SE 4485: Software Engineering Projects

Fall 2025

Requirement Documentation

Group Number	Group 4
Project Title	Intelligent EMR Note Generation Service
Sponsoring Company	[REDACTED]
Sponsor(s)	[REDACTED] [REDACTED]
Students	1. Samar Siddiqui – Team Lead 2. Kunuth Siddiqui 3. Pedro Garcia 4.Tammy Tran 5. Nashrah Siddiqui 6. Ziyad Alsoudani

Project Title: Intelligent EMR Note Generation Service
Course: SE 4485 – Software Engineering Project
Term: Fall 2025
Company Sponsor: [REDACTED]
Corporate Mentor: [REDACTED]
Business Contact: [REDACTED]
University: The University of Texas at Dallas
Team Size: 6 Students

ABSTRACT
This document defines the architecture of the Intelligent EMR Note Generation Service, created in collaboration with [REDACTED]. The system expands [REDACTED]'s [REDACTED] Simulator by adding an artificial intelligence (AI)-driven note generation feature. It produces accurate and consistent clinical documentation, whether notes are generated manually or automatically.

The architecture follows a microservices model. The Note Generation Service functions as an independent unit that communicates with both the Simulator Service and an external AI service. This setup allows scalability, flexibility, and independent deployment. Each component remains loosely connected, so one can change or restart without disrupting others.

All communication occurs through secure RESTful application programming interfaces (APIs). These APIs ensure structured, reliable, and safe data exchange between the user interface, Note Generation Service, and AI system.

The design supports a React-based front end, a RESTful backend for service coordination with batch processing capabilities that enable users to generate multiple clinical notes on demand. Each microservice handles a specific role—user interaction, AI note generation, or simulation management. This separation improves maintainability and isolates faults to individual components.

By adopting this distributed, service-oriented structure, the system gains flexibility, simplified updates, and long-term reliability. The architecture aligns with recognized standards, including IEEE 1471 and ISO/IEC/IEEE 42030, ensuring consistent and high-quality documentation.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

INTRODUCTION

The *Intelligent EMR Note Generation Service* is a software system developed in collaboration with [REDACTED] to enhance the existing [REDACTED] Simulator. Its primary goal is streamline clinical documentation by integrating artificial intelligence into electronic medical record (EMR) workflows. This service allows clinicians to generate accurate and structured clinical notes either on demand or through batch processing, significantly reducing manual effort and improving efficiency.

The purpose of this document is to define the architectural structure, technologies, and design principles used to implement the *Intelligent EMR Note Generation Service*. The document explains how the system's architecture—based on a microservices pattern—enables modularity, scalability, and independent deployment. The scope covers the system's major components, including the Note Generation Service, Simulator Service, AI integration module, user interface, and batch processing subsystem, as well as their interactions through RESTful APIs.

This architecture ensures that each service operates as a distinct unit while maintaining seamless communication across the system. The document serves as a reference for developers, sponsors, and evaluators to understand how the system achieves its functional and non-functional requirements within a modern, distributed framework.

This Architecture Documentation is organized into several sections:

- Architectural Styles Used – Describes the microservices-based design and how it supports the system's goals.
- Architectural Model – Provides subsystem-level diagrams showing component relationships and data flow.
- Technology, Software, and Hardware Used – Lists all implementation technologies and communication mechanisms between servers.
- Rationale for Architectural Style and Model – Explains the reasoning behind design decisions and their alignment with project needs.
- Traceability from Requirements to Architecture – Maps the system's architecture to the functional and non-functional requirements defined in the Requirements Documentation.
- Configuration Management and Standards – Outlines version control practices and applicable IEEE and ISO/IEC software architecture standards.

ARCHITECTURAL STYLE(S) USED

The architecture of the Intelligent EMR Note Generation Service combines a microservices pattern with a layered design to support modularity and scalability across all major features.

Microservices Pattern:
Each major system feature is an independent deployable service:

- The User Interface microservice manages clinician interactions, including manual note generation, review, and regeneration.

- The Note Generation Service microservice handles AI prompt construction and communication with the Gemini LLM. It also takes care of integration with the [REDACTED] Simulator API.
- Batch Processing microservice supports on-demand batch note creation for multiple patients, allowing users to initiate and monitor batch jobs through the UI.

This separation allows independent development, testing, and scaling of specific features without impacting others. This is essential for supporting both manual and automated note generation modes.

Layered Design:
The architecture is also organized into distinct layers that work together to support the system's functional and non-functional requirements:

- Presentation Layer (React UI): Supports user interaction, allowing clinicians to initiate note generation, preview results, and accept or regenerate notes.
- Application Layer (Note Generation Logic): Manages business logic and orchestrates communication between the frontend, AI, and [REDACTED] services.
- Integration Layer (External Services): Connects securely to Gemini's AI API and the [REDACTED] Simulator using RESTful communication over HTTPS/TLS.
- Batch Processing Layer: Handles on-demand batch execution of note generation for multiple patients, allowing asynchronous processing while maintaining scalability and efficiency.

Support for Application Features:

- The microservices structure enables parallel development and flexible deployment of UI, batch processing, and backend modules.
- The layered approach isolates user-facing logic from integration and AI services, improving maintainability and fault tolerance.
- Combined, these patterns ensure secure data flow, real-time responsiveness, and scalability for batch processing of over 1,000 patient records per run.

ARCHITECTURAL MODEL

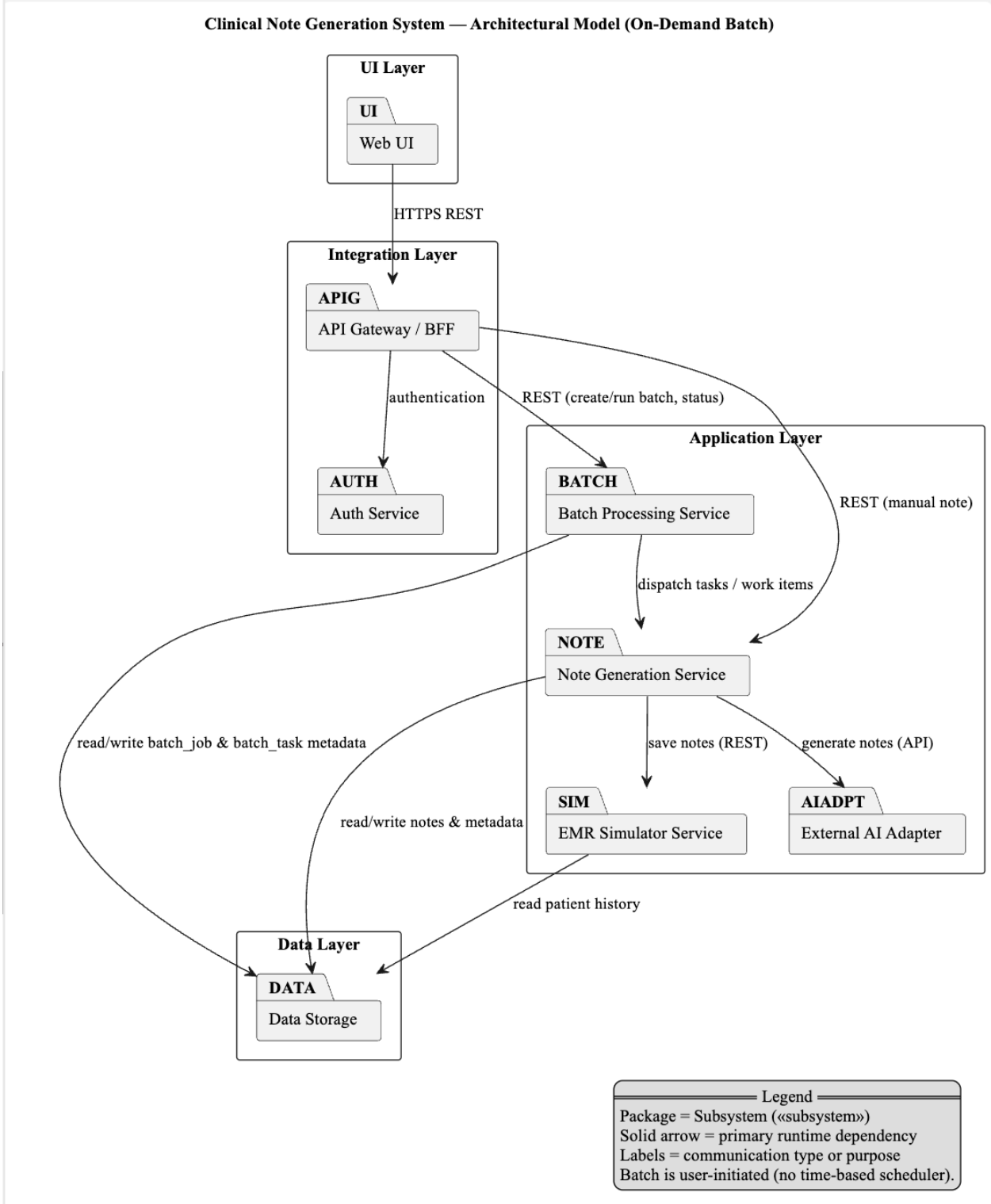

Figure 1.1. Clinical Note Generation Architectural Model

Architectural Model Abbreviations Explained:

- **UI** – Web-based user interface used by clinicians to initiate and monitor note generation.
- **APIG** – API Gateway / Backend-for-Frontend that routes UI requests to backend services.
- **AUTH** – Authentication Service that verifies user identity and access tokens.
- **BATCH** – Batch Processing Service that handles on-demand bulk note generation jobs.
- **NOTE** – Note Generation Service that constructs prompts and interacts with the AI to create notes.
- **SIM** – EMR Simulator Service ([REDACTED] Simulator) that stores and retrieves patient information.
- **AIADPT** – External AI Adapter connecting to the Gemini AI model for note creation.
- **DATA** – Central Data Storage for notes, batch metadata, and related records.

TECHNOLOGY, SOFTWARE, AND HARDWARE USED

The Intelligent EMR Note Generation service is built with a web-based architecture that links the frontend, backend, and AI components through RESTful APIs.

Technologies used for implementation:

- Frontend Technology:

The user interface is developed with React.js. It uses HTML, CSS, and JavaScript to let clinicians generate, preview, and manage AI-generated notes.

- Back End Technology:

The backend is developed with Node.js and Express.js. It exposes RESTful endpoints, processes API requests, integrates with the AI service, and manages on demand batch processing for multiple patients.

- AI Integration:

The system connected to the Gemini LLM through HTTPS requests. The model generates structured, clinically relevant notes based on patient data from the [REDACTED] Simulator.

- Batch Processing:

The system supports **user-initiated batch processing**, allowing clinicians or admins to trigger the generation of notes for multiple patients simultaneously. This enables large-scale testing and simulation while maintaining manual control over execution timing.

Table 1.1. Software and Hardware Required

Category	Component	Purpose/Description
Frontend	React.js	Builds and manages UI for clinician interaction
Backend	Node.js/Express.js	Hosts REST API endpoints and handles data exchange.
Database / API	[REDACTED] Simulator APIs	Stores and retrieves mock patient records and generated notes.
AI Service	Geminin LLM API	Generates AI-powered clinical notes.
Version Control	GitHub	Maintains course code versions and review logs.
Testing Tools	Vitest	For unit and integration testing of frontend and backend modules.
Hardware	Developer Laptop	Used to develop, deploy, and test services.

Communication Between Application Server and Database Server:

The application server (Note Generation Service) communicates with the [REDACTED] Simulator's database through

its RESTful API layer.

1. The frontend sends HTTPS requests to the backend application server.
2. The backend processes the request, calls the Gemini LLM API to generate notes, and formats the response.
3. The backend then makes a secure HTTPS POST call to the [REDACTED] Simulator API, which writes the note into the simulator's database.
4. All data transmission occurs over TLS 1.2 or higher for security.

This indirect API-based communication approach preserves modulatory and aligns with [REDACTED]'s existing system
design.

RATIONALE FOR YOUR ARCHITECTURAL MODEL

The chosen architectural model uses a microservices pattern to keep the system modular, scalable, and easy to update. This approach works best because the Intelligent EMR Note Generation Service needs to connect smoothly with both the [REDACTED] Simulator and an external AI service while staying flexible for future changes.

Each subsystem — the User Interface, Note Generation Service, AI Integration Module, Batch Processing Service, and Simulator Service — operates as an independent microservice communicating through RESTful APIs. This separation of concerns allows developers to modify or redeploy one service without impacting others, which is critical for maintaining uptime and ensuring reliability across different environments.

The microservices model also aligns with the project's Incremental Development Lifecycle, where each component is built and validated independently. This structure supports continuous delivery, enabling partial deployments, and faster feedback from mentors and sponsors during each phase.

Furthermore, the model reinforces the system's non-functional requirements such as reliability (isolating service failures), and security (secure API communication via HTTPS/TLS). By isolating the AI service and simulator integration layers, the architecture ensures compliance with medical data standards while enabling rapid AI model updates.

