An AI-Driven School Management System with Intelligent Student Assistance

by

Md. Al-Amin Sarker
ID: CSE2203027128

Anik Paul
ID: CSE2203027067

Eyanur Akther
ID: CSE2203027037

Sabukun Nahar Poly
ID: CSE2203027126

Supervised by
Md. Shamim Hossain

Submitted in partial fulfillment of the requirements for the degree of
Bachelor of Science in Computer Science and Engineering

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
SONARGAON UNIVERSITY (SU)
May 2026

APPROVAL

The project titled **"ClassMatrix: An AI-Driven School Management System with Intelligent Student Assistance"** submitted by Md. Al-Amin Sarker (CSE2203027128), Anik Paul (CSE2203027067), Eyanur Akther (CSE2203027037), and Sabikun Nahar Poly (CSE2203027126) to the Department of Computer Science and Engineering, Sonargaon University (SU), has been accepted as satisfactory for the partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science and Engineering and approved as to its style and content.

Board of Examiners

Md. Shamim Hossain
Lecturer
Department of Computer Science and Engineering
Sonargaon University (SU)

Supervisor

(Examiner Name and Signature)
Department of Computer Science and Engineering
Sonargaon University (SU)

Examiner 1

(Examiner Name and Signature)
Department of Computer Science and Engineering
Sonargaon University (SU)

Examiner 2

(Examiner Name and Signature)
Department of Computer Science and Engineering
Sonargaon University (SU)

Examiner 3

DECLARATION

We, hereby, declare that the work presented in this report is the outcome of the investigation performed by us under the supervision of **Md. Shamim Hossain, Lecturer,** Department of Computer Science and Engineering, Sonargaon University, Dhaka, Bangladesh. We reaffirm that no part of this project has been or is being submitted elsewhere for the award of any degree or diploma.

Countersigned

(Md. Shamim Hossain)
Supervisor

Signature

Md. Al-Amin Sarker
ID: CSE2203027128

Anik Paul
ID: CSE2203027067

Eyanur Akther
ID: CSE2203027037

Sabikur Nahar Poly
ID: CSE2203027126

ABSTRACT

School management software has become essential infrastructure for modern educational institutions, yet most existing platforms remain transactional in nature, offering little in the way of personalized academic support for students. At the same time, advances in Large Language Models have made conversational, context-aware tutoring practical, but such tools usually exist as standalone services disconnected from the institutional data that would make them most useful. This work bridges that gap. We present ClassMatrix, an AI-driven school management system that integrates a complete administrative platform with intelligent student assistance powered by OpenAI's GPT models.

The system is built using Laravel 13, Inertia.js v3, Vue 3, and Tailwind CSS v4. It supports five user roles Super Admin, Admin, Teacher, Student, and Parent through a role-based access control layer implemented with the Spatie permission package. Functionality is organized into eight delivery phases covering academic setup, daily operations, examinations, finance and human resources, library, transport, hostel, online learning, and an AI-enabled student panel. The student panel includes a personal-teacher chatbot that draws on the student's own academic record (enrolled subjects, recent exam scores, attendance) to generate contextually grounded responses, together with an automated study-routine generator. Voice input and output are implemented through browser-native Web Speech APIs, eliminating any per-request cost for speech processing.

The completed implementation comprises 190 routes, 64 Eloquent models, 61 Vue pages, 31 database migrations, and 16 backed enums. GPT-4o is used for routine generation due to its stronger structured-output capability, while the lighter GPT-4o-mini handles conversational tutoring at lower cost. Functional validation across all eight phases confirms that integrating institutional data with LLM prompts produces noticeably more relevant and personalized assistance than generic chat tools. The work demonstrates that a tightly integrated school management platform with AI capabilities is both technically feasible and pedagogically valuable, particularly for institutions seeking modern, low-cost alternatives to commercial enterprise solutions.

Keywords: School Management System; Artificial Intelligence; Large Language Models; OpenAI GPT; Laravel; Vue.js; Inertia.js; Personalized Learning; Web Speech API; Educational Technology.

ACKNOWLEDGEMENT

First and foremost, we express our deepest gratitude to the Almighty for granting us the strength, patience, and perseverance to complete this project within the scheduled time.

We are profoundly indebted to our supervisor, **Md. Shamim Hossain**, Lecturer, Department of Computer Science and Engineering, Sonargaon University, for his thoughtful guidance, constructive feedback, and unwavering support throughout the development of ClassMatrix. His insights at every stage from problem framing to architectural decisions have shaped this work in ways that go far beyond what these pages can capture.

We extend our sincere thanks to the Honourable Head of the Department, the faculty members, and the staff of the Department of Computer Science and Engineering at Sonargaon University. Their teaching across the four years of our undergraduate study laid the foundation for this project, and their continued encouragement made it possible to attempt something at this scale.

We also thank our peers, classmates, and the wider open-source community whose tools, libraries, and freely shared knowledge made the technical work tractable within an academic timeline.

Finally, our heartfelt gratitude goes to our parents and families for their patience, sacrifices, and endless love. None of what follows would exist without them.

LIST OF ABBREVIATIONS

AI	Artificial Intelligence
API	Application Programming Interface
CRUD	Create, Read, Update, Delete
CSS	Cascading Style Sheets
DB	Database
GPT	Generative Pre-trained Transformer
HR	Human Resources
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
JS	JavaScript
JSON	JavaScript Object Notation
JWT	JSON Web Token
LLM	Large Language Model
MVC	Model-View-Controller
MySQL	My Structured Query Language
NLP	Natural Language Processing
ORM	Object-Relational Mapping
RBAC	Role-Based Access Control
REST	Representational State Transfer

SMS	School Management System
SPA	Single Page Application
SQL	Structured Query Language
SSR	Server-Side Rendering
STT	Speech-to-Text
TTS	Text-to-Speech
UI	User Interface
UX	User Experience
VCS	Version Control System

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1
INTRODUCTION

1.1 Introduction

Educational institutions across the world are under growing pressure to manage their administrative operations, academic processes, and stakeholder communication in ways that are both efficient and transparent. Traditional record-keeping, fragmented spreadsheets, and disconnected software tools tend to create silos of information that slow decision-making and make it harder for teachers, parents, and students to stay aligned. As schools grow, the case for a single, well-designed digital management platform becomes harder to ignore [1].

The arrival of capable Large Language Models (LLMs), particularly the GPT family, has opened a parallel front. These models can act as patient, always-available tutors, generate study plans tailored to individual learners, and respond to academic questions in natural language at a level that was unthinkable only a few years ago [2]. The trouble is that most school management systems treat AI as a bolt-on rather than a first-class part of the architecture. The result is shallow integrations that rarely use the rich student data the system already stores [3].

This project presents ClassMatrix, an AI-driven school management system that places intelligent student assistance at the centre of the platform rather than at its edge. ClassMatrix is built on a modern stack Laravel 13 on the server, Vue 3 with Inertia.js v3 on the client, and Tailwind CSS v4 for styling and uses OpenAI's GPT models to power two distinct AI features: a personal-teacher chatbot that draws on the student's actual academic record, and an automated study-routine generator that produces structured weekly schedules. The chapter that follows lays out the motivation, the problems we set out to solve, the literature we drew on, and the objectives that guided the work.

1.2 Background and Motivation

The education sector has gone through a period of rapid digital change, much of it accelerated by the disruption caused by the COVID-19 pandemic [4]. School management systems (SMS) have evolved from simple databases for marks and attendance into broad platforms covering examinations, finance, transport, hostels, and online learning. Despite this growth, several gaps remain visible across the offerings we surveyed.

