
Bilkent University

Department of Computer Engineering

Senior Design Project

Group T2515 - PatchMatch

# Detailed Design Report

22102541, Bertan Uran, bertan.uran@ug.bilkent.edu.tr

22103867, Ekin Köylü, ekin.koylu@ug.bilkent.edu.tr

22203138, Elif Lara Oğuzhan, lara.oguzhan@ug.bilkent.edu.tr

22201668, Emre Yazıcıoğlu, e.yazicioglu@ug.bilkent.edu.tr

22203818, İlke Latifoğlu, ilke.latifoglu@ug.bilkent.edu.tr

Supervisor: Selim Aksoy

Course Instructors: Mert Bıçakçı, İlker Burak Kurt

13/03/2026

1. Introduction..... 3
1.1 Purpose of the system.....3
1.2 Design goals.....4
1.3 Definitions, acronyms, and abbreviations.....7
1.4 Overview.....8
2. Current software architecture..... 9
3. Proposed software architecture..... 10
3.1 Overview.....10
3.2 Subsystem decomposition.....12
3.2.1 Frontend Layer:.....12
3.2.2 Backend Layer:.....13
3.2.3 ML Layer:..... 14
3.2.4 Data Storage Layer:.....15
3.3 Hardware/software mapping..... 16
3.4 Persistent data management.....16
3.5 Access control and security.....17
4. Subsystem services.....18
4.1 Frontend Subsystem..... 18
4.2 Backend Subsystem.....20
4.3 ML Subsystem.....23
4.4 Sequence and Activity Diagrams.....25
5. Test Cases.....29
5.1 Functional test cases.....29
5.2 Non functional test cases.....49
6. Consideration of Various Factors in Engineering Design.....55
6.1 Constraints.....55
6.2 Standards.....59
7. Teamwork Details.....61
7.1 Contributing and functioning effectively on the team..... 61
7.2 Helping creating a collaborative and inclusive environment.....62
7.3 Taking lead role and sharing leadership on the team.....63
8. Glossary.....64
9. References..... 65

# 1. Introduction

Digital pathology has fundamentally changed how tissue analysis is conducted, shifting from optical microscopy to high-resolution digital workflows that generate gigapixel-scale Whole-Slide Images. While this transition has expanded accessibility and enabled computational analysis, it has also exposed a critical gap: existing platforms prioritize storage and visualization over intelligent case comparison. This means that pathologists should either rely on manual inspection or accept slide-level retrieval tools that miss the localized patterns most relevant to diagnosis while searching for similar cases. PatchMatch was conceived to close this gap by combining region-level retrieval, multimodal querying, and explainable similarity reasoning in a single clinically oriented system.

This document presents the Detailed System Design for PatchMatch, developed as part of the Bilkent University CS 492 Senior Design Project. The sections that follow provide the technical foundation needed to guide the implementation, validation, and deployment of the system.

## 1.1 Purpose of the system

PatchMatch is an intelligent Whole-Slide Image (WSI) retrieval system designed for digital pathology. Its purpose is to empower pathologists, medical educators, and researchers to quickly identify visually and clinically similar cases from large-scale digital pathology databases, enabling faster, more consistent, and explainable diagnostic support.

Current digital pathology platforms focus primarily on image storage and visualization, and where retrieval is offered, it operates at the slide level rather than the region level. PatchMatch addresses this gap by supporting region-of-interest (ROI) retrieval, text-based queries, and combined image–text searches, all of which are enriched with explainable similarity reasoning through visual heatmaps and shared feature indicators.

The system is designed as a decision-support tool, not a diagnostic replacement. Clinicians retain full responsibility for diagnostic conclusions. PatchMatch reduces manual search effort, supports knowledge transfer between institutions, and improves reproducibility in pathology research workflows.

## 1.2 Design goals

The following design goals guide the architectural and implementation decisions for PatchMatch.

- Usability:
    - PatchMatch should not disrupt the workflow of the pathologists, so it should be easy to learn and integrate into the existing workflow in under 30 minutes. The pathologists should be able to utilize the application following the instructions and tips provided by the application without formal training.
    - The terminology given in the graphical user interface should align with the pathological terms.
    - The graphical user interface should be easy to interact with, the user should be able to find similar images in no more than 5 interactions.
    - The application should provide feedback after user operations within ten seconds to indicate success or failure of operations.
    - The retrieved images should be displayed in a visually easy to interpret and explainable manner.
    - The images should be examined by zooming in and out just like the real-world procedure.
- Performance:
    - The initial view of a WSI should be available upon selection within 10 seconds for 95% of the requests.
    - Zooming in or out should provide the new image view within 0.5 seconds on average.
    - For a selected region of interest, the most similar top-K images should be displayed under 7 seconds for 90% of the queries.
    - For a text query, the most associated top-K images should be displayed under 7 seconds for 90% of the queries.
    - The system should encode the tiles in a fast manner to create the embeddings and make them accessible.
- Reliability:
    - The system should be available during the clinical hours more than 99% percent of the time. This way, healthcare services will not halt.
    - If an error happens and the system shuts down, it should save the progress up to the point of the error. The progress may consist of

