Solution Design & Architecture: UTS_SMS

Version: 1.0

Date: February 17, 2026

Status: Draft

---

1. Executive Summary

UTS_SMS is a comprehensive, enterprise-grade School Management System designed to digitize and automate the academic, administrative, and financial operations of educational institutions. Built on the **.NET 8** ecosystem, it leverages a robust **Monolithic MVC Architecture** to ensure data consistency, rapid development, and ease of deployment.

A key differentiator of UTS_SMS is its integration of **Agentic AI** (via Semantic Kernel), providing intelligent features such as a "Chat with Data" (RAG) system for students and automated administrative insights.

---

2. Architectural Principles

The solution is architected with the following core principles:

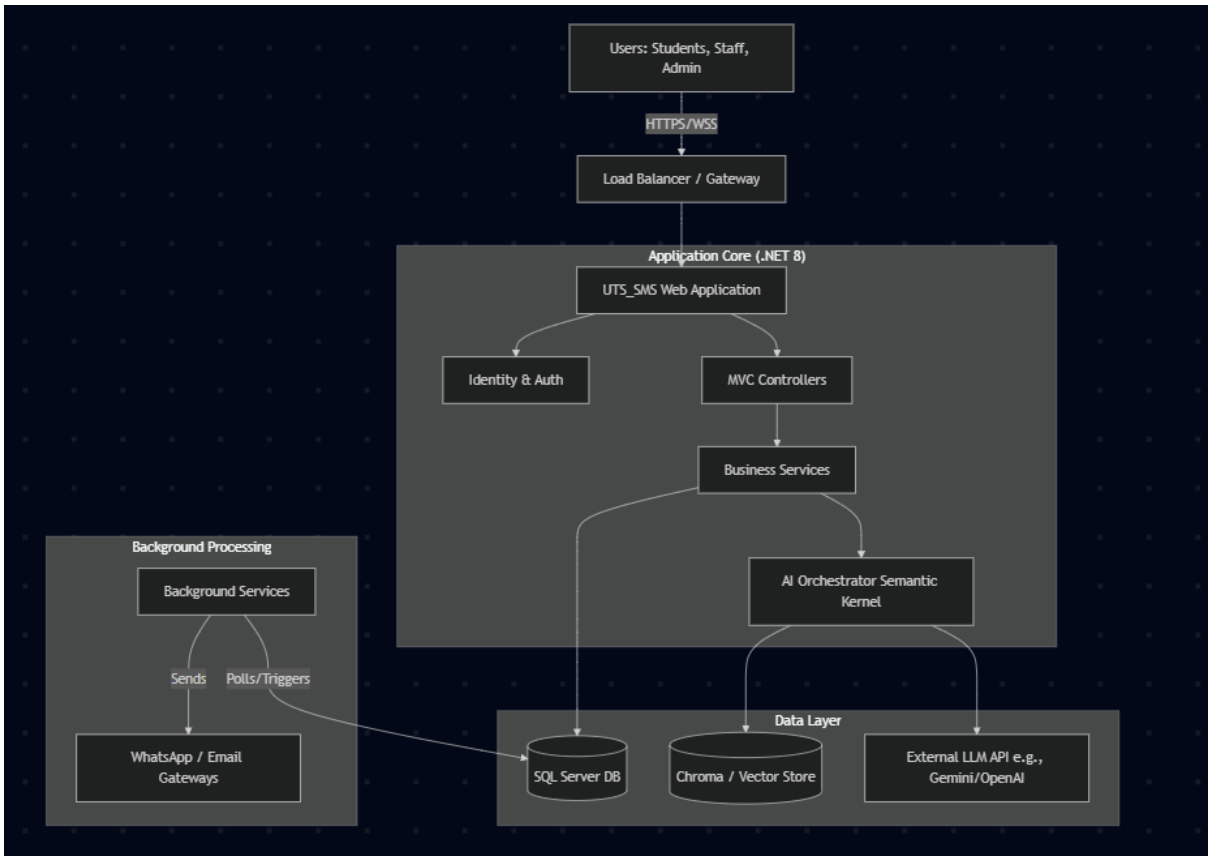
- **Modularity**: While monolithic, the codebase is strictly layered (Controllers -> Services -> Data Access) to facilitate future migration to microservices if required.
- **Scalability**: Stateless web nodes allow horizontal scaling, while background services handle long-running processes asynchronously.
- **Security-First**: Integrated Identity management, Role-Based Access Control (RBAC), and secure API endpoints.
- **AI-Native**: Artificial Intelligence is not an afterthought but integrated into the core service layer for contextual data retrieval and user interaction.

---

3. High-Level Architecture

The system follows a typical N-Tier architecture pattern optimized for ASP.NET Core.