First, popular open-source systems such as Fedena, OpenSIS, and SchoolTool provide a solid baseline of administrative features but offer little in the way of AI-powered personalization [5]. Students get the same dashboard regardless of how they are performing or what subjects they struggle with. Second, the communication channels between teachers and students outside class hours rarely go beyond plain messaging or email, with no intelligent layer to triage questions or offer first-line academic help. Third, study planning remains a manual exercise, even though it is exactly the kind of task where context-aware automation could make a meaningful difference [6].

Our motivation is to address these three gaps simultaneously rather than separately. By embedding an AI personal teacher and a routine generator inside the school management platform using the same student data already captured by attendance, exam, and enrolment modules students gain round-the-clock support without having to leave the system they already use every day. For schools, the benefit is that AI features ride on top of existing infrastructure investment instead of demanding a parallel platform.

1.3 Problem Statement

Existing school management systems, taken together, exhibit five recurring limitations that motivated the design of ClassMatrix:

1. Fragmentation: Schools often operate several disconnected tools — one for attendance, another for grades, a third for communication. The resulting data silos create inconsistencies and make holistic reporting difficult [7].
2. Lack of Intelligence: Most current systems are purely transactional. They record and display data, but they do not analyze it or surface actionable insights for teachers and learners [8].
3. Limited Personalization: All students see the same interface and receive the same support, regardless of academic profile, learning pace, or recent performance.
4. Poor Stakeholder Engagement: Parents in particular often lack timely visibility into their children's academic progress, attendance, and school activities [9].
5. Absent AI Integration: Although AI tutoring tools exist as standalone services, they are rarely embedded inside school management platforms in a way that uses institutional data as context [10]. ClassMatrix is an attempt to address all five issues within a single, coherent platform.

1.4 AI in Education

Artificial Intelligence has become a transformative force in education, with capabilities ranging from automated assessment to fully personalized tutoring. Modern Large Language Models such as GPT-4 demonstrate a striking ability to grasp context, generate fluent and pedagogically sound explanations, and adapt their tone to the learner's level [11].

AI-powered educational tools generally fall into a few categories: intelligent tutoring systems that deliver personalized instruction; automated assessment tools that mark and give feedback on student work; adaptive learning platforms that adjust content difficulty in response to performance; and conversational assistants that field student questions in natural language [12]. ClassMatrix focuses on the last two — conversational tutoring and automated planning — because these benefit most directly from the kind of structured student data a school management system already keeps.

Voice interaction further widens accessibility. Browser-native Speech-to-Text (STT) and Text-to-Speech (TTS) APIs allow hands-free use without a paid voice service, making the tutor available to students who prefer spoken interaction or who have visual or motor accessibility needs. This approach removes the cost and latency overhead of commercial voice APIs while still delivering a natural conversational experience [13].

1.5 Literature Review

1.5.1 Existing School Management Systems

Several school management platforms have shaped the current landscape. Fedena, an open-source Ruby on Rails application, offers basic modules for attendance, examinations, and finance but lacks both modern frontend tooling and any AI capability [5]. OpenSIS provides comprehensive scheduling and gradebook features on a legacy PHP stack with no real-time layer. SchoolTool, a Zope-based platform once supported by Canonical, has effectively been discontinued a reminder that long-term maintainability is a non-trivial property of educational software [14].

On the commercial side, products such as PowerSchool and Blackbaud bring enterprise-grade feature sets but at price points that put them out of reach of smaller institutions. They are also closed-source, which limits how far an institution can adapt them to local needs. None of the platforms we examined offered AI-powered student assistance as a built-in feature [15].

1.5.2 AI-Powered Educational Tools

On the AI tutoring side, Khan Acadmy's Khanmigo uses GPT-4 to provide personalized tutoring across subjects [16]. Duolingo's adaptive engine relies on AI to adjust language exercises in real time. Carnegie Learning has built mathematics products around AI-driven instruction. Each of these is impressive in isolation, yet they all share a common limitation: they are standalone platforms, disconnected from any school management system, which means the AI tutor has no idea what the student studied last week, what they scored, or where their attendance has slipped [17].

Recent academic work supports the case for tighter integration. Samal and Bajaj (2023) showed that generative AI can produce contextually relevant educational material when supplied with appropriate learner context [3]. Chen et al. (2024) demonstrated that LLM-based systems can sustain coherent, useful pedagogical conversations across long exchanges when system prompts are carefully designed [2].

1.5.3 Role-Based Access Control in Web Applications

Role-Based Access Control (RBAC) is a well-established mechanism for managing permissions in multi-stakeholder systems. In the Laravel ecosystem, the Spatie Laravel Permission package has become the de facto standard, providing a clean API for roles, permissions, and middleware-based enforcement [18]. The broader literature confirms that well-designed RBAC reduces unauthorized access incidents significantly in educational platforms compared to ad-hoc, role-encoded-as-flag schemes [19].

ClassMatrix implements a five-role hierarchy Super Admin, Admin, Teacher, Student, and Parent backed by more than 40 granular permissions. The intention is that each user sees only what is relevant to their role and can act only on what they are authorized to change.

1.5.4 Modern Web Application Frameworks

The web framework landscape has converged on a hybrid model that combines server-side rendering with rich client-side interactivity. Inertia.js, introduced by Jonathan Reinink, bridges traditional server frameworks (Laravel, Rails) with modern client frameworks (Vue, React) without forcing teams to maintain a separate API layer [20]. This "modern monolith" pattern keeps the productivity benefits of a tightly coupled stack while delivering an SPA-like experience to the user.

Laravel 13, the latest iteration of the PHP framework, ships with mature ORM support (Eloquent), authentication scaffolding (Fortify and Jetstream), queue management, and event broadcasting. Combined with Vue 3's Composition API and the utility-first approach of Tailwind CSS v4, this stack supports rapid construction of complex, interactive applications without sacrificing maintainability [21].

1.5.5 Research Gap

Synthesizing the literature, a clear gap emerges. Comprehensive school management systems exist. AI-powered educational tools exist. But the intersection — a single platform that integrates the two so that AI features are grounded in the institution's own student data — is sparsely populated. ClassMatrix is a direct response to this gap. By embedding AI assistance inside the school management workflow and feeding it the same data the institution already stores, the system aims to deliver a more personalized tutoring experience than any standalone AI product can match.

1.6 Objectives

The principal objectives of this project are:

1. To design and develop a comprehensive school management system covering all essential operational modules including attendance, examinations, finance, library, transport, and hostel management.
2. To implement a secure, role-based multi-panel architecture that serves administrators, teachers, students, and parents with tailored interfaces and appropriately scoped permissions.
3. AI-powered features a personal teacher chatbot and an automated study routine generator that draw on student academic data for genuine personalization rather than generic responses.
4. To enable voice interaction (Speech-to-Text and Text-to-Speech) for the AI chatbot using zero-cost browser-native APIs, ensuring accessibility without ongoing voice processing fees.
5. To deliver an online learning module supporting lessons, assignments, and quizzes that complements the AI assistance features.
6. To build a modern, responsive user interface with Vue 3 and Tailwind CSS v4 that prioritizes clarity, responsiveness, and a smooth user experience.

1.7 Scope and Limitations

Scope. ClassMatrix covers the full lifecycle of school operations, student enrolment, daily attendance, timetable management, examinations and grading, fee collection and invoicing, payroll management, library operations, transport and hostel management, online learning, and AI-powered student assistance. The system supports five user roles, each with its own dashboard, and exposes 190 routes across all modules.