uploading the data, annotating text on the data, selecting the region of interest.
    - In the case of errors the user should see an informative message.
    - The data of WSIs should be preserved in a non-corrupted way. If data corruption happens, the administrator and the uploader of the file should be notified.
    - In case of a system failure, the system should be operable back in under six hours.
- Marketability:
    - PatchMatch targets pathologists, medical educators, and researchers who need fast, explainable case comparison across large WSI archives, which is an unmet need by existing platforms.
    - The system differentiates itself from competitors such as Lagotto [1] by combining ROI-level retrieval, text-based queries, multimodal fusion, and explainable similarity reasoning in a single clinically usable platform.
    - The growing adoption of digital pathology workflows and the increasing demand for AI-assisted diagnostic tools position PatchMatch well within an expanding market.
- Extendibility:
    - The system should support the addition of new retrieval modalities, such as new query types or data sources, without requiring architectural changes to existing components.
    - New WSI file formats should be supportable by adding format handlers without affecting other parts of the pipeline.
    - New AI models for feature extraction should be integrable by running the embedding pipeline on existing WSIs in the background, without interrupting the availability of the current retrieval system during the transition.
- Security:
    - The system should require authenticated access to see patient data.
    - The WSIs, embeddings, and associated reports should be stored in an encrypted manner.
    - Professionals should be provided access based on which data they can see. The scope of the available data should be determined based on the user.
    - The communication through the network should be done in an encrypted way using a protocol like TLS.

- Scalability:
    - The retrieval engine should search images from a few thousands of WSIs to a much larger dataset without compromising latency.
    - Multi-resolution embeddings should be stored using compression or vector representations to decrease the amount of required storage space. This way, more embeddings can be stored with an increased amount of data.
    - Vertical scaling of the GPU and server infrastructure can be considered to increase embedding computation capacity and retrieval throughput as the dataset grows.
    - Different AI models should be added and used without breaking the working logic of the previous models or requiring to reprocess all existing WSIs.
- Maintainability:
    - The codebase should be documented and tracked using a version control system to support collaborative development and future maintenance.
    - The system should expose logging and monitoring mechanisms so that issues can be identified and diagnosed without requiring access to the source code.
- Flexibility:
    - The value of K in top-K similarity retrieval should be adjustable by the administrator without requiring system changes.
    - The system should support multiple WSI file formats including SVS, NDPI, and TIFF to accommodate different scanner types used across institutions.
    - Different similarity metrics and embedding models should be selectable at query time to allow users to tailor retrieval behavior to their specific diagnostic or research needs.
- Modularity:
    - The components such as the retrieval engine, embedding pipeline, and viewer backend should be modular so that each can be updated, replaced, or repaired without affecting the rest of the system.
    - Dependencies between subsystems should be managed through well-defined interfaces so that internal changes to one subsystem do not propagate to others.
- Aesthetics:
    - The user interface should follow a clean, professional visual design consistent with clinical tools, avoiding unnecessary complexity that could distract pathologists during diagnostic workflows.

- Similarity scores, heatmaps, and retrieved slide thumbnails should be presented in a visually coherent layout that makes comparison between cases intuitive without requiring additional interpretation effort.
- The interface should be optimized for the high-resolution monitors typically used in digital pathology environments, making full use of available screen space for image viewing and result presentation.

## 1.3 Definitions, acronyms, and abbreviations

- **WSI (Whole Slide Image)**: Complete digitized microscopy slides with gigapixel sizes (50,000 × 50,000 pixels) that pathologists examine and that PatchMatch searches through.
- **ROI (Region of Interest)**: A specific area of a WSI that the user selects for retrieval. Instead of comparing entire slides, users can select a small region to find similar patches.
- **CBIR (Content-Based Image Retrieval)**: A method of finding images based on their visual content rather than text labels. PatchMatch uses CBIR to find similar tissue patterns by comparing the actual image features.
- **API (Application Programming Interface)**: A set of rules that lets different software talk to each other. PatchMatch uses APIs to connect the frontend with the backend and to access external services.
- **JWT (JSON Web Token):** A compact, signed token used for stateless authentication between the frontend and backend.
- **KVKK (Kişisel Verilerin Korunması Kanunu)**: Turkey's data protection law. Requires storing patient data securely and getting consent before processing medical images.
- **GDPR (General Data Protection Regulation)**: Europe's data privacy law. Similar to KVKK. Both laws guide how to store and protect WSI data.
- **GCP (Google Cloud Platform)**: A cloud platform that provides servers and storage. PatchMatch uses it to host the backend and therefore handle the computational demands of processing large images.
- **LLM (Large Language Model)**: AI models trained on text that can understand and generate language that can help interpret pathology report descriptions.
- **UML (Unified Modeling Language)**: A standard way to visualize software design through diagrams that is used for class diagrams, sequence diagrams, and use case models in this report.
- **TCGA (The Cancer Genome Atlas):** A publicly available US-based cancer genomics archive from which the development dataset was sourced.

- **FAISS (Facebook AI Similarity Search):** An open-source library for efficient approximate nearest-neighbor search over high-dimensional vectors.
- **PLIP (Pathology Language-Image Pretraining):** A vision-language model trained on pathology image-text pairs, used as the primary encoder in PatchMatch.
- **IIIF (International Image Interoperability Framework):** An open standard for serving large images in a resolution-independent and interoperable manner.
- **IEC 62304:** International standard for medical device software lifecycle processes.
- **ISO 14971:** International standard for risk management in medical devices.
- **SVS:** A proprietary WSI file format produced by Aperio scanners.

## 1.4 Overview

The PatchMatch system is built using a layered, modular architecture that separates the user interface, core application logic, machine learning pipeline, and data management to ensure scalability, maintainability, and clinical usability. At the top, the React-based presentation layer provides an interactive interface where pathologists can log in, browse whole-slide images, select regions of interest, and view retrieval results alongside explainability visualizations. The FastAPI-based application layer orchestrates the overall system, handling API requests, managing user authentication, and coordinating with the machine learning pipeline to process queries and return similarity results.

The machine learning pipeline is composed of dedicated modules that work independently to preprocess WSIs, extract multi-resolution patch embeddings using deep learning models, and perform similarity search across large image archives. This separation enables each module such as preprocessing, feature extraction, and retrieval to be developed, tested, and updated without impacting the rest of the system. Multimodal fusion and explainability components are integrated into this pipeline to support text-based queries and visual similarity reasoning through heatmaps and shared feature indicators.

Data management is handled through a combination of file storage and structured database management. WSIs, pathology reports, and computed embeddings are stored as files, with the system designed for local on-premise deployment in a clinical environment. For the purposes of development and testing, these files are currently hosted in a Google Cloud Storage bucket with the backend deployed on a

Google Cloud virtual machine. PostgreSQL manages metadata, user accounts, annotations, and session information for efficient structured data retrieval.

Together, these architectural layers form a cohesive system that supports the entire pathology retrieval workflow, from WSI upload and embedding generation to interactive similarity exploration and secure result sharing, ensuring that the platform is both clinically reliable and technically extensible.

## 2. Current software architecture

The digital pathology ecosystem comprises enterprise platforms, research prototypes, and AI-assisted diagnostic tools. While these systems address important aspects of pathology workflows such as image storage, visualization, and task-specific analysis, none fully satisfies the need for fast, explainable, and flexible similarity retrieval. The following analysis examines seven representative systems, highlighting their capabilities and the specific gaps that motivate the PatchMatch design.

- **Lagotto (Huron) [1]**: Lagotto is a patented CBIR system that links retrieved slides to associated reports and metadata, supporting similarity search across large WSI archives. It is the closest existing competitor to PatchMatch in terms of retrieval intent.
  *Differentiating gaps:* Retrieval operates at the slide level only, with no support for ROI-level matching. The system lacks explainable similarity reasoning, text-to-image retrieval, multi-resolution embeddings, and personalized retrieval sessions.
- **Sectra Digital Pathology [2]**: Sectra is a diagnostic workstation offering WSI viewing, annotation, and teaching workflows, with third-party AI tools providing some degree of explainability.
  *Differentiating gaps:* Similarity-based image retrieval is not supported. Multimodal fusion and text-to-image search are not core features of the platform.
- **Iron Mountain [3]**: Iron Mountain provides secure archival storage and management of WSIs, with metadata-based retrieval for locating files within large archives.
  *Differentiating gaps:* The platform is storage-focused and does not support content-based image retrieval, explainability mechanisms, or any form of multimodal analysis.
- **SMILY [4]**: SMILY is a deep learning-based reverse image search system for histopathology that supports ROI-driven patch-level similarity queries, making it technically closer to PatchMatch than slide-level systems.

*Differentiating gaps:* As a research prototype, SMILY lacks production-level infrastructure and clinical integration readiness. It provides no explanations for similarity results and does not support session-level personalization.

- **Yottixel [5]**: Yottixel is a WSI search engine that uses Bunch of Barcodes (BoB) indexing to support large-scale similarity retrieval across extensive slide archives with efficient binary representations.
  *Differentiating gaps:* Retrieval is primarily slide-level with no ROI-level matching. The system lacks explainability, multimodal fusion, text-based querying, and personalized exploration capabilities.
- **FastPathology [6]**: FastPathology is an open-source platform built on the FAST framework, enabling real-time visualization and deployment of deep learning models for classification, segmentation, and detection on WSIs.
  *Differentiating gaps:* Similarity-based retrieval is not supported. Explainability is limited to task-specific model predictions rather than retrieval reasoning, and multimodal fusion and personalized workflows are absent.
- **Ibex Medical Analytics [7]**: Ibex is a task-specific AI system designed for detection and grading in digital pathology, delivering high performance on predefined diagnostic tasks such as cancer detection and Gleason grading.
  *Differentiating gaps:* The system is designed around fixed diagnostic tasks rather than general-purpose similarity retrieval or cross-dataset case comparison, making it unsuitable as a flexible retrieval platform.

# 3. Proposed software architecture

## 3.1 Overview

PatchMatch is a Content-Based Image Retrieval system built for digital pathology. It enables pathologists and researchers to search a repository of Whole Slide Images using image queries, text queries, or a weighted combination of both through multimodal fusion. The system is designed around a layered, modular architecture that separates the user interface, API points, machine-learning retrieval, and data persistence into four independent layers: Frontend, Backend, ML, and Data Storage.

At the top of the stack, the frontend layer provides an interactive workspace where pathologists can authenticate, explore whole-slide images at full resolution, and run visual or textual searches against the entire slide repository. Search results are presented as ranked cards with similarity scores and clinical context, giving users immediate insight into how and why each match was returned. The backend layer sits under the frontend and it receives every API request, validates parameters, delegates

search and encoding tasks to the ML Layer, manages user accounts and collaboration features through PostgreSQL, and streams high-resolution slide tiles to the viewer. No machine-learning computation takes place in the backend.

The ML Layer is responsible for all computationally intensive retrieval operations. It encodes user queries into dense embedding vectors using a domain-specific vision-language model (PLIP), performs fast approximate nearest-neighbor search over pre-computed patch embeddings via a FAISS index, and fuses similarity scores from multiple modalities: image, text, and clinical report into a single ranked output. Each module within this layer operates independently, which means new analysis capabilities, such as pattern clustering or alternative scoring strategies, can be added or replaced without disrupting existing search functionality.

Data management follows a dual-storage strategy. Whole-slide images, pre-computed embedding vectors, and clinical reports are stored in Google Cloud Storage, which provides the scalability and durability needed for files that can individually exceed one gigabyte. Structured, transactional data such as user accounts, sharing permissions, and slide annotations is managed in PostgreSQL, where ACID guarantees and relational queries ensure consistency under concurrent access. Security is woven throughout the architecture: passwords are hashed with bcrypt, sessions are managed through signed JWT tokens, and role-based access control restricts administrative operations to privileged users.

Together, these four layers form a pipeline that supports the full retrieval workflow, from slide ingestion and offline embedding generation, through real-time multimodal search and score fusion, to secure result presentation and collaborative annotation. It remains flexible enough to accommodate new data sources, search modalities, and deployment configurations as the project evolves.

## 3.2 Subsystem decomposition

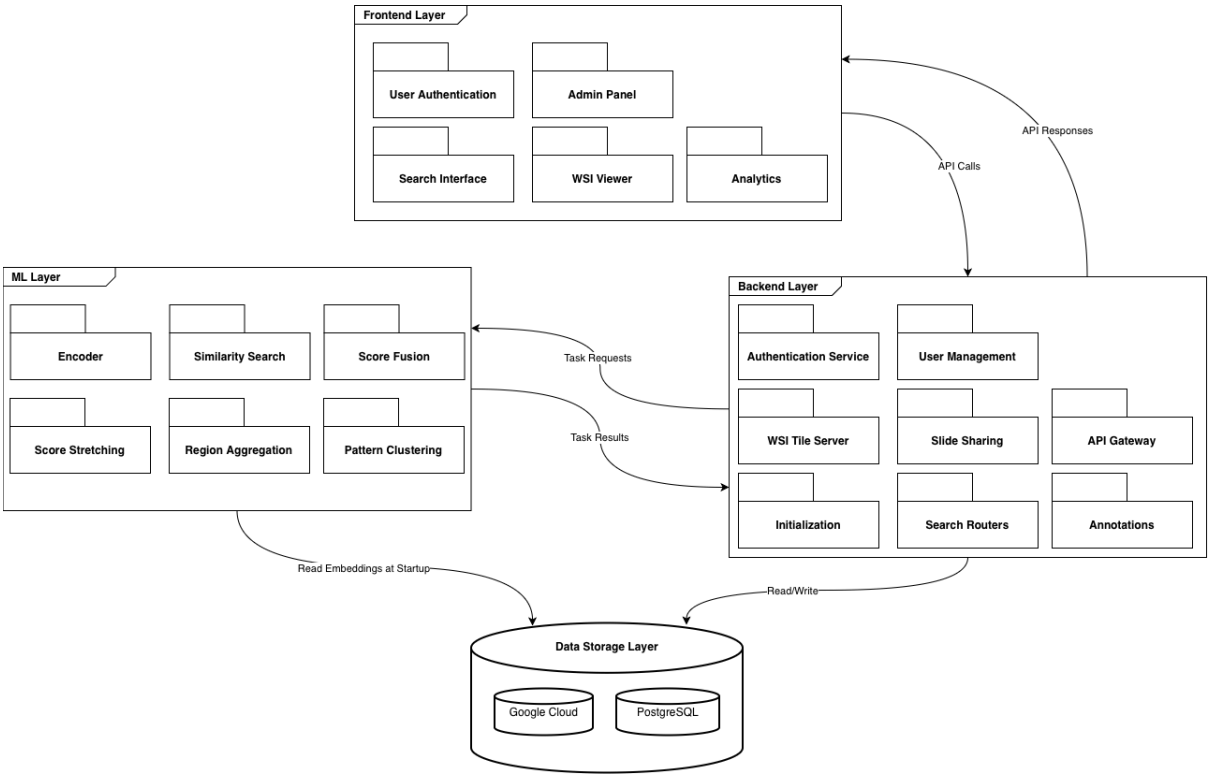


Figure 1. PatchMatch subsystem decomposition.

### 3.2.1 Frontend Layer:

**Purpose:** Provides an interactive interface for pathologists to view whole-slide images, perform visual and textual searches across the slide repository, and explore analysis results.

**Modules**:

Login Interface: Handles user login via JWT-based authentication.

Admin Panel: Provides system administration capabilities including user management.

Search Interface: Main search console. Supports text-to-image, image-to-image, and fused (multimodal) search. Displays ranked result cards with similarity scores and clinical context.

WSI Viewer: Renders whole-slide images via OpenSeadragon with IIIF-compatible tile serving. Supports pan, zoom, and region-of-interest selection for image-based queries.

Analytics: Provides visual summaries of the slide repository and retrieval patterns through dedicated analytics page components.

**Data Flow:**

Frontend → Backend: Search queries (text, image coordinates, fusion weights), authentication credentials, filter parameters, slide and report requests, annotation data, slide sharing requests.

Backend → Frontend: Ranked search results with scores and metadata, IIIF tile images, clinical reports, authentication tokens, system health status.

### 3.2.2 Backend Layer:

**Purpose:** Serves as the central API gateway. Receives requests from the frontend, delegates work to the ML/Retrieval Engine, manages user data through the database, and serves WSI tiles.

**Modules**:

Authentication Service: Manages the full authentication lifecycle, from accepting user credentials to issuing and verifying JWT bearer tokens

User Management: Provides administrative operations like creating, deleting users.

WSI Tile Server: Responsible for making whole-slide images viewable in the browser.

Search Routers: Accept search queries from the frontend (e.g., text-to-image, image-to-image, or fused multimodal) and transmit them.

API Gateway: Serves as the single entry point for all client requests. The backend exposes several routers each acting as a thin HTTP layer that validates incoming parameters and delegates the actual work to the appropriate service.

Slide Sharing: Enables collaboration between users by allowing one user to share specific slides with another

Annotations: Handles the creation, storage, and retrieval of slide-level notes and region annotations added by users.

Initialization: Tracks the multi-phase startup sequence. When the backend starts, it loads embedding files and builds FAISS indices.

**Data Flow:**

Frontend → Backend: Search queries, authentication credentials, filter parameters, fusion weights, annotation requests, slide sharing requests.

Backend → Frontend: Ranked search results with scores and metadata, IIIF tile images, clinical reports, JWT tokens, system initialization status.

Backend → ML Layer: Query text or image patch for encoding, retrieval parameters.

Backend → Storage: User account reads and writes, annotation persistence, sharing record management.

### 3.2.3 ML Layer:

**Purpose:** Encodes queries into embedding vectors, performs approximate nearest-neighbor retrieval, fuses multi-modal scores, and clusters visual patterns. All ML components run in-process as singleton services initialized at startup.

**Modules**:

Encoder: Loads the PLIP vision-language model. Encodes query images and query text into embedding vectors that reside in a shared latent space, enabling direct comparison between image and text representations.

Similarity Search: Builds a FAISS index from all pre-computed patch embeddings at startup, with a NumPy fallback for environments where FAISS cannot be installed. Performs k-nearest-neighbor retrieval in sub-second time.

Score Fusion: Combines “image similarity”, “text similarity”, and “clinical report similarity” into a single fused score using configurable weights.

Region Aggregation: Groups spatially adjacent patch-level results into coherent tissue regions.

Pattern Clustering: Runs K-Means clustering on patch embeddings with zero-shot labeling via PLIP text embeddings.

**Data Flow:**

Backend → ML Layer: Query text or image patch for encoding; retrieval parameters.

ML Layer → Storage: Reads pre-computed embedding files and metadata at startup.

ML Layer → Backend: Ranked lists of slide IDs, scores, coordinates, reports, and aggregated regions.

3.2.4 Data Storage Layer:

**Purpose:** Persists all application data across two complementary storage systems chosen for their respective strengths.

**Components**:

PostgreSQL: Stores user accounts, hashed passwords, slide-sharing records, slide annotations. Chosen for its reliability in managing structured relational data.

Google Cloud Storage: Stores whole-slide images, pre-computed PLIP patch embeddings, patch metadata, clinical reports, and report embeddings. Embedding files follow a per-slide naming convention and are loaded into memory at startup to build the FAISS index.

Note on Clinical Deployment: In the current development environment, WSIs, embeddings, and clinical reports are hosted on Google Cloud Storage for ease of access during development. In a real clinical deployment, this storage layer would reside on hospital-managed on-premise servers to comply with data sovereignty and patient privacy requirements under KVKK and GDPR, ensuring that sensitive pathology data never leaves the institution's controlled infrastructure.

**Data Flow:**

Backend → Data Storage Layer (PostgreSQL): Writes and reads user account data, hashed passwords, session information, annotation records, and slide sharing entries on every relevant API request.

ML Layer → Data Storage Layer (File Storage): Reads pre-computed patch embedding files, patch metadata, clinical report JSON files, and report embedding files at startup to initialize the FAISS index and populate in-memory data structures.

Backend → File Storage: Reads whole-slide image files to serve IIIF-compatible tiles to the frontend on demand.

## 3.3 Hardware/software mapping

PatchMatch does not include any custom hardware components. The system runs entirely on standard server and client infrastructure. The mapping between software components and their target execution environments is as follows:

- Frontend Layer: Runs in the user's web browser on any standard workstation or clinical desktop. No installation is required beyond a modern browser such as Chrome, Firefox, Safari, or Edge. The OpenSeadragon WSI viewer runs entirely client-side as a JavaScript library.
- Backend Layer: Runs on a server-side Docker container hosting the FastAPI application. In the current development environment this container is deployed on a cloud virtual machine. In a clinical deployment this would run on a hospital-managed server.
- ML Layer: Runs within the same backend Docker container as the FastAPI application. The PLIP model and FAISS index are loaded into memory at startup and remain resident for the lifetime of the process. GPU acceleration is used for PLIP inference if available; the system falls back to CPU execution otherwise.
- Data Storage Layer: PostgreSQL runs as a managed cloud database service via Neon in the development environment. File storage for WSIs and embeddings is hosted on Google Cloud Storage during development. Both would be migrated to on-premise hospital infrastructure in a clinical deployment.

The entire system is containerized using Docker Compose. This makes the deployment environment reproducible across development, testing, and production targets without hardware-specific configuration.

## 3.4 Persistent data management

PatchMatch does not implement a custom file system or database engine. However, the nature of WSI data and deep learning embeddings introduces persistent data management challenges that go beyond what a standard relational database alone can address, and the team has made deliberate storage design decisions to handle

these. The core challenge is that whole-slide images are gigapixel-scale files, each potentially several gigabytes in size, and the patch embeddings derived from them are high-dimensional numerical arrays that are not suited to row-based relational storage. To address this, PatchMatch uses a split persistence strategy, which is a deliberate engineering decision for simpler infrastructure and faster startup in the current development context.

- Structured data such as user accounts, annotations, and sharing records is stored in PostgreSQL, where relational integrity and query flexibility are needed.
- Unstructured and numerical data such as WSIs, patch embeddings, patch metadata, clinical reports, and report embeddings are stored as flat files using a per-slide naming convention. Patch vectors are stored as NumPy .npy files and patch metadata as .jsonl files, which allows the entire embedding corpus to be memory-mapped and loaded into a FAISS index at startup in a single bulk read rather than through repeated database queries. In the current development environment these files are hosted in a Google Cloud Storage bucket, which serves as a remotely accessible flat file system and was chosen purely for development convenience. This is provider-agnostic by design and maps directly to on-premise block or object storage in a clinical deployment.

## 3.5 Access control and security

PatchMatch handles sensitive medical imaging data and operates in a clinical context, making access control and security integral to the system design rather than supplementary concerns.

- The system uses JWT-based authentication. When a user submits valid credentials, the backend issues a signed JWT bearer token which the frontend includes in the Authorization header of all subsequent requests. The Authentication Service validates this token on every protected endpoint through a dependency injection mechanism, meaning no protected route can be reached without a valid token. Passwords are stored as hashed values in PostgreSQL and are never transmitted or logged in plaintext.
- Two user roles are defined in the system. Regular users can perform searches, create annotations, and share slides with other users. Administrators have additional access to the Admin Panel, where they can create and delete user accounts and monitor system usage. Access to slides and annotations is scoped to the owning user, meaning a user cannot access another user's data unless it has been explicitly shared with them through the slide sharing mechanism.

- All communication between the frontend and backend is conducted over HTTPS, ensuring that queries, authentication tokens, and retrieval results are protected in transit. WSIs, embeddings, and clinical reports stored in the file storage layer are access-controlled at the infrastructure level through cloud storage bucket permissions during development. In a clinical deployment these would be further protected by hospital network access controls and storage encryption at rest in compliance with GDPR and KVKK requirements.

# 4. Subsystem services

Details of the subsystems and methods can be viewed by downloading the class diagram and opening it in a web browser via the the following link:
UMLClassDiagram_PatchMatch.svg

## 4.1 Frontend Subsystem

The frontend subsystem is a React and TypeScript single-page application served by Vite. On launch, it presents a login screen that obtains a JWT before granting access. The main workspace combines a search bar and clinical criteria dropdowns, and a results grid of ranked patch thumbnails. Selecting a result opens the WSI Viewer, which streams tiles from the backend's IIIF endpoints via OpenSeadragon and supports region-of-interest selection for image-based queries. Additional views include an Analytics page for dataset-level statistics and a System Status dashboard for monitoring initialization, hardware, and index state. Access control is enforced client-side: unauthenticated users are blocked at login, and the Admin Dashboard is gated behind the admin flag.

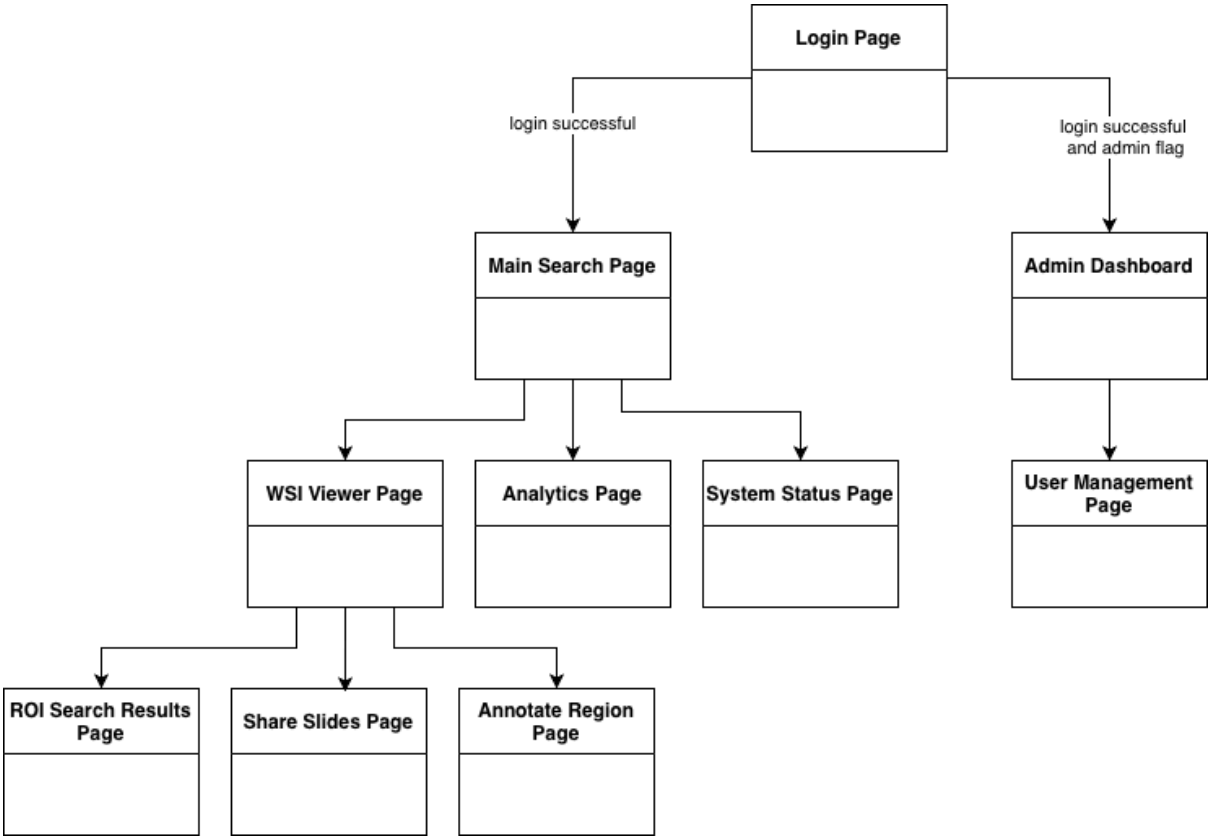


Figure 2. PatchMatch frontend subsystem.

Methods:

Auth Service:

**login(email, password):** Sends credentials to /auth/login, stores the returned JWT in localStorage, and fetches the user profile.

**fetchUser():** Retrieves the current user's profile from /users/me using the stored token; logs out on 401.

**logout():** Clears the token from localStorage and resets all auth state.

Search Service:

**setFilters(filters):** Merges new clinical filter values (ER/PR/HER2, grade) into the active filter set.

**setWeights(weights):** Updates the α/β/γ fusion weight values.

**setSearchMode(mode):** Switches between text, image, and fusedsearch modes.

**clearResults():** Resets result arrays, region selection, and cluster filter to initial state.

**searchByText(query, topK):** Sends a text query to /text-search/text-to-image and stores the ranked patch results.

**searchByImageAndText(slideId, x, y, textQuery, alpha, topK)**: Sends a combined image+text query to /text-search/image-text-paired.

**searchFused(params):** Sends a multimodal fusion query to /text-search/fused-search with weights, filters, region, and cluster parameters; handles both patch and aggregated region responses.

**getSlideReport(slideId):** Fetches the clinical report for a specific slide from /text-search/report/{slideId}.

**toggleClustering(enabled):** Toggles pattern clustering on/off via /text-search/clustering-toggle.

**fetchClusters():** Retrieves cluster summaries from /text-search/clusters.

## 4.2 Backend Subsystem

The backend subsystem is a FastAPI application that acts as the single API gateway between the frontend and all data and ML services. Every request passes through CORS middleware; authenticated endpoints use a dependency-injection layer that decodes the JWT Bearer token and resolves the user from the database. The Authentication Service issues signed JWTs on login and creates bcrypt-hashed user records on registration. The WSI and IIIF services handle slide listing, upload, metadata, thumbnails, and tile serving, backed by a LRU tile cache for fast viewer interaction. Slide Sharing service lets users share slides by email with optional ROI annotations, and persist per-user labels and notes. The Initialization Service provides both a lightweight readiness probe (consumed by the startup screen and Docker) and a comprehensive diagnostics endpoint. At startup, a background thread initializes the ML layer, loading the encoder, ingesting embeddings, building FAISS indices, and running clustering, while the HTTP server remains responsive.

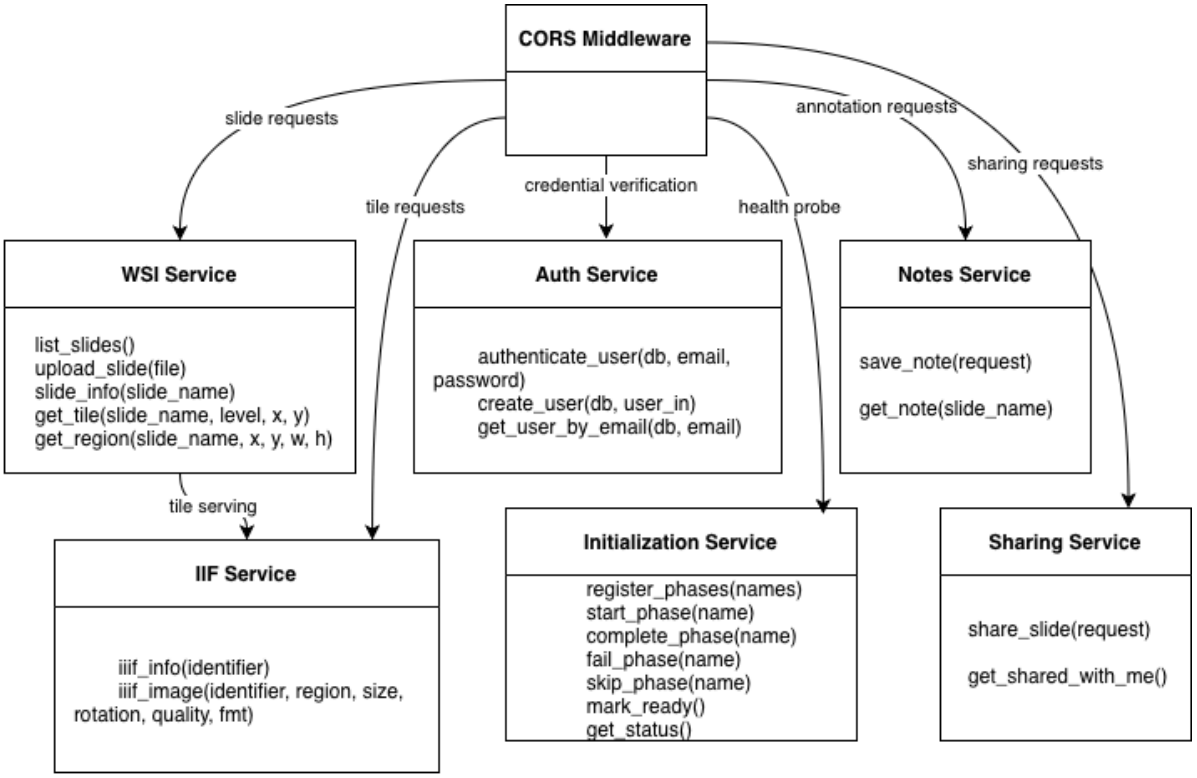


Figure 3. PatchMatch backend subsystem.

Auth Service: Handles user credential verification and account creation.

**authenticate_user(db, email, password):** Looks up the user by email and verifies the bcrypt-hashed password; returns the User or None.

**create_user(db, user_in):** Creates a new User record with a bcrypt-hashed password and commits it to the database.

**get_user_by_email(db, email):** Queries the database for a user by email address.

WSI Service: Manages whole-slide image files and serves tile data.

**list_slides():** Returns all slide filenames in the slides directory filtered by supported extensions.

**upload_slide(file):** Saves an uploaded WSI file to the slides directory after validating the file extension.

**slide_info(slide_name)**: Returns slide metadata: dimensions, tile size, pyramid level count, and downsamples.

**get_tile(slide_name, level, x, y):** Returns a JPEG tile at the given pyramid level and grid position, using an LRU cache of 8 000 tiles.

**get_region(slide_name, x, y, w, h):** Extracts an arbitrary rectangle from level 0 and returns it as JPEG.

IIIF Service: Implements the IIIF Image API v2 for standards-compliant tile serving.

**iiif_info(identifier):** Returns the IIIF info.json manifest with dimensions, tile size, and scale factors.

**iