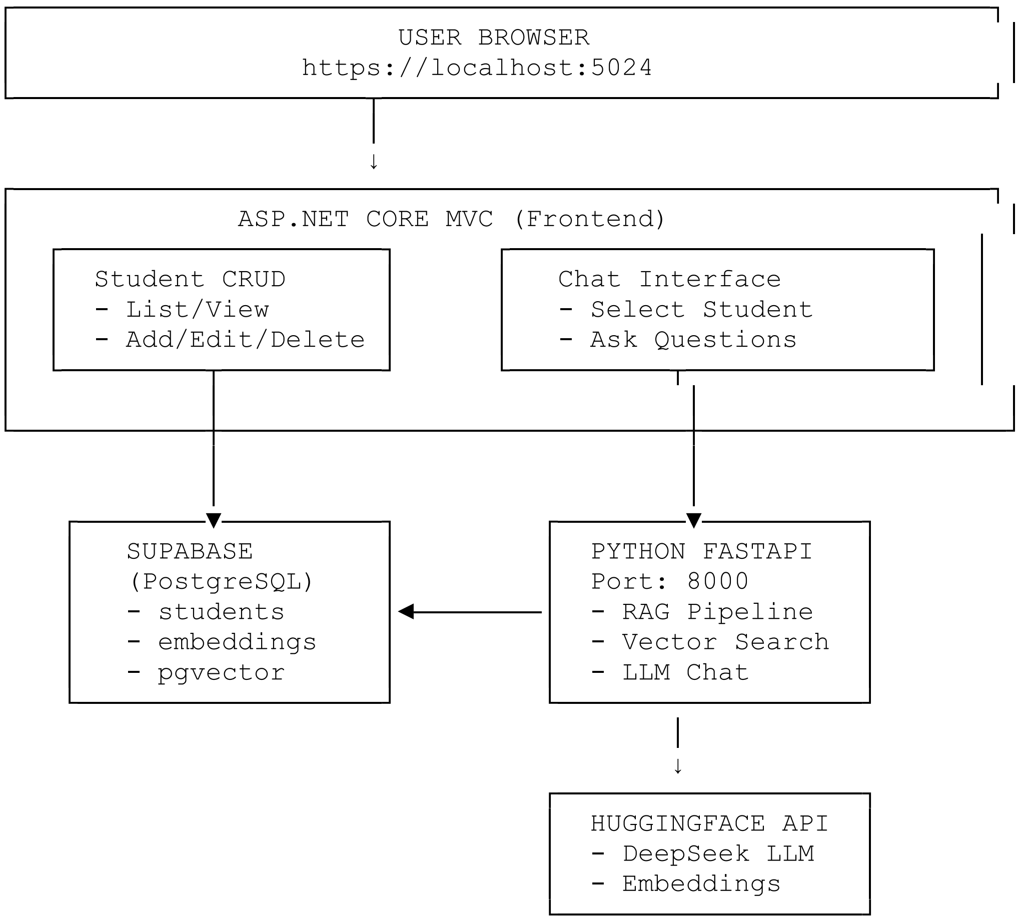
The system follows a three-tier architecture with clear separation of concerns:

Presentation Tier (Frontend): The presentation layer is built using ASP.NET Core 8.0 MVC, running on `https://localhost:5024`. This tier handles all user interactions including displaying student information, managing CRUD operations, and providing the chat interface. The frontend communicates with both the database (Supabase) for student data operations and the Python backend for AI-powered analysis.

Application Tier (Backend API): The application logic for AI processing is implemented using Python FastAPI, running on port 8000. This tier handles the RAG (Retrieval-Augmented Generation) pipeline, vector search operations, and communication with the HuggingFace API for AI model inference. FastAPI was chosen for its high performance, automatic API documentation, and excellent support for asynchronous operations.

Data Tier (Database): The data layer uses Supabase, a cloud-hosted PostgreSQL database with the pgvector extension. The database stores two main types of data: business data in the `students` table containing all student information, and AI data in the `student_embeddings` table containing vector representations for semantic search.