Overall, this microservices-based architectural model was selected to achieve a balance between flexibility, maintainability, and performance, ensuring that the *Intelligent EMR Note Generation Service* can evolve independently while remaining fully integrated with [REDACTED]'s enterprise simulation ecosystem.

TRACEABILITY FROM REQUIREMENTS TO ARCHITECTURE

Table 2.1: Functional Requirements to Architecture

Requirement ID	Requirement Description	Architectural Element(s)	How It's Achieved
FR-1	Clinician can generate notes manually through the UI.	UI, APIG, NOTE, AIADPT, SIM	User submits note request → APIG routes to NOTE → NOTE calls AIADPT → result saved via SIM.
FR-2	System can generate notes in batches on demand.	UI, APG, BATCH, NOTE, AIADPT, SIM, DATA	Scheduler triggers NOTE → NOTE requests notes from AIADPT → SIM stores generated notes.
FR-3	Clinician can preview, accept, or regenerate AI notes.	UI, NOTE, AIADPT	UI displays generated note → user selects Accept/Regenerate → NOTE re-calls AIADPT if needed.
FR-4	Notes are stored in the [REDACTED] Simulator.	NOTE, SIM, DATA	NOTE sends note data to SIM through REST → SIM writes it to DATA layer.
FR-5	User authentication and secure API access.	APIG, AUTH	APIG forwards each request to AUTH for token validation before processing.

Table 2.2: Non-Functional Requirements to Architecture

NFR ID	Non-Functional Requirement	Architectural Support
NFR-1 Accuracy	AI notes must follow clinical structure and standards.	**NOTE** enforces prompt templates & schema validation; AIADPT returns structured output before save.
NFR-2 Scalability	Must support ≥ 1,000 patients per batch.	**BATCH** orchestrates; **NOTE** workers scale horizontally; configurable batch size/concurrency.
NFR-3 Performance	Manual note generation should complete within 2 seconds.	Async REST calls between NOTE and AIADPT; lightweight UI and API Gateway routing.
NFR-4 Reliability	99 % uptime during scheduled runs.	Independent services (BATCH, NOTE, AIADPT, SIM) isolate failures.
NFR-5 Security	All communication must be encrypted.	AUTH enforces tokens; all REST traffic uses HTTPS/TLS.
NFR-6 Maintainability	Code should be modular and easy to update.	Microservices architecture with clear subsystem boundaries.
NFR-7 Traceability	All changes tracked in version control.	GitHub used for commits, version tags, and peer reviews.

EVIDENCE THE DOCUMENT HAS BEEN PLACED UNDER CONFIGURATION MANAGEMENT

Configuration management will be handled through GitHub for each deliverable. For simultaneous collaboration, GitHub will be used to identify differences between two consecutive versions. For large changes/official submissions, GitHub will be used to maintain the version number of each document and check in/check out major changes for deliverables. Each document will follow the version number format vMAJOR.MINOR.PATCH (e.g. project_plan.pdf v1.0.0).

Table 3.1 Configuration Management Tool

Version	Date	Change Type	Description of Changes	Difference Highlights	Reviewers
v1.0.0	2025-10-15	Created	Architectural Documentation: Creation of initial draft.	Initial commit.	@ksiddii (Kunuth S.) - Ship-It @nashrah-s (Nashrah S.) - Ship-It
v1.1.0	2025-10-23	Updated	Architectural Documentation: Added functional and nonfunctional requirements and architectural model.	+ 1 figure + 2 sections + 3 tables	@ksiddii (Kunuth S.) - Ship-It @nashrah-s (Nashrah S.) - Ship-It
v1.2.0	2025-10-24	Updated	Architectural Documentation: New requirements from [REDACTED] include batch processing and no longer auto-scheduling notes.	Context modifications for architecture and requirements. Edits to architectural model.	@ksiddii (Kunuth S.) - Ship-It @nashrah-s (Nashrah S.) - Ship-It

ENGINEERING STANDARDS AND MULTIPLE CONSTRAINTS

- IEEE Std 1471-2000: Software Architecture [pdf]
- ISO/IEC/IEEE Std 42030:2019: Software, Systems and Enterprise
- Architecture Evaluation Framework [pdf]

ADDITIONAL REFERENCES

- Lattanze, A.J., 2008. *Architecting Software Intensive Systems: A Practitioner's Guide.*
- CRC Press
- Bass, L., Clements, P. and Kazman, R., 2003. *Software Architecture in Practice.*
- Addison-Wesley