3.1 System Context Diagram (Mermaid)



3.2 Technology Stack

Layer	Technology	Description
Presentation	ASP.NET Core MVC (Razor)	Server-side rendering for SEO and performance.
Styling	Tailwind CSS	Utility-first CSS framework for responsive UI.
Logic	C# / .NET 8	High-performance, type-safe backend logic.
Database	SQL Server (EF Core)	Relational data persistence and complex querying.

Layer	Technology	Description
AI / ML	Microsoft Semantic Kernel	Orchestration for LLM interactions and RAG.
Vector Search	ChromaDB / Custom	Storage for document embeddings (PDFs/Text).
Real-time	SignalR	Live chat and notification pushing.
Background	IHostedService	Payroll calculation, Notification dispatch.

---

4. Key Module Workflows

4.1 AI-Powered Student Assistance (RAG Pipeline)

The system enables students to query academic material using natural language. This works via a Retrieval-Augmented Generation (RAG) pipeline.

Workflow:

1. **Ingestion**: PdfIngestionService reads academic PDFs, chunks text, and generates embeddings using VectorStoreService.
2. **Storage**: Embeddings are stored in the Vector Database linked to specific Subjects/Chapters.
3. **Query**:
    - Student asks: *"Explain the concept of Polymorphism in Chapter 3."*
    - AiChatController passes query to AiChatService.
4. **Retrieval**: System converts query to embedding $\to$ Performs Cosine Similarity Search in Vector DB.
5. **Generation**: Retrieved chunks + User Query are sent to the LLM via Semantic Kernel.
6. **Response**: Context-aware answer is streamed back via SignalR (AiChatHub).

4.2 Financial & Billing Cycle

Handles complex fee structures, dynamic assignments, and transaction tracking.

- **Definition Phase**: Admin defines ClassFee structures (Tuition, Lab, Library) and ClassFeeExtraCharges.
- **Assignment Phase**: Background jobs or triggers assign fees to StudentBillingViewModel.
- **Collection Phase**: BillingController processes payments, generating BillingTransaction and BillingMaster records.
- **Reporting**: BillingReportsController aggregates data for financial health dashboards.

---

5. Database Schema Overview (Key Entities)

The data model is highly relational to ensure integrity across academic and financial records.

- **Identity**: ApplicationUser extends IdentityUser (Fields: CampusId, UserType, ProfileImage).
- **Academic Structure**:
    - Campus $\to$ Class $\to$ ClassSection.
    - Subject $\to$ AcademicMaterial $\to$ Chapter $\to$ DocumentChunk (Vector link).
- **People**:
    - Student (Links to Parent, Class, Billing).
    - Employee (Links to Roles, Attendance, Payroll).
- **Operations**:
    - Attendance (Student), EmployeeAttendance, NamazAttendance.
    - LeaveRequest (State machine: Pending $\to$ Approved/Rejected).

---

6. Scalability and Performance Strategy

6.1 Database Optimization

- **Indexing**: Foreign keys (e.g., StudentId, ClassId) and frequently queried fields (e.g., Date in Attendance) must be indexed.

- **EF Core**: Use AsNoTracking() for read-only reports to reduce overhead.

6.2 Caching Strategy

- **In-Memory Cache**: Use IMemoryCache for configuration data (SiteSettings, ClassDefinitions) that changes rarely.
- **Distributed Cache (Future)**: Redis implementation for Session State and AI Response caching to reduce LLM costs.

6.3 Asynchronous Processing

Heavy operations are offloaded to BackgroundService implementations to keep the UI responsive:

- NotificationBackgroundService: Queues email/WhatsApp messages and sends them in batches.
- SalaryDeductionBackgroundService: Runs nightly to calculate deductions based on attendance.

---

7. Security Design

- **Authentication**: ASP.NET Core Identity with Cookie Authentication.
- **Authorization**:
    - **Role-Based (RBAC)**: [Authorize(Roles = "Admin,Teacher")] decorators on Controllers.
    - **Resource-Based**: Custom logic ensures Parents can only view their own Child's data.
- **Data Protection**:
    - Passwords hashed (PBKDF2).
    - Sensitive files (PDFs) stored with randomized filenames (Guid) to prevent enumeration attacks.
    - CSRF Protection enabled globally.

---

8. Deployment Plan

- **Containerization**: Docker support (via Dockerfile and docker-compose.yml) allows consistent environments across Dev/Prod.

- **CI/CD**: GitHub Actions workflow to build the .NET solution, run tests, and push the Docker image to a registry.
- **Infrastructure**:
    - **Web Server**: Kestrel reverse-proxied by Nginx or IIS.
    - **DB Server**: Managed SQL Server instance.
    - **Storage**: Local SSD or S3-compatible object storage for wwwroot/uploads.