2.2.3 System Architecture Diagram



2.2.4 Project Structure

The project follows a modular structure with clear separation between frontend and backend components:

```
Project Root/
├── chatbot/                     # Python Backend
│   ├── api/
│   │   └── main.py             # FastAPI endpoints
│   ├── src/
│   │   ├── config.py           # Configuration
│   │   ├── embeddings.py       # HuggingFace embeddings
│   │   ├── llm_handler.py      # LLM integration
│   │   ├── rag_pipeline.py     # RAG logic
│   │   ├── supabase_vector_store.py  # Vector DB
│   │   ├── utils.py            # Utilities
│   │   └── models.py           # Data models
│   ├── .env                    # Secrets (not in Git)
│   ├── requirements.txt        # Python dependencies
│   └── create_index.py         # Generate embeddings
│
└── ILearn/                      # ASP.NET Frontend
    ├── Controllers/
    │   ├── HomeController.cs
    │   ├── StudentController.cs # CRUD operations
    │   └── ChatController.cs    # Chat interface
    ├── Models/
    │   ├── Student.cs          # Student model
    │   ├── ChatModels.cs       # Chat DTOs
    │   └── ErrorViewModel.cs
    ├── Services/
    │   ├── SupabaseService.cs   # Database service
    │   └── ChatService.cs      # Python API client
    ├── Views/
    │   ├── Home/
    │   ├── Student/            # CRUD views
    │   └── Chat/               # Chat UI
    ├── .env                    # Secrets (not in Git)
    ├── Program.cs              # App configuration
    └── appsettings.json        # Settings template
```

2.2.5 Database Schema

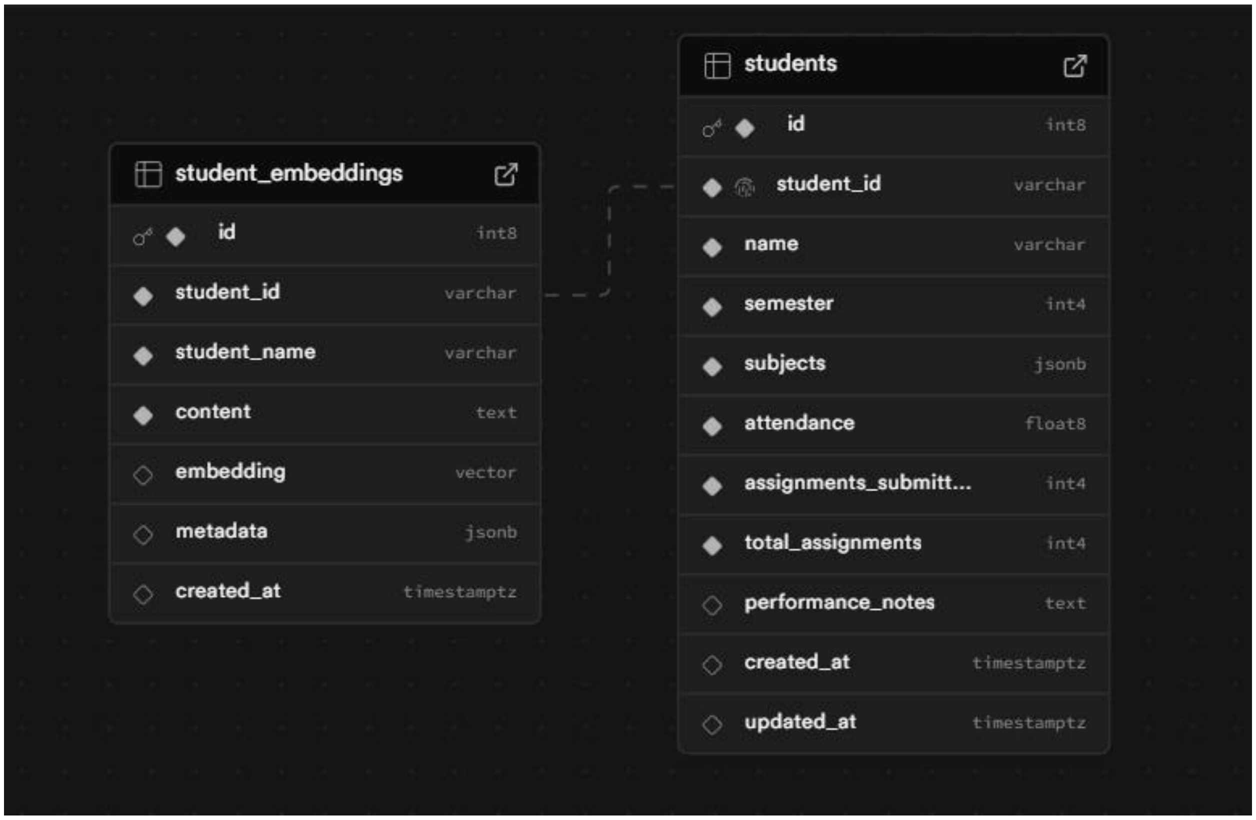


Table: students

```
- id (bigserial, PK)
- student_id (varchar, unique)
- name (varchar)
- semester (integer)
- subjects (jsonb)
- attendance (float)
- assignments_submitted (integer)
- total_assignments (integer)
- performance_notes (text)
- created_at (timestamp)
- updated_at (timestamp)
```

Table: student_embeddings

```
- id (bigserial, PK)
- student_id (varchar, FK → students)
- student_name (varchar)
- content (text)
- embedding (vector(384))
- metadata (jsonb)
- created_at (timestamp)
```

Function: match_student_embeddings

- Vector similarity search using cosine distance
- Returns top K most similar students

🔌 API ENDPOINTS

Python Backend (FastAPI) - Port 8000

Method	Endpoint	Purpose
GET	/health	Health check
GET	/students	List all students
POST	/chat	AI chat endpoint
POST	/reset-conversation	Reset chat

ASP.NET Frontend - Port 5024

Route	Purpose
/	Home page
/Student	Student list
/Student/Details/{id}	View student
/Student/Create	Add student
/Student/Edit/{id}	Edit student
/Student/Delete/{id}	Delete student
/Chat	AI chat interface

🎨 USER INTERFACE

1. Home Page

- Welcome message
- Navigation to Students and Chat

2. Student Management

- **List View:** Table with student ID, name, semester, attendance

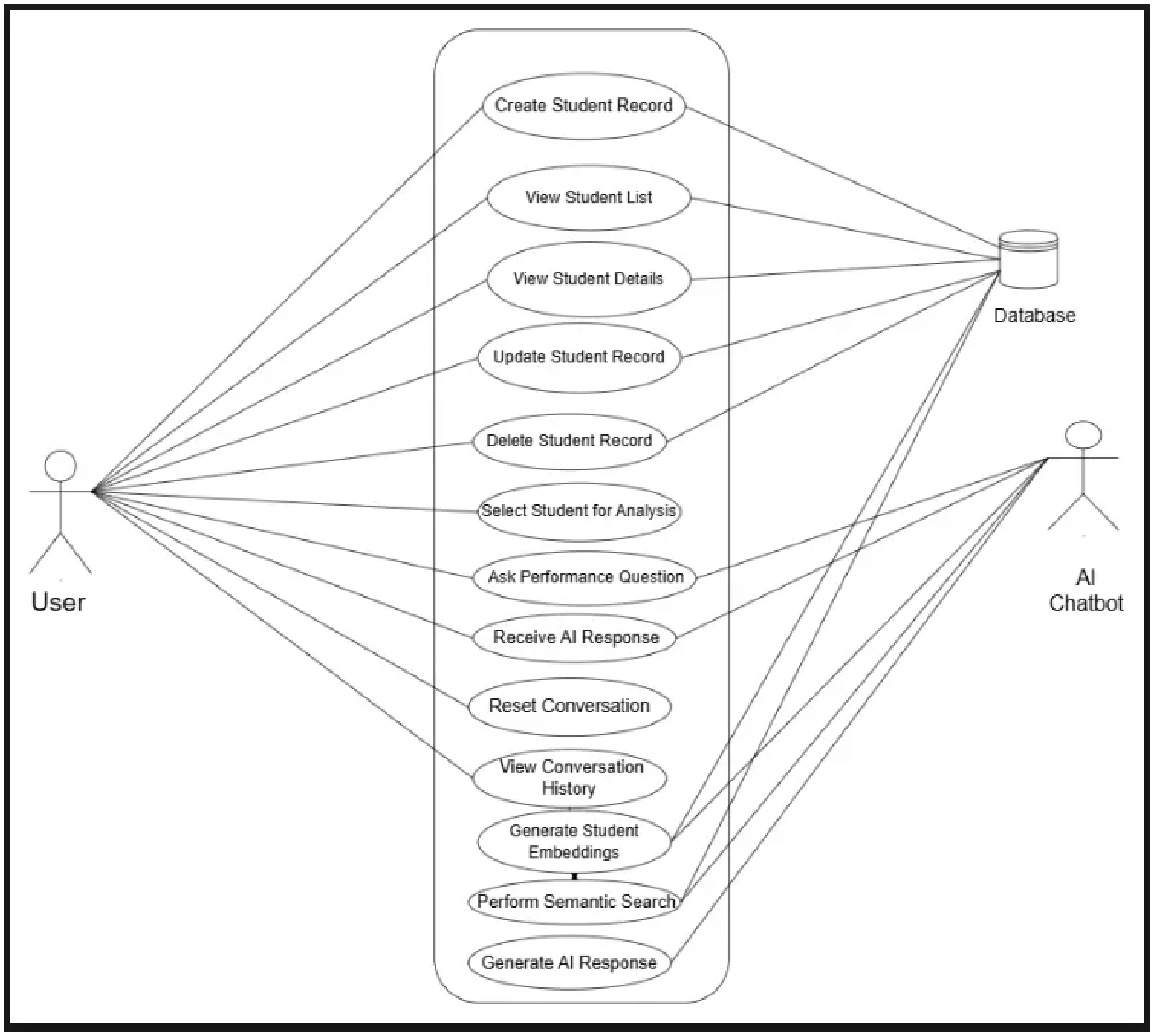
- **Details View:** Complete student profile with subjects and progress bars
- **Forms:** Create/Edit forms with validation