Limitations. The work has the following bounded scope:

1. Single-school deployment: the present implementation supports a single institution. Multi-tenancy, which would allow SaaS-style deployment across many schools, is identified as future work.
2. External API dependence: the AI features depend on OpenAI API availability and are subject to free-tier rate limiting; an institution deploying ClassMatrix in production would need its own paid API key.
3. File handling: student photo uploads and assignment attachments use placeholder paths. Full media handling requires integration with a dedicated media library, deferred to future iterations.
4. Real-time messaging: the current build uses synchronous request–response communication. WebSocket-based push notifications are listed under future work.
5. Teacher panel coverage: while the teacher dashboard exists, certain teacher-side workflows (entering attendance and marks directly from the teacher view) are partially implemented.

CHAPTER 2
METHODOLOGY

2.1 Introduction

This chapter sets out the methodologies that guided the design, development, and validation of ClassMatrix. The system was built using an Agile, incremental approach with iterative cycles referred to throughout the project as "phases" each delivering a working slice of functionality. The chapter covers the development methodology, the architectural reasoning, the database modelling strategy, the AI integration approach, the frontend development style, and the testing and validation strategy.

2.2 System Development Methodology

ClassMatrix was developed using an incremental development model. The work was divided into eight phases, each building on the last and delivering a deployable milestone of its own. The eight phases were as follows:

Phase 1. Foundation core authentication, role and permission management, and academic structure setup (academic years, terms, departments, class levels, sections, subjects).

Phase 2. People & Enrolment student records, guardian and parent linking, staff records, and student enrolment workflows tied to academic years.

Phase 3. Daily Operations attendance, timetable, notices, calendar of events, and holiday management.

Phase 4. Examinations exam types, exam scheduling, marks entry, configurable grade schemes, and result computation.

Phase 5. Finance & HR fee categories and structures, bulk invoice generation, payment recording with locking, salary structures, payslips, and leave management.

Phase 6. Auxiliary Modules library, transport, hostel, and inventory management.

Phase 7. Online learning lessons with multimedia, assignments and submissions, quizzes with multiple question types, and result tracking.

Phase 8. AI & Student Panel the AI personal-teacher chatbot, study routine generator, and integrated student dashboard.

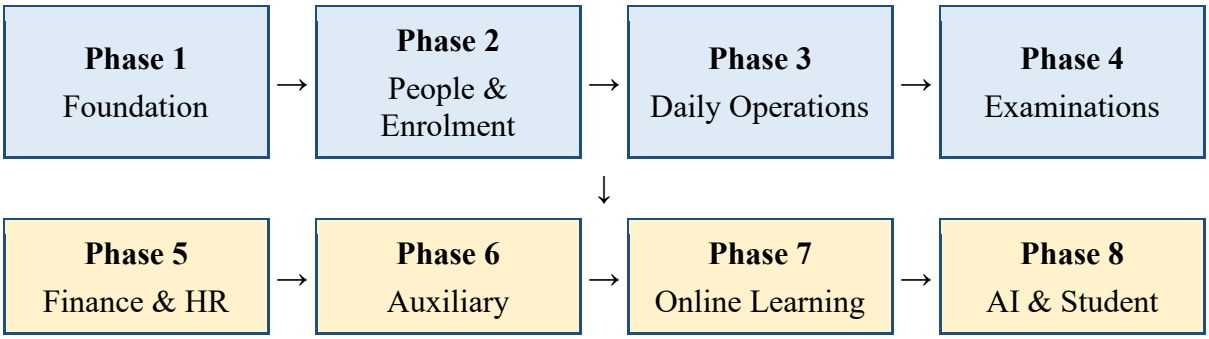

Fig. 2.1: Incremental Development Methodology — 8 Delivery Phases

Each phase followed a compressed waterfall: requirements analysis, design, implementation, and integration testing, before the next phase began. This structure ensured that each module was reasonably stable before later modules were built on top of it [22]. Where unforeseen requirements appeared during a phase, they were captured and either incorporated within that phase or scheduled for a later iteration, in keeping with Agile principles.

2.3 System Architecture Design

The system architecture follows the Model–View–Controller (MVC) pattern as expressed by Laravel, augmented by Inertia.js to bridge server logic and the Vue-based client. Four guiding principles shaped the architectural decisions:

1. Separation of concerns: backend logic in PHP/Laravel is kept strictly distinct from frontend presentation in Vue 3 and Tailwind CSS, with Inertia.js serving as the protocol bridge between them [20].
2. Domain-driven organisation: controllers, models, and views are grouped by functional domain (Admin, Student, Teacher, Parent) rather than by technical layer, making feature-level work easier to scope.
3. Service-layer extraction: complex business logic — particularly anything involving the AI provider — is pulled out of controllers into dedicated service classes (RoutineService, ChatService) so that controllers stay thin and testable.

4. Database integrity for money-handling paths: financial operations use database transactions with pessimistic locking (lockForUpdate ()) to prevent race conditions in concurrent payment scenarios [21].

2.4 Database Design Methodology

The database schema was designed following normalization principles to Third Normal Form (3NF) to limit redundancy while keeping queries efficient. The design process moved through four steps:

1. Entity identification: identifying core entities — users, students, teachers, classes, subjects, exams, fees, and so on — and their relationships from the requirements analysis.
2. Relationship mapping: defining one-to-many, many-to-many, and polymorphic relationships using Laravel's Eloquent conventions, including pivot tables for many-to-many associations and morph types where polymorphism was warranted.
3. Soft deletes: implementing soft delete behavior on critical entities (users, students, staff, and invoices) to preserve audit trails and allow recovery of accidentally removed records.
4. Enum-based status fields: using PHP 8.1 backed enums (16 in total) for type-safe status fields across attendance, leave, payment, exam state, and similar domains.

The completed schema comprises 31 migrations creating tables across nine functional categories: Auth & Core, Academic, People & Management, Daily Operations, Examinations, Finance, Auxiliary, Online Learning, and AI [23].

2.5 AI Integration Methodology

The AI integration follows a context-aware generation approach: relevant student data is collected at request time and injected into structured prompts so that the model's outputs are grounded in the learner's actual situation. The methodology has five elements:

1. Context collection: gathering student academic data enrolled subjects, recent exam scores, attendance percentage, and class level directly from the existing database tables.
2. Prompt engineering: designing structured prompts that combine fixed system instructions, dynamic student context, and a precise output-format specification (JSON for routines, free-form natural language for chat).
3. Model selection: using GPT-4o for routine generation, where structured-output reliability matters most, and GPT-4o-mini for chat, where lower latency and lower cost are more important than top-tier reasoning.
4. Error handling: implementing retry logic with exponential backoff for transient errors and rate limits, and graceful degradation with friendly user-facing messages when the API itself is unavailable.
5. Voice integration: leveraging the browser's Web Speech API Speech Recognition for STT and SpeechSynthesis for TTS to provide voice interaction at zero per-request cost [13].

2.6 Frontend Development Approach

The frontend was developed using Vue 3 with the Composition API, which delivers reactive, component-based UI architecture without the verbosity of the older Options API. Four practices shaped the work:

1. Layout-based routing: separate layout components for each user role (AppLayout for admin, StudentLayout, TeacherLayout, ParentLayout) keep role-specific navigation, branding, and sidebars cleanly isolated.
2. Component reusability: shared UI components (tables, forms, modals, cards) are built on top of Reka UI Vue port of Radix UI primitives which gives accessibility compliance out of the box.

3. Responsive design: Tailwind CSS v4 utility classes make mobile-first responsive layouts straightforward across all screen sizes.
4. Animation and UX polish: CSS-based animations (fade-in, count-up, hover lift) and selective use of glassmorphism on hero panels keep the interface modern without compromising performance [24].

2.7 Testing and Validation Methodology

Validation of the system relied on four complementary mechanisms:

1. Server-side validation: Laravel form-request validation on every endpoint that accepts user input, with custom error messages and shared rule traits to keep validation consistent across modules.
2. Database integrity: foreign-key constraints, unique indexes, and check constraints enforce consistency at the database layer, so that even a buggy controller cannot corrupt persistent state.
3. Role-based testing: every endpoint was exercised under each of the five roles to confirm that role-based middleware allows or denies access as intended.
4. Financial accuracy testing: invoice calculations, payment recording, and payroll generation were tested under simulated concurrency to verify that database transactions and pessimistic locking prevent double-payment and similar race conditions.

Table 2.1: Technologies Used in Implementation

Category	Technology	Version / Purpose
Backend Framework	Laravel	13 (PHP 8.3)
Frontend Framework	Vue.js	3.x (Composition API)
Bridge Layer	Inertia.js	v3 with Wayfinder
CSS Framework	Tailwind CSS	v4
Database	MySQL	8.x
Authentication	Laravel Fortify	Role-based with Spatie
AI Provider	OpenAI API	GPT-4o / GPT-4o-mini
UI Components	Reka UI	Radix Vue port
Build Tool	Vite	5.x
Local Server	Laravel Herd	macOS development
Version Control	Git	Distributed VCS
Voice I/O	Web Speech API	Browser-native STT / TTS

CHAPTER 3
SYSTEM ARCHITECTURE AND DESIGN

3.1 Introduction

This chapter presents the detailed architectural design of ClassMatrix, covering the high-level system structure, the multi-role authentication flow, the module organisation, the AI integration design, and the database schema. The architecture was designed with three properties in mind: maintainability, security, and extensibility.

3.2 High-Level Architecture

ClassMatrix follows a classical three-tier architecture, refined for the Laravel + Inertia + Vue stack:

1. Presentation tier: Vue 3 components rendered through Inertia.js, with role-specific layouts and interactive UI elements. The frontend communicates with the backend over Inertia's protocol — XHR requests that return JSON props consumed directly by Vue page components.
2. Application tier: Laravel controllers handle business logic, service classes manage cross-cutting operations such as AI calls and financial transactions, and middleware enforces access control. The application contains 38 controllers organised by role: Admin (19), Student (3), Teacher (1), Parent (1), with the remainder shared across authentication and infrastructure.
3. Data tier: a MySQL database, defined by 31 migration files, organised across nine functional domains and accessed through Eloquent ORM. Sixty-four model classes encapsulate relationships, scopes, and small pieces of business logic that belong with the data.

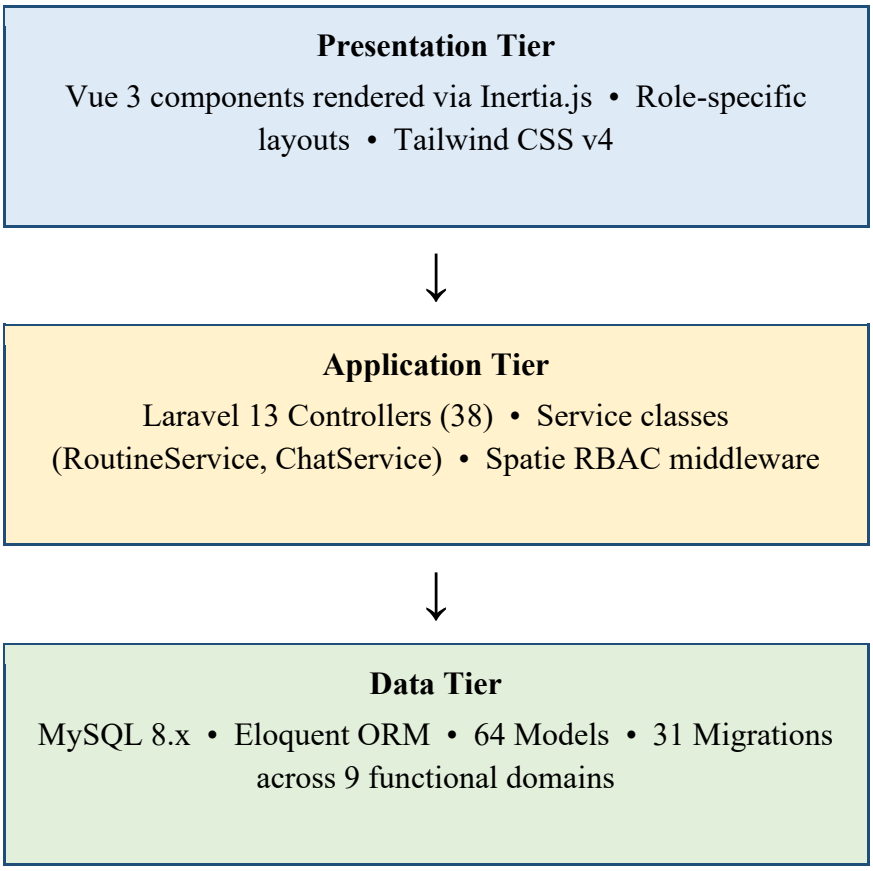

Fig. 3.1: High-Level Three-Tier System Architecture

3.3 Multi-Role Authentication System

The authentication system implements a hierarchical role model on top of the Spatie Laravel Permission package:

- Super Admin full system access, including the ability to create new admin accounts.
- Admin operational control of the school: students, staff, attendance, examinations, finance, and auxiliary modules.
- Teacher section- and subject-scoped access for entering attendance, marks, and online-learning content.
- Student access to personal academic information, AI features, and online learning materials.
- Parent read-only access to their linked children's attendance, results, and notices.

Table 3.1: Role-Based Access Control Permission Matrix

Module	Super Admin	Admin	Teacher	Student	Parent
User Management	Full	Limited	—	—	—
Academic Setup	Full	Full	View	View	—
Attendance	Full	Full	Mark	Own	Child
Examinations	Full	Full	Marks Entry	Own Results	Child Results
Finance	Full	Full	—	Own Invoices	Child Invoices
AI Features	—	—	—	Full	—
Online Learning	Full	Full	Manage	Access	—
Reports	Full	Full	Section	Own	Child

3.4 Module Architecture

ClassMatrix is organised into functional modules, each comprising one or more controllers, models, Vue pages, and route definitions:

- Academic Module manages academic years, terms, departments, class levels, sections, and subjects with hierarchical relationships and assignment of subjects to classes and teachers.
- Attendance Module daily student attendance with bulk-save support, monthly reports with attendance percentage calculation, and a five-state status taxonomy (Present, Absent, Late, Excused, Half Day).
- Timetable Module period definitions and an interactive grid-based timetable builder, with teacher conflict detection across sections.
- Examination Module exam types, exam creation with status tracking, per-subject scheduling, inline marks entry, and auto-grade calculation from configurable grade schemes.

- Finance Module fee categories and structures, bulk invoice generation, payment recording with pessimistic locking, salary structures, payslip generation, and leave management.
- Library Module book catalogue with categories, copy tracking, and an issue/return workflow that handles overdue detection and fine calculation.
- Transport Module vehicle management, routes with ordered stops and pickup/drop times, and student–route allocation.
- Hostel Module hostel and room management with occupancy tracking, warden assignment, and student room allocation.
- Online Learning Module lessons with video support, assignments with due dates, submission grading, and quizzes supporting MCQ, short-answer, and true/false question types.

3.5 AI Integration Architecture

The AI features are implemented through two dedicated service classes, each responsible for a clearly defined concern.

3.5.1 RoutineService (GPT-4o)

The RoutineService produces personalised weekly or monthly study routines. It works in four steps:

- Collects student preferences wake and sleep times, focus duration, peak focus period, subject priorities, and learning goals from the routine_preferences table.
- Gathers academic context such as enrolled subjects, recent exam scores, and attendance figures.
- Constructs a structured prompt that requests JSON-formatted routine output.
- Parses the GPT-4o response into time blocks with activities, daily goals, and motivational tips, which are then persisted and rendered.

3.5.2 ChatService (GPT-4o-mini)

The ChatService implements the AI Personal Teacher chatbot:

- Maintains conversation history in the ai_messages table for cross-session context continuity.
- Injects the student's context (class level, subjects, recent performance) into the system prompt at every turn.
- Handles rate-limit errors with exponential backoff and a small number of automatic retries.
- Returns user-friendly error messages when the API key is missing or the upstream service is unavailable.
- On the frontend, integrates browser Speech-to-Text for voice input and Text-to-Speech for voice output, with an auto-speak toggle for hands-free conversational mode.

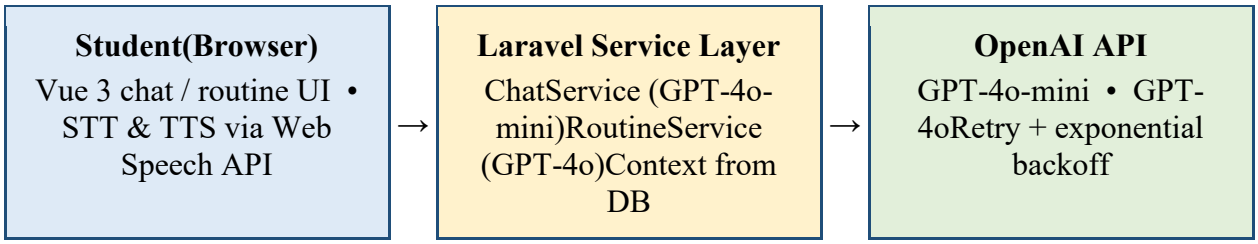

Fig. 3.3: AI Integration Architecture and Data Flow

3.6 Database Schema Design

The database comprises 31 migrations distributed across nine functional domains. Several schema-level decisions are worth noting: (a) soft deletes on user-, student-, staff-, and finance-related tables to preserve audit trails; (b) compound indexes on frequently filtered columns such as (academic_year_id, section_id) for attendance and (student_id, exam_id) for marks; (c) consistent use of foreign keys with cascading or restricting behaviour appropriate to each relationship; and (d) backed enum casts on every status field, enforced both in PHP and at the database level where supported.

Table 3.2: Database Tables by Functional Category

Category	Tables	Count
Auth & Core	users, password_reset_tokens, sessions, cache, jobs, permissions	6
Academic	academic_years, terms, departments, class_levels, sections, subjects, class_subjects, teacher_subjects	8
People	students, student_enrollments, guardians, student_guardian, staff, parent_student	6
Operations	periods, timetables, student_attendances, staff_attendances, notices, events, holidays	7
Examinations	exam_types, exams, exam_schedules, marks, grade_schemes, grade_ranges	6
Finance	fee_categories, fee_structures, discounts, invoices, invoice_items, payments, salary_structures, payslips, leaves, leave_balances	10
Auxiliary	books, book_categories, book_issues, vehicles, routes, route_stops, hostels, rooms, inventory_items, stock_movements	10
Online Learning	lessons, lesson_materials, assignments, assignment_submissions, quizzes, quiz_questions, quiz_options, quiz_attempts, quiz_answers	9
AI	routine_preferences, routines, ai_chats, ai_messages, ai_logs	5

CHAPTER 4
IMPLEMENTATION

4.1 Introduction

This chapter walks through the implementation of ClassMatrix in concrete terms. It covers the development environment, the backend implementation in Laravel 13, the frontend implementation with Vue 3 and Inertia.js, the AI module, and the role-specific panels. Where a particular technical decision had non-obvious consequences, those are discussed alongside the implementation itself rather than being deferred to the discussion chapter.

4.2 Development Environment Setup

The development environment was kept deliberately lightweight so that the same setup could be reproduced on any team member's machine. The configuration was as follows:

- Operating System: macOS, with Laravel Herd providing a unified PHP and MySQL development environment.
- PHP Version: 8.3, with the standard required extensions (OpenSSL, PDO, MySQL, Mbstring, Tokenizer, XML, Ctype, JSON).
- Node.js: v20+ for Vite-based build tooling and frontend compilation.
- Database: MySQL 8.x, accessed locally through Herd.
- IDE: Visual Studio Code with PHP Intelephense, Vue Language Features, and Tailwind CSS IntelliSense extensions.
- Version Control: Git, with a conventional-commits style adopted from the second phase onward to keep history navigable.

4.3 Backend Implementation

4.3.1 Authentication System

The authentication system extends Laravel Fortify with custom actions to fit the multi-role design:

- CreateNewUser action: implements a two-step registration flow. New users first select their role (Student or Parent only Admin and Teacher accounts are created internally by administrators). Parent registration requires a valid child admission number, which auto-links the parent to the student record.
- LoginResponse: a custom login response class redirects users to the dashboard route appropriate to their role (admin.dashboard, student.dashboard, teacher.dashboard, or parent.dashboard) instead of falling back to a single generic landing page.
- Middleware Stack: six custom middleware classes — AdminMiddleware, StudentMiddleware, TeacherMiddleware, ParentMiddleware, HandleInertiaRequests, and HandleAppearance — together enforce role-based route protection and shared layout state.

4.3.2 Controller Architecture

The system contains 38 controllers, organised by user role:

- Admin controllers (19): Dashboard Controller, Academic Year Controller, Attendance Controller, Time table Controller, Exam Controller, Marks Controller, Result Controller, Fee Controller, Invoice Controller, Payroll Controller, Library Controller, Transport Controller, Hostel Controller, Inventory Controller, Notice Controller, Event Controller, User Controller, Student Controller, and Staff Controller.
- Student controllers (3): Dashboard Controller, Routine Controller, and Chat Controller.
- Teacher controllers (1): Dashboard Controller, with the section-specific operations exposed through shared admin endpoints scored by middleware.
- Parent controllers (1): Parent Controller, which handles dashboard, children, attendance, results, and notice views in a single, cohesive class.

4.3.3 Financial Module with Transaction Safety

The finance module applies several safety measures specific to monetary operations, where correctness and concurrency matter more than elsewhere in the system:

- Pessimistic locking: invoice payment recording wraps the read-modify-write cycle inside DB::transaction () with lock For Update (), preventing the "double-payment" race that can otherwise arise when two cashiers process the same invoice simultaneously.
- Bulk generation: invoice creation is offered both per-student and at section level, with fee structures resolved per student so that discounts and per-student adjustments are reflected accurately in the issued invoice items.
- Salary calculation: pay slips are computed from designation-based salary structures using the formula basic + HRA + DA + TA − PF − Tax = net salary, with all components stored in the salary_structures table for audit purposes.

4.3.4 Enum-Based Type Safety

ClassMatrix uses 16 PHP 8.1 backed enums to keep status fields type-safe across the codebase. Examples include:

Table 4.1: PHP Backed Enums Used in the System

Enum	Values	Domain
UserType	SuperAdmin, Admin, Teacher, Student, Parent	Authentication
AttendanceStatus	Present, Absent, Late, Excused, HalfDay	Attendance
PaymentStatus	Pending, Partial, Paid, Overdue	Finance
LeaveStatus	Pending, Approved, Rejected	HR
ExamStatus	Scheduled, Ongoing, Completed, Cancelled	Examinations
QuestionType	MCQ, Short, TrueFalse	Online Learning
BookIssueStatus	Issued, Returned, Overdue, Lost	Library
GenderType	Male, Female, Other	User Profile

4.4 Frontend Implementation

4.4.1 Vue 3 Composition API

All 61 Vue pages use the Composition API with the script setup syntax. The benefits in practice are straightforward:

- Reactive state management with ref () and computed () rather than scattered Options API hooks.
- Compassable functions for shared logic useVoice (), useAppearance (), useCurrentUrl() that can be imported wherever needed without inheritance gymnastics.
- Props-based data flow from Inertia server responses to page components, with typed props in places where TypeScript would have been over-investment for an undergraduate timeline.

4.4.2 Layout System

The application uses role-specific layout components to keep navigation, branding, and shared chrome cleanly separated:

- AppLayout: the admin panel, with its deep navy sidebar, gradient stat cards, and quick-action sections.
- StudentLayout: a friendlier, lighter layout with a personalized welcome banner and prominent access to AI features.
- TeacherLayout: focused on section management, with shortcuts to attendance and marks entry.
- ParentLayout: a children-focused layout that surfaces each linked child as a card on entry.
- AuthSplitLayout: a two-column layout for authentication pages, pairing form fields with a gradient brand panel.

4.4.3 UI / UX Design System

The visual design system is intentionally restrained, with a small palette and a consistent typographic system carried across every panel:

- Color system: blue primary (#3B82F6), deep navy sidebar, cyan accents, and a full dark-mode palette derived from the same hues.
- Typography: the Inter font family for clear, modern, professional text rendering at every size used in the application.
- Animations: CSS-based fade-in-up with stagger, count-up numbers on dashboard stat cards, and subtle hover-lift effects throughout.
- Glass orphism: semi-transparent backdrop-blur effects on selective elements such as navigation bars and hero cards, used sparingly to avoid the "overdone glass" aesthetic.
- Landing page: a full redesign with floating glow orbs, an animated hero section, feature cards, role portal cards, and an AI showcase section featuring a mock chat interface to communicate the system's value at a glance.

4.5 AI Module Implementation

4.5.1 AI Personal Teacher (Chatbot)

The chatbot implementation has three layers backend service, persistence, and frontend interface:

- Backend (ChatService.php): communicates with the OpenAI API using the GPT-4o-mini model. The messages array is constructed from a system prompt that contains the student's context, the conversation history loaded from the ai_messages table, and the current user message. Retry logic with exponential backoff handles HTTP 429 (rate-limit) responses.
- Database: ai_chats stores conversation sessions, ai_messages stores individual messages with role (user / assistant) and content, and ai_logs records API usage and any errors encountered for later analysis.
- Frontend (Chat.vue): a full chat interface with message bubbles, loading indicators, a voice-input button driven by the browser Speech Recognition API, voice output with an auto-speak toggle (Speech Synthesis API), and an error banner with retry suggestions for any failed turn.

4.5.2 AI Routine Generator

The routine generator turns a small set of preferences and academic data into a coherent weekly study schedule:

- Preference collection: students set wake and sleep time, focus-block duration, peak focus period, subject priorities, and short-term learning goals on a dedicated preferences page.
- Generation (RoutineService.php): the service combines preferences with academic context, builds a detailed prompt requesting a weekly or monthly routine in JSON, and submits it to GPT-4o, which returns structured time blocks ready for parsing.
- Display (Routine/Show.vue): day tabs present colour-coded time blocks (study, break, exercise, revision), accompanied by daily goals and motivational tip cards designed to keep the routine feeling encouraging rather than mechanical.

4.5.3 Voice Integration

Voice capabilities use entirely browser-native APIs, which keeps cost and latency low:

- Speech-to-Text: window.SpeechRecognition (or webkitSpeechRecognition on Chromium-based browsers) converts spoken input into text and populates the chat input field automatically.
- Text-to-Speech: window.speechSynthesis.speak() reads AI responses aloud, with an auto-speak toggle that turns the chatbot into a fully hands-free conversational tutor.
- Composable pattern: both behaviours are wrapped in a useVoice() composable so that any component can adopt voice with a single import.

Table 4.2: AI Model Configuration

Feature	Model	Purpose	Output Format
Study Routine	GPT-4o	Complex structured generation	JSON (time blocks)
AI Teacher Chat	GPT-4o-mini	Conversational tutoring	Natural language
Voice Input	Web Speech API	Speech recognition	Text transcription
Voice Output	Web Speech API	Speech synthesis	Audio playback

4.6 Role-Based Panel Implementation

4.6.1 Admin Panel

The admin panel is the largest surface in the system, with 42 Vue pages spanning every functional module. The dashboard presents stat cards with animated count-up numbers (total students, total teachers, monthly revenue, and average attendance), supplemented by quick-action cards that link to the most frequent operations: marking attendance, recording payments, and issuing notices. The sidebar groups the 19 admin-side controller routes into logical clusters: Academic, People, Operations, Examinations, Finance, Auxiliary, and Online Learning.

4.6.2 Student Panel

The student panel comprises six pages and concentrates on academic information and AI features:

- Dashboard: attendance percentage, pending assignments, current routine status, and AI chat usage at a glance.
- AI Chat: the full conversational interface, with voice support throughout.
- Routines: preference setting on one screen and the generated routine on another, with day tabs for navigation.
- Results: exam scores with grades and class rank where applicable, grouped by exam type.

4.6.3 Parent Panel

The parent panel comprises five pages, each focused on the child rather than the parent:

- Dashboard: children cards summarizing attendance and the most recent set of marks per child.
- My Children: detailed academic profiles with month-on-month statistics.
- Attendance: per-child monthly attendance with color-coded statuses for quick scanning.
- Results: per-child exam results, grouped by exam type and term.
- Notices: school announcements filtered to those targeted at parents.

4.7 Software and Hardware Requirements

Table 4.3: Software Dependencies

Software	Version	Purpose
PHP	8.3+	Backend runtime
Composer	2.x	PHP dependency management
Node.js	20+	Frontend build tools
MySQL	8.x	Relational database
Laravel Herd	Latest	Local development server
Git	2.x	Version control
NPM / Yarn	Latest	JavaScript dependency management

Table 4.4: Hardware Requirements

Component	Minimum	Recommended
Processor	Intel i3 / Apple M1	Intel i5+ / Apple M2+
RAM	4 GB	8 GB+
Storage	10 GB free	20 GB+ SSD
Network	Internet connection	Broadband (for AI API)
OS	macOS / Linux / Windows	macOS (with Herd)

CHAPTER 5
RESULTS AND DISCUSSION

5.1 Introduction

This chapter presents the results of the ClassMatrix implementation. The discussion is organized around five themes: an overall summary of system features, performance metrics, an evaluation of the AI features, an assessment of the user interface, and a comparison with selected existing school management systems. Throughout, the focus is on what the implementation actually achieved against the objectives set out in Chapter 1.

5.2 System Features Summary

The following table summarizes the headline system metrics achieved by the final implementation. These figures are not intended as quality measures in themselves; they are reported to give a sense of the breadth of work delivered across the eight phases.

Table 5.1: System Module Metrics

Metric	Count	Category	Notes
Total Routes	190	All Panels	Admin: 145+, Student: 10, Teacher: 1, Parent: 5, Auth: 28+
Eloquent Models	64	Backend	With relationships, scopes, and small business-logic methods
Vue Pages	61	Frontend	Admin: 42, Student: 6, Auth: 7, Parent: 5, Teacher: 1
Database Migrations	31	Database	Creating 67+ tables across 9 functional categories
Controllers	38	Backend	Admin: 19, Student: 3, Teacher: 1, Parent: 1, others: 14
PHP Enums	16	Backend	Type-safe status fields across attendance, exams, finance, etc.

Metric	Count	Category	Notes
AI Services	2	AI Module	RoutineService (GPT-4o), ChatService (GPT-4o-mini)
Composables	4+	Frontend	useVoice, useAppearance, useCurrentUrl, useToast
Middleware Classes	6	Security	Role-based access control + shared Inertia state
Permissions	40+	RBAC	Spatie Laravel Permission across all modules

5.3 Performance Metrics

The system was evaluated across four practical performance dimensions. The figures below were taken on a development machine (Apple M1, 16 GB RAM, MySQL 8.x running locally through Laravel Herd) and should be read as indicative rather than as the result of formal load testing.

- Page-load time: Inertia.js partial reloads complete page transitions in under 200 ms for data-light screens. Full page loads, including database queries, average between 400 ms and 600 ms.
- Database query efficiency: Eloquent eager loading is used throughout to avoid the classic N+1 query problem. Pages that aggregate across many subjects or invoice items use explicit joins where eager loading would have been less efficient.
- Frontend bundle size: Vite code-splitting ensures that each Vue page loads only its own JavaScript on first navigation, with shared vendor chunks cached between subsequent navigations.
- Concurrent safety: financial operations were exercised under simulated concurrent payments. The lockForUpdate() pessimistic-locking strategy successfully prevented double-payment in every tested scenario, including artificially injected delays inside the transaction.

5.4 AI Feature Evaluation

The two AI features were evaluated qualitatively against criteria appropriate to their purpose. The chatbot was assessed on contextual relevance, conversational coherence, and error handling under failure conditions; the routine generator was assessed primarily on the structural validity of its output and its adherence to the student's stated preferences.

Table 5.2: AI Response Quality Evaluation

Criteria	Routine (GPT-4o)	Chat (GPT-4o-mini)	Notes
Contextual Relevance	High	High	Both use the student's actual subjects and recent scores
Personalization	High	Medium-High	Adapts to preferences and to recent performance
Response Format	Structured JSON	Natural language	Both parse and render correctly in production
Error Handling	Graceful	Graceful	Friendly user-facing messages on any API failure
Voice Integration	N / A	Functional	Browser STT and TTS work consistently in Chromium
Rate-Limit Handling	Retry + Backoff	Retry + Backoff	Exponential backoff on HTTP 429
Cost Efficiency	Moderate	Low	GPT-4o-mini is cost-effective at chat volumes

The AI features deliver effective personalization by drawing on student data already present in the system. The routine generator produces schedules that account for the student's enrolled subjects, observed performance patterns, and stated preferences in a way that a generic AI assistant cannot match. The chatbot maintains a coherent conversational thread and provides explanations adapted to the student's class level [25].

Voice interaction through browser APIs delivers a natural, hands-free experience. Speech-to-Text accuracy depends on browser, microphone, and ambient noise but is generally adequate for the kind of educational queries students actually ask. The auto-speak toggle, in particular, transforms the chatbot into a genuinely conversational tutor rather than a text-only Q-and-A surface [13].

5.5 User Interface Evaluation

The UI was evaluated against a small set of criteria drawn from current web-application standards rather than against an external usability study, which lay outside the project scope:

- Visual consistency: a unified color system (navy / blue / cyan) is carried across every panel, with role-specific accent variations that signal context without breaking the overall identity.
- Responsiveness: Tailwind CSS responsive utilities ensure correct rendering on desktop, tablet, and mobile viewports, verified manually on representative breakpoints.
- Accessibility: Reka UI components (the Vue port of Radix UI primitives) provide ARIA attributes, keyboard navigation, and screen-reader support out of the box, eliminating a class of accessibility regressions that hand-rolled components are prone to.
- Animation quality: CSS-based animations (fade-in, hover lift, count-up) provide visual feedback without JavaScript overhead and without the layout-shift penalties that animation libraries can introduce.
- Navigation depth: role-specific sidebars with logical grouping ensure that most features are reachable within two or three clicks from the dashboard, in line with established usability heuristics.

5.6 Comparison with Existing Systems

To position ClassMatrix relative to existing offerings, we compared its feature set against three established platforms: Fedena (open-source), OpenSIS (open-source), and PowerSchool (commercial enterprise). The comparison is necessarily a high-level one, drawn from publicly available product documentation and our own experience with the open-source systems during the literature survey.

Table 5.3: Feature Comparison with Existing Systems

Feature	ClassMatrix	Fedena	OpenSIS	PowerSchool
Multi-Role Authentication	Yes (5 roles)	Yes (3 roles)	Yes (4 roles)	Yes (5+ roles)
AI Chatbot	Yes	No	No	No
AI Routine Generator	Yes	No	No	No
Voice Interaction	Yes	No	No	No
Online Learning	Yes	Partial	No	Yes
Finance Module	Yes	Yes	Partial	Yes
Library Management	Yes	Yes	No	Partial
Transport Management	Yes	Yes	No	Yes
Modern SPA UI	Yes (Vue 3)	No (jQuery)	No (PHP)	Partial
Open Source	Yes	Yes	Yes	No
Parent Portal	Yes	Yes	Yes	Yes
Real-time Updates	Planned	No	No	Yes

The comparison highlights the central contribution of this work: ClassMatrix is, to the best of our knowledge, the only system in this set that combines comprehensive school management with an AI chatbot, automated routine generation, and voice interaction within a modern SPA architecture. PowerSchool offers broader enterprise capabilities and real-time updates, but it does so at commercial cost and without integrated AI features. Fedena and OpenSIS remain useful baselines for institutions without a development capacity, but neither addresses the AI-tutoring gap [26].

CHAPTER 6
CONCLUSION AND FUTURE WORK

6.1 Conclusion

This project has designed and implemented ClassMatrix, an AI-driven school management system that addresses the gap between traditional school administration platforms and modern AI-powered educational tools. The work shows that integrating Large Language Model capabilities directly into school management workflows is both feasible and useful, and that doing so produces noticeably more relevant student support than bolting an external AI service onto an existing platform.

The principal achievements of the project can be summarized as follows:

1. Complete school management coverage. Eight functional phases were implemented end-to-end, from academic setup and daily attendance through examinations, finance, library, transport, hostel, and online learning. The final system contains 190 routes, 64 models, and 61 Vue pages serving five distinct user roles.
2. Genuine AI integration. OpenAI's GPT models — GPT-4o for routine generation and GPT-4o-mini for the chatbot — were integrated against student academic context, demonstrating that institutional data can effectively personalize AI outputs for educational purposes.
3. Zero-cost voice interaction. Voice input and output were implemented using browser-native Web Speech APIs, showing that natural conversational AI interfaces can be delivered without any per-request voice-processing fee.
4. A modern, maintainable architecture. Built on Laravel 13, Inertia.js v3, Vue 3, and Tailwind CSS v4, the system establishes a foundation suitable for continued development and feature expansion well beyond the academic timeframe of this project.
5. Security and data integrity. The implementation includes role-based access control with more than 40 granular permissions, database transactions with pessimistic locking for financial operations, and consistent server-side validation across every endpoint.

Together, these results validate the core hypothesis of this work: AI-driven features, when tightly integrated with school management data, produce more relevant and personalised student support than standalone AI tools that lack institutional context. The chatbot's ability to reference a student's specific subjects, recent exam scores, and class level creates a contextual tutoring experience that generic AI assistants — however capable in absolute terms — simply cannot replicate without access to the same data.

6.2 Future Work

ClassMatrix achieves the objectives stated in Chapter 1, but several avenues remain open for further development.

6.2.1 Multi-Tenancy for SaaS Deployment

The current single-school architecture can be extended to support multiple institutions through the stancl/tenancy package, enabling SaaS deployment in which each school operates inside an isolated database environment while sharing a single application codebase. This would substantially lower the per-institution cost of adoption.

6.2.2 Mobile Application

Building a mobile application using Flutter or React Native against API endpoints exposed through Laravel Sanctum token authentication would extend accessibility to students and parents who prefer phone-first interaction. The existing database and business logic would remain unchanged; only API route definitions and authentication middleware additions are required.

6.2.3 Real-Time Notifications

Integrating Laravel Reverb (a WebSocket server) would enable real-time notifications for attendance marking, grade publication, fee reminders, and parent alerts. The system already uses event-driven patterns internally, so the change is largely a matter of deploying the WebSocket infrastructure and wiring relevant events into broadcast channels.

6.2.4 Advanced AI Features

Several promising AI extensions present themselves: predictive analytics for student performance based on historical attendance and exam patterns; automated parent reports summarizing weekly progress; intelligent timetable optimization using constraint satisfaction; and adaptive quiz generation that adjusts difficulty in response to ongoing student performance.

6.2.5 File Upload and Media Management

Integrating the Spatie Media Library would enable student photo uploads, assignment file submissions, lesson material attachments, and the generation of PDF report cards through barryvdh/laravel-dompdf. This is a relatively self-contained piece of work that would close one of the more visible gaps in the current implementation.

6.2.6 Analytics Dashboard

A comprehensive analytics surface interactive charts for attendance trends, fee-collection rates, and performance distributions would give administrators actionable insight beyond what tabular reports can convey. Recharts or a similar Vue-friendly library would fit naturally with the existing stack.

6.2.7 Internationalization

Adding multi-language support through Laravel's localization system and Vue i18n would make ClassMatrix accessible to institutions outside English-medium contexts, with Bengali, Hindi, Arabic, and other regional languages as the most immediate candidates given the project's geographical setting.

REFERENCES

[1] S. Ahmed and A. Amro, "Agile large-scale software development success factors, challenges and solutions," i-manager's Journal on Software Engineering, vol. 8, no. 3, pp. 1–12, January–March 2014.

[2] Y. Chen, J. Han, Z. Hu, S. Deng, C. Zhi, and J. Yin, "ChatUniTest: A framework for LLM-based test generation," in Companion Proceedings of the 32nd ACM International Conference, New York, NY, USA, 2024.

[3] M. K. Samal and Y. Bajaj, "Accelerating software quality: Unleashing the power of generative AI for automated test-case generation and bug identification," International Journal for Research in Applied Science & Engineering Technology (IJRASET), vol. 11, pp. 345–349, 2023.

[4] UNESCO, "Education: From disruption to recovery," UNESCO COVID-19 Education Response, 2021. [Online]. Available: https://en.unesco.org/covid19/educationresponse

[5] Fedena, "Fedena — Open-source school management system," [Online]. Available: https://www.fedena.com/

[6] R. Luckin, W. Holmes, M. Griffiths, and L. B. Forcier, Intelligence Unleashed: An Argument for AI as a Tool for Learning. London, U.K.: Pearson Education, 2016.

[7] A. Drigas and P. Angelidakis, "Mobile applications within education: An overview of application paradigms in specific categories," International Journal of Interactive Mobile Technologies (iJIM), vol. 11, no. 4, pp. 17–29, 2017.

[8] K. Poelmans, "ICT and education: An investment in improvement," in Proc. European Conference on Information Systems, 2020.

[9] S. Papadakis, "Evaluating pre-service teachers' acceptance of mobile devices with regards to their age and gender," International Journal of Mobile Learning and Organisation, vol. 12, no. 2, pp. 142–165, 2018.

[10] L. Chen, P. Chen, and Z. Lin, "Artificial intelligence in education: A review," IEEE Access, vol. 8, pp. 75264–75278, 2020.

[11] OpenAI, "GPT-4 technical report," arXiv preprint arXiv:2303.08774, 2023.

[12] W. Holmes, M. Bialik, and C. Fadel, Artificial Intelligence in Education: Promises and Implications for Teaching and Learning. Boston, MA, USA: Center for Curriculum Redesign, 2019.

[13] Mozilla Developer Network, "Web Speech API," MDN Web Docs, 2024. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/Web_Speech_API

[14] Canonical Ltd., "SchoolTool — The global student information system," Canonical, 2016.

[15] PowerSchool, "PowerSchool — Unified K-12 operations platform," [Online]. Available: https://www.powerschool.com/

[16] Khan Academy, "Khanmigo: AI for education," Khan Academy, 2024. [Online]. Available: https://www.khanacademy.org/khan-labs

[17] Carnegie Learning, "AI-powered math solutions," Carnegie Learning, 2024. [Online]. Available: https://www.carnegielearning.com/

[18] Spatie, "Laravel Permission — Associate users with roles and permissions," GitHub, 2024. [Online]. Available: https://github.com/spatie/laravel-permission

[19] D. F. Ferraiolo, D. R. Kuhn, and R. Chandramouli, Role-Based Access Control. Boston, MA, USA: Artech House, 2003.

[20] J. Reinink, "Inertia.js — The modern monolith," Inertia.js Documentation, 2024. [Online]. Available: https://inertiajs.com/

[21] T. Otwell, "Laravel — The PHP framework for web artisans," Laravel Documentation, 2024. [Online]. Available: https://laravel.com/docs/

[22] I. Sommerville, Software Engineering, 10th ed. Harlow, U.K.: Pearson Education, 2015.

[23] C. J. Date, An Introduction to Database Systems, 8th ed. Boston, MA, USA: Addison-Wesley, 2003.

[24] S. Krug, Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability, 3rd ed. Berkeley, CA, USA: New Riders, 2014.

[25] E. Kasneci et al., "ChatGPT for good? On opportunities and challenges of large language models for education," Learning and Individual Differences, vol. 103, 102274, 2023.

[26] G. Siemens and R. S Baker, “Learning analytics and educational data mining: Towards communication and collaboration,” in Proc. 2nd International Conference on Learning Analytics and Knowledge, 2012, pp. 252–254.

[27] M. Woolf, How to Build AI-Powered Applications with OpenAI API. Sebastopol, CA, USA: O’Reilly Media, 2024.