SE 4485: Software Engineering Projects

Fall 2025

Detailed Design 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 presents the detailed design of the Intelligent EMR Note Generation Service, a microservices-based system developed to automate and enhance the creation of clinical encounter notes within Electronic Medical Record (EMR) systems. The design builds upon the approved system requirements and architectural documentation to define subsystem interactions, interfaces, data flows, and control mechanisms.

The system is structured into modular microservices. This includes the User Interface, API Gateway, Authentication Service, Note Generation Service, Batch Processing Service**,** EMR Simulator, External AI Adapter, and Data Storage. This is organized using a layered architecture pattern that promotes scalability and maintainability.

This document provides comprehensive UML models, GUI prototypes, traceability matrices linking functional requirements to design components, and rationale supporting design choices based on performance, usability, and compliance constraints. Together, these components establish a robust foundation for implementation, testing, and future integration with enterprise healthcare platforms.

Table of Contents

LIST OF FIGURES

LIST OF TABLES

INTRODUCTION

This *Detailed Design Documentation* provides a comprehensive description of the system design for the *Intelligent EMR Note Generation Service*, a project sponsored by [REDACTED]. This document expands upon the architectural and requirements documentation to specify how each subsystem and feature will be implemented. It defines the internal structure, component interactions, data models, GUI layouts, and UML-based behavioral and structural models necessary for software construction.

The *Intelligent EMR Note Generation Service* is designed to integrate with [REDACTED]'s [REDACTED] Simulator to automate the creation of clinical notes using an AI-driven backend. The system supports both manual

and on-demand batch generation of notes, offering improved accuracy, reduced clinician workload, and seamless integration within EMR workflows.

The purpose of this document is to translate the system's high-level architecture and requirements into a detailed technical design that guides the implementation and testing phases. It ensures that every requirement from the *Requirements Documentation* is traceable to a specific design component.

The scope of this documentation covers:

- The internal design of each microservice, including the User Interface (UI), Authentication, API Gateway, Note Generation Service, Batch Processing Service, EMR Simulator Service, External AI Adapter.
- UML models such as class diagrams (static view) and sequence diagrams (dynamic view) describing subsystem behavior, dependencies, and data flow.
- GUI mockups and design descriptions illustrating user interactions with manual and batch note generation features.
- Traceability matrices linking Functional and Non-Functional Requirements (FRs/NFRs) to specific design elements.
- The rationale for design decisions, emphasizing modularity, maintainability, scalability, and compliance with relevant IEEE and ISO/IEC standards.

This document serves as a reference for developers and project sponsors to ensure that the final implementation remains aligned with project objectives and industry design best practices.

GUI (Graphical User Interface) Design

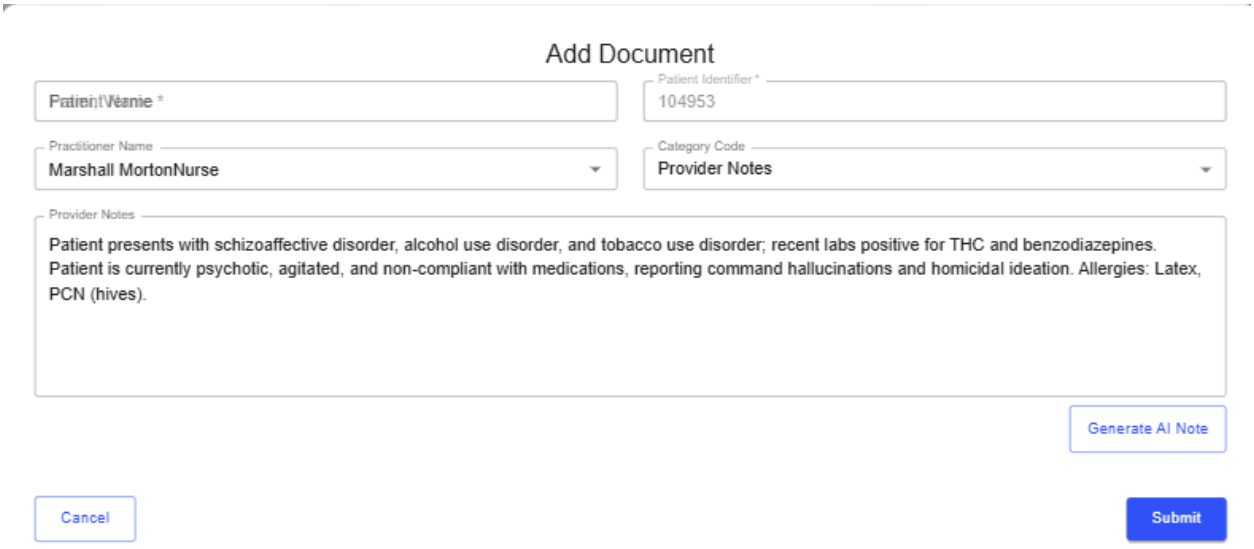

Figure 1: Provider Notes

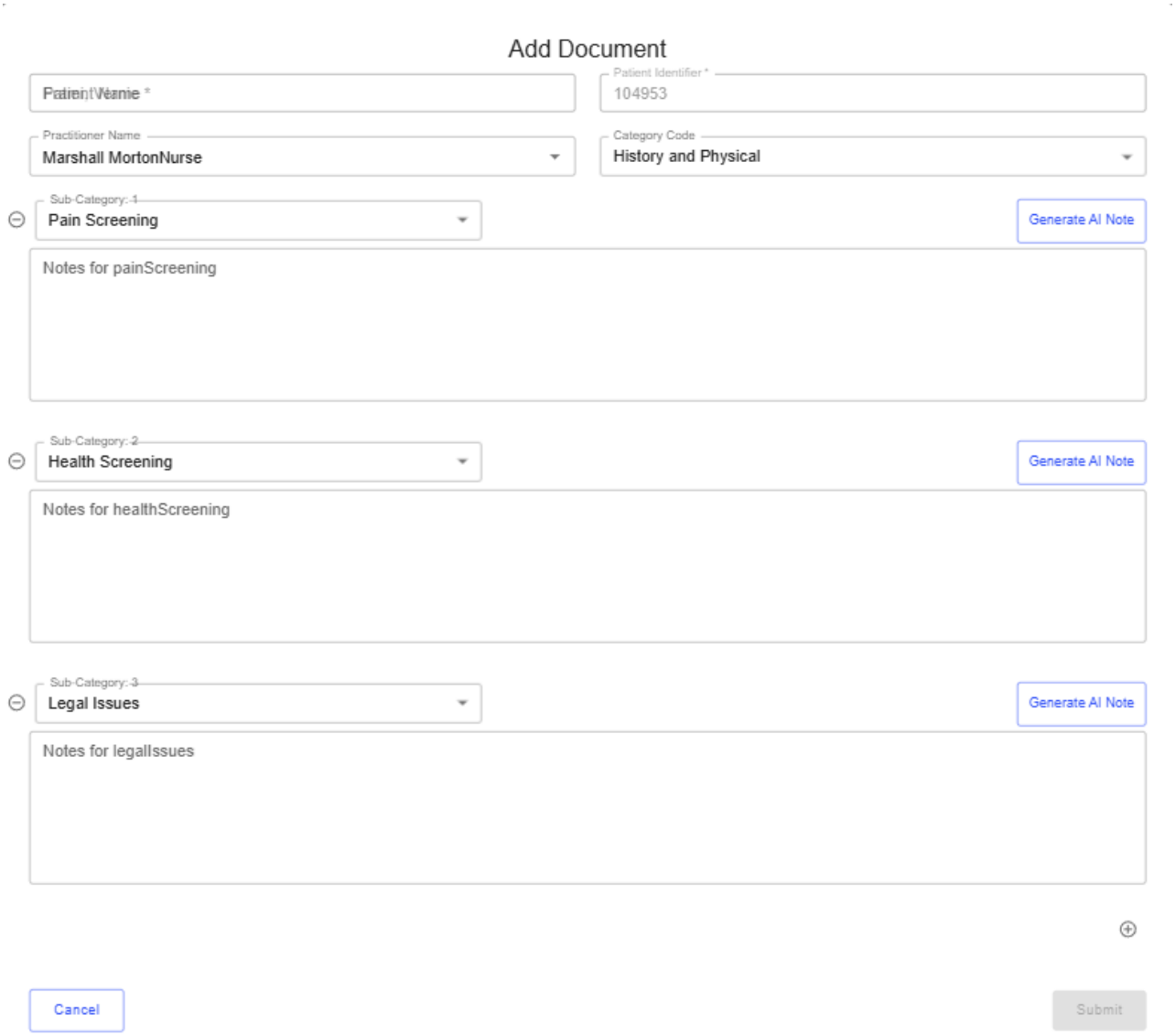

Figure 2: Sub-category Spacing

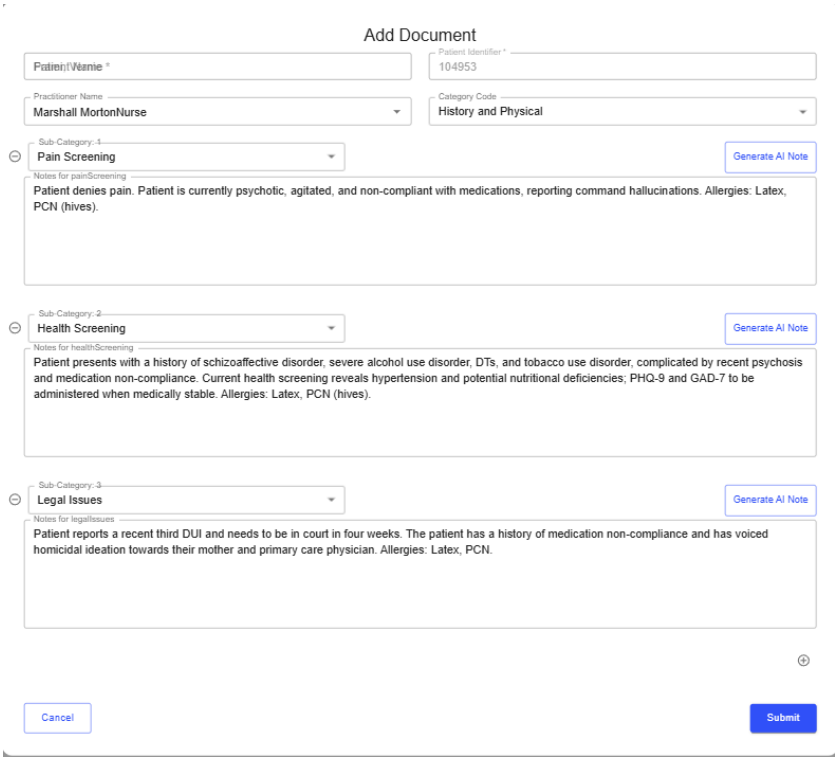

Figure 3: AI Prompts

STATIC MODEL

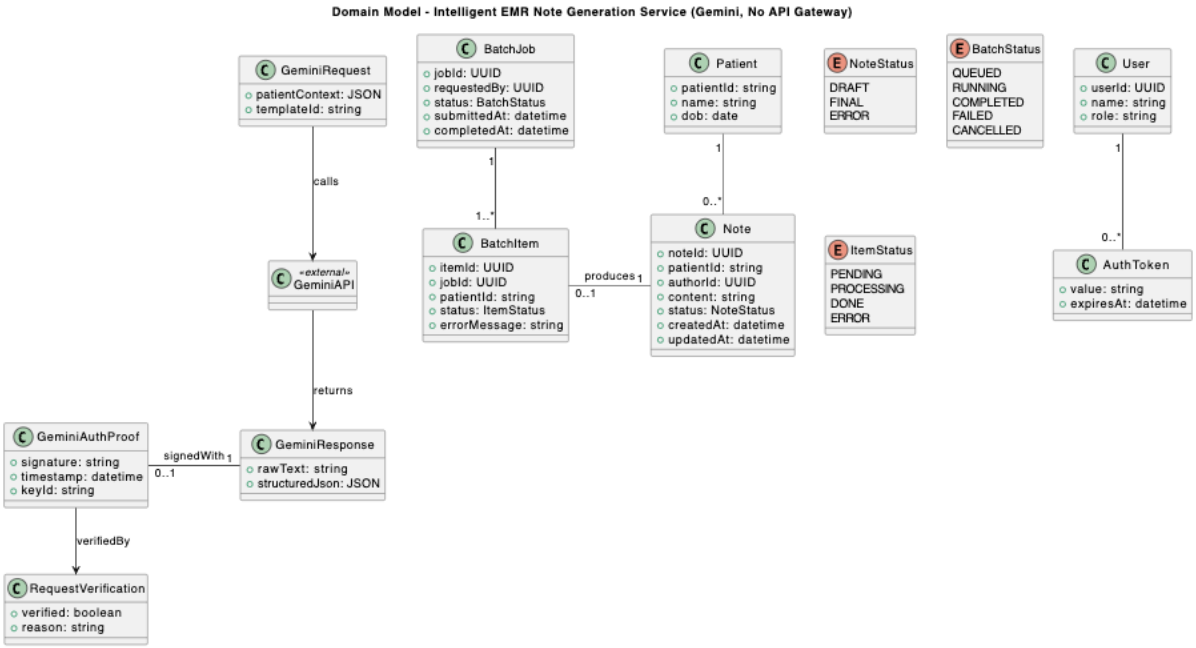

Figure 4: Domain Model

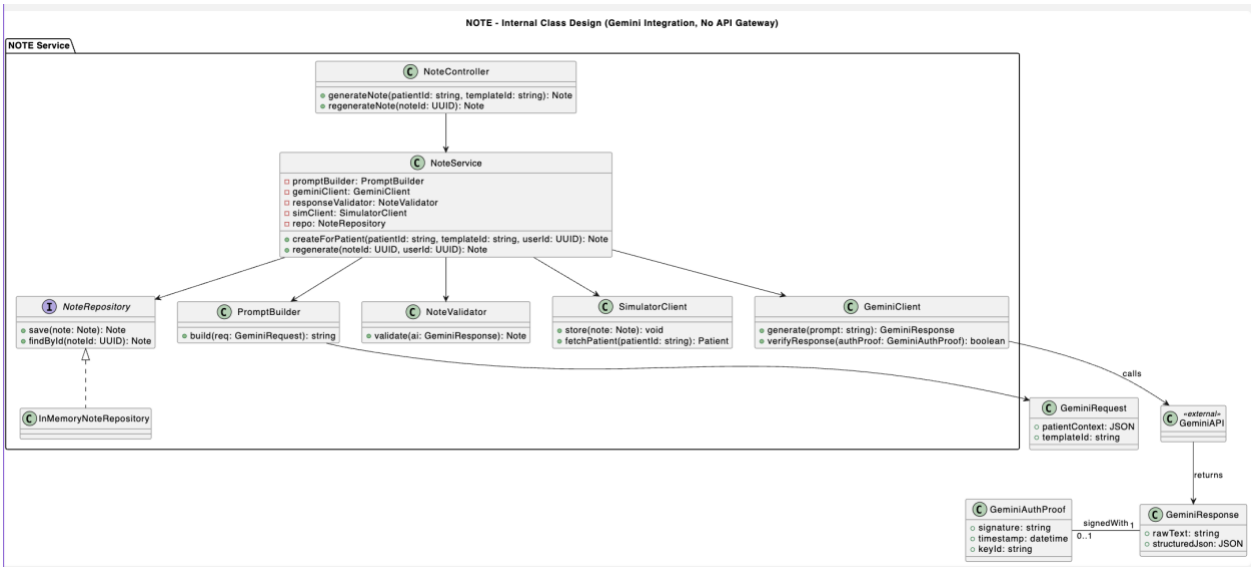

Figure 5: NOTE - Internal Class Design

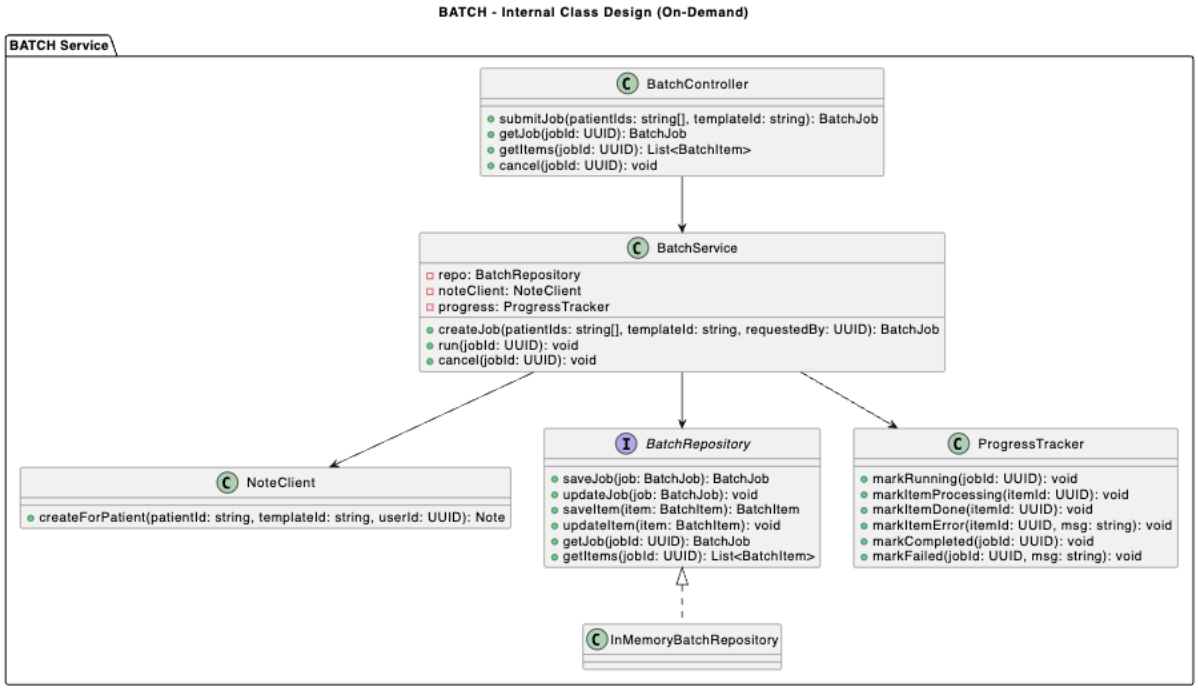

Figure 6: BATCH - Internal Class Design

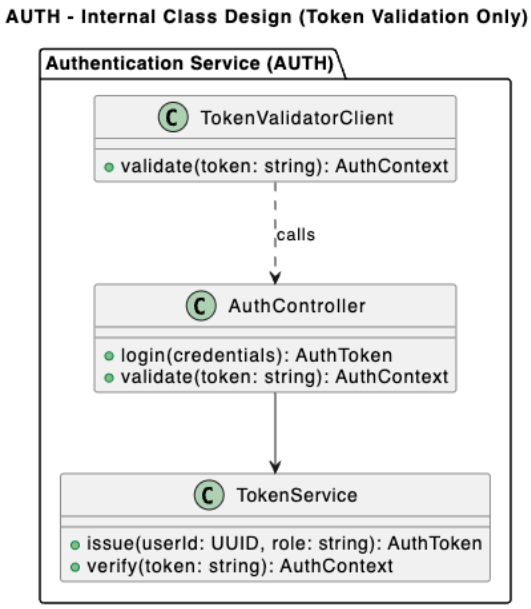

Figure 7: AUTH - Internal Class Design

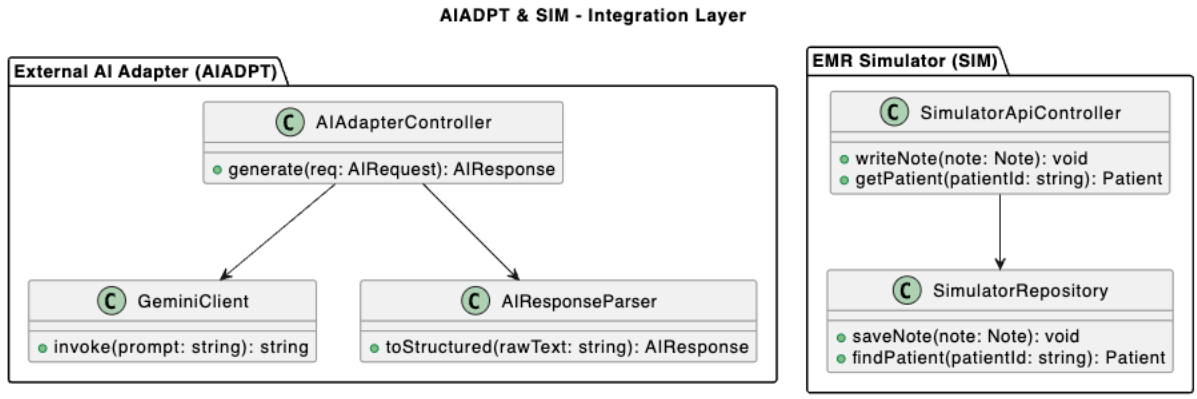

Figure 8: AIADPT & SIM - Integration Layer

DYNAMIC MODEL

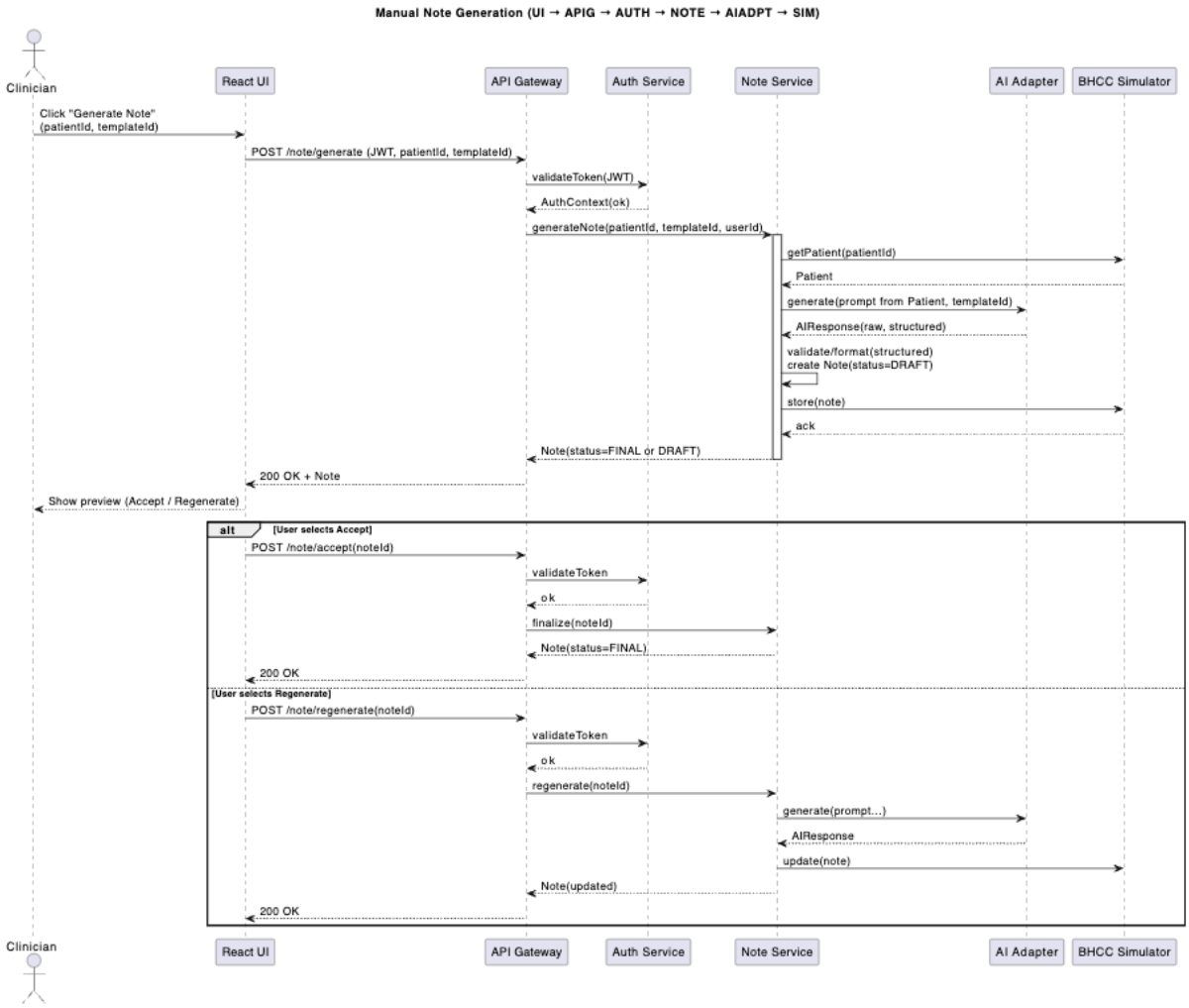

Figure 9: Manual Note Generation

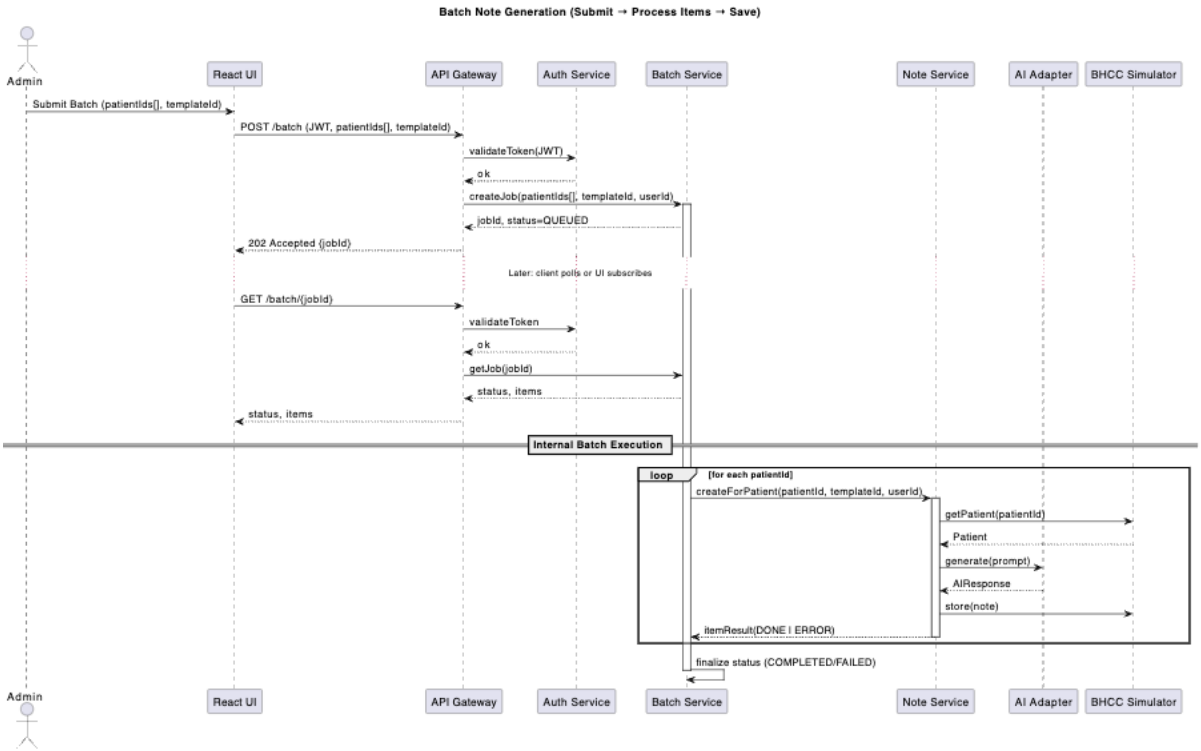

Figure 10: Batch Note Generation

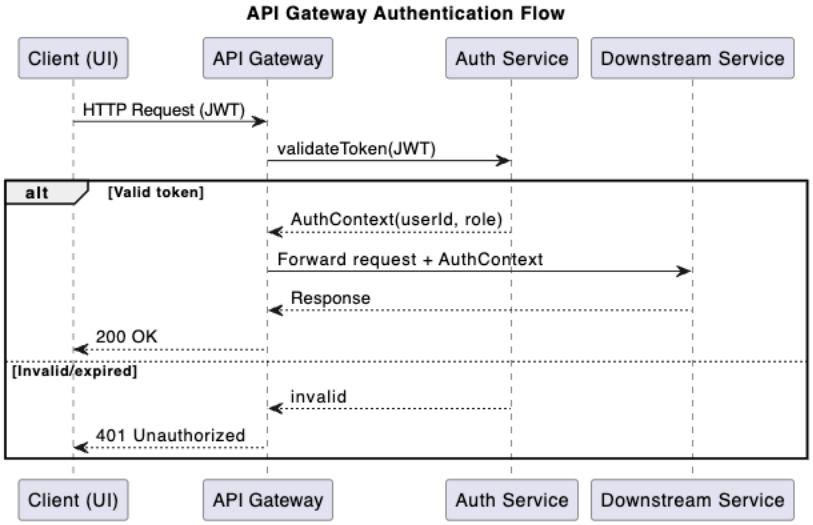

Figure 11: API Gateway Authentication Flow

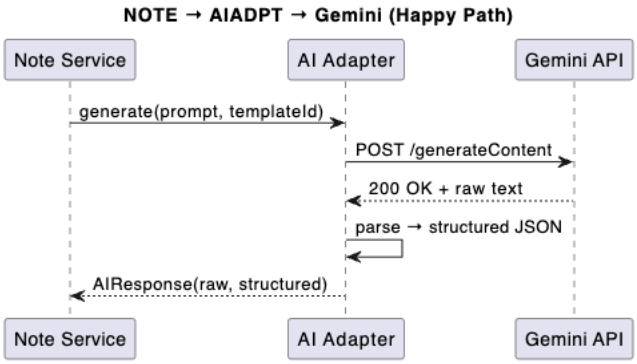

Figure 12: Happy Path

RATIONALE FOR CLASS DIAGRAMS

1) Domain Model

This class diagram models the internal data structure and relationships within the Intelligent EMR Note Generation Service (Gemini). It defines how user-authenticated requests generate clinical notes through batch jobs and items, linking requests, patients, and notes with clear status tracking.

2) Note

This class diagram illustrates the internal design of the Note Service, focusing on how clinical notes are generated and validated through Gemini integration. It shows the collaboration between components such as the NoteController, NoteService, PromptBuilder, and GeminiClient to process requests, validate AI-generated content, and store results.

3) Batch-CHANGED

This class diagram represents the Batch Service responsible for on-demand generation of multiple notes in bulk. It details how the BatchController coordinates with BatchService, which manages job creation, execution, and cancellation through components like NoteClient, BatchRepository, and ProgressTracker.

4) AUTH

This class diagram depicts the Authentication Service (AUTH) focused on token validation and issuance. It illustrates how the TokenValidatorClient interacts with the AuthController, which delegates authentication and verification tasks to the TokenService.

5) AIADPT & SIM

This diagram depicts system integration with external services. The AI Adapter sends prompts to the Gemini API and parses structured responses, while the EMR Simulator stores and retrieves patient data. It ensures interoperability and maintains boundaries between internal logic and external dependencies.

RATIONALE FOR SEQUENCE DIAGRAMS

1) Manual Note Generation

Shows the end-to-end flow when a clinician generates a note: UI → API Gateway → Auth → Note Service → AI Adapter → [REDACTED] Simulator. It highlights token validation, patient lookup, AI generation, validation/formatting, and persistence, supporting real-time, secure note creation.

2) On-Demand Batch Job

Illustrates submitting a batch, returning a jobId, and processing each patient item independently through the Note

Service. Emphasize scalability, progress tracking, and fault isolation while keeping UI polling separate from backend execution.

3) Authentication Flow figure 10
Depicts token validation at the gateway before forwarding to downstream services. Clarifies success (request proceeds with AuthContext) vs. failure (401) paths, ensuring consistent, centralized access control.

4) Minimal AI Adapter Call
Focuses on NOTE → AIADPT → Gemini interaction for the happy path. It shows prompt submission, model response, parsing to structured output, and clean return to the Note Service to decouple AI specifics from business logic.

TRACEABILITY FROM REQUIREMENTS TO DETAILED DESIGN MODEL

Table 1: Functional Requirements Design Mapping

ID	Requirement (Summary)	Design Classes / Packages	Sequence Diagram(s)	Data / Interfaces	Design Implementation Summary
FR-1	Manually generate a clinical note for a patient using AI and show it to the user	**NOTE**: NoteController, NoteService, PromptBuilder, AIAdapterClient, NoteValidator, SimulatorClient, NoteRepository	*SD-1 Manual Note Generation*	Patient, Note, AIRequest, AIResponse; API: /note/generate	UI sends request → APIG → AUTH → NOTE builds prompt, sends to AI, validates, stores in SIM, returns final note.
FR-2	Run an on-demand batch to generate notes for multiple patients	**BATCH**: BatchController, BatchService, BatchRepository, ProgressTracker, RetryPolicy, NoteClient	*SD-2 Batch Note Generation*	BatchJob, BatchItem, Note; APIs: /batch, /batch/{jobId}	Batch job created, NOTE invoked for each patient, results tracked; supports retries and progress updates.
FR-3	Authenticate and authorize all requests	**AUTH**: AuthController, TokenService; **APIG**: AuthMiddleware, TokenValidatorClient	*SD-3 API Gateway Auth Flow*	User, AuthToken; API: /auth/login	Token validation enforced on all requests; tokens expire; AUTH service verifies JWT.
FR-4	Integrate with [REDACTED] Simulator to fetch patient data and store notes	**NOTE.SimulatorClient**, **SIM**: SimulatorApiController, SimulatorRepository	*SD-1*, *SD-2*, *SD-4*	Patient, Note; APIs: getPatient(), storeNote()	Simulator Client isolates EMR operations; note storage and retrieval modeled explicitly.
FR-5	Route all UI calls through an API Gateway to backend services	**APIG**: ApiGatewayController, RouteRegistry, ServiceRouter, AuthMiddleware	*SD-1*, *SD-2*, *SD-3*, *SD-4*	Routing methods; RouteRegistry.match, ServiceRouter.forward*	Gateway mediates every request, handles authentication, and forwards to correct service.

FR-6	View job status and item-level results for batch operations	**BATCH**: BatchService.getJob/getItems, BatchRepository	*SD-2 Batch Note Generation*	BatchJob, BatchItem; API: /batch/{jobId}	Job progress exposed via repository methods and status enums (BatchStatus, ItemStatus).

Table 2: Non-Functional Requirements Design Mapping

NFR	Requirement (Summary)	Design Mechanisms
NFR-1 Reliability & Validation	Notes must be valid and safe	NoteValidator, error-handling flows (SD-4), status enums for tracking, batch retry logic.
NFR-2 Scalability	Handle large batches efficiently	Microservice decomposition; BATCH iteration; APIG routing; decoupled data repositories.
NFR-3 Maintainability	System should be modular and extensible	Clean separation between controllers, services, repositories, and external adapters.
NFR-4 Observability	Track progress and detect failures	ProgressTracker, BatchStatus, and ItemStatus for visibility into operations.
NFR-5 Interoperability	Integrate seamlessly with AI and EMR	Dedicated adapters: AIAdapterController, SimulatorApiController.
NFR-6 Compliance / Traceability	Design must map clearly to requirements	This matrix links each FR/NFR to classes and sequence diagrams for full coverage.

EVIDENCE THE DESIGN MODEL 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 on deliverables. Each document will have a designated version number (e.g. project_plan.pdf v1.0.0).

Version	Date	Change Type	Detailed Design Documentation	Difference Highlights	Reviewers
v1.0.0	2025-11-07	Added	Design Documentation: Creation of initial draft, tables, and figures.	Initial commit.	@presto21 (Pedro G.) — Ship-It v @TammyTran (Tammy T.) — Ship-It
v1.1.0	2025-11-26	Fixed	Design Documentation: Removed extra figures as they were redundant for the document.	-2 tables -1 section	@presto21 (Pedro G.) — Ship-It @TammyTran (Tammy T.) — Ship-It

- Please download the template from this [link]

ENGINEERING STANDARDS AND MULTIPLE CONSTRAINTS

- IEEE Std 1016-1998-(Revision-2009): Software Design [pdf]

ADDITIONAL REFERENCES

- Larman, C., 2012. Applying UML and Patterns: An Introduction to Object Oriented Analysis and Design and Iterative Development. Pearson Education
- Hyman, B., 1998. Fundamentals of Engineering Design. New Jersey: Prentice Hall
- Simon, H.A., 2014. A Student's Introduction to Engineering Design: Pergamon Unified Engineering Series (Vol. 21). Elsevier