3. Chat Interface

- **Sidebar:** Student selector, conversation reset, quick tips
- **Chat Area:** Message history with user/AI bubbles
- **Input:** Text box with send button
- **Suggestions:** Smart follow-up question buttons
- **Status:** API connection indicator

2.3 System Diagrams

2.3.1 Use Case Diagram



Use Case Diagram for ILearn System

Actors:

1. **Educator (Primary Actor)** - The main user who interacts with the system
2. **AI Chatbot (Secondary Actor)** - External AI service that processes queries
3. **Supabase Database (Supporting Actor)** - Data storage system

Use Cases:

1. Student Management Module

- **UC-1: Create Student Record**
  - Actor: Educator
  - Description: Add new student with details (ID, name, semester, subjects, marks, attendance)
  - Precondition: Educator is authenticated
  - Postcondition: Student record saved to database
- **UC-2: View Student List**
  - Actor: Educator
  - Description: Display all students in table format
  - Includes: Filter and search functionality
- **UC-3: View Student Details**
  - Actor: Educator
  - Description: Display complete profile with performance visualization
  - Includes: Progress bars, subject-wise marks, attendance
- **UC-4: Update Student Record**
  - Actor: Educator
  - Description: Edit existing student information
  - Precondition: Student exists in system
- **UC-5: Delete Student Record**
  - Actor: Educator
  - Description: Remove student from database
  - Includes: Confirmation dialog

2. AI Chat Module

- **UC-6: Select Student for Analysis**
  - Actor: Educator
  - Description: Choose specific student for AI analysis
  - Precondition: At least one student exists
  - Postcondition: Student context loaded for chatbot
- **UC-7: Ask Performance Question**
  - Actor: Educator, AI Chatbot
  - Description: Submit natural language query about student
  - Includes: Quick suggestion buttons
  - Extends: Generate Vector Embeddings
- **UC-8: Receive AI Response**
  - Actor: Educator, AI Chatbot
  - Description: View AI-generated analysis and insights
  - Includes: Conversation history display
- **UC-9: Reset Conversation**
  - Actor: Educator
  - Description: Clear chat history and start fresh
  - Postcondition: Conversation context cleared
- **UC-10: View Conversation History**
  - Actor: Educator
  - Description: Review previous chat messages

3. AI Processing (Backend)

- **UC-11: Generate Student Embeddings**
  - Actor: AI chatbot
  - Description: Convert student data to vector embeddings
  - Trigger: New student created or updated
- **UC-12: Perform Semantic Search**
  - Actor: AI Chatbot, Supabase Database
  - Description: Find relevant student data using vector similarity
  - Includes: Cosine distance calculation
- **UC-13: Generate AI Response**
  - Actor: AI Chatbot
  - Description: Use RAG pipeline to create contextual answer
  - Includes: Context retrieval, LLM inference

2.4 Project Links

Description: "For more information and updates, please visit the following:

- **GitHub Repository:**
- https://github.com/abdullahKhanKhetran/Ilearn
  https://github.com/abdullahKhanKhetran/Ilearn-Backend
-
- - Access the complete source code, documentation, and project files
- **LinkedIn:**
- https://www.linkedin.com/posts/abdullah-khan-845607362_dotnet-postgresql-supabase-activity-7408985373776699392-IYQF?utm_source=share&utm_medium=member_android&rcm=ACoAAFolm_UBg788HaUhBCSe4HdKV17rVkX2ttg
- - Connect for professional updates and project discussions"

2.5 CONCLUSION

The ILearn - Student Performance Analysis System successfully demonstrates the effective integration of modern web technologies with artificial intelligence to address real-world challenges in educational data management. Through the collaborative efforts of our four-member team over a three-week development period, we have created a comprehensive solution that transforms how educators interact with student performance data.

The project achieved all its primary objectives by developing a fully functional web application that combines traditional CRUD operations with an intelligent AI-powered chatbot interface. The use of ASP.NET Core 8.0 MVC for the frontend provided a robust, scalable, and maintainable foundation for the user interface, while the integration with Python FastAPI for AI processing showcased the power of multi-technology architectures in modern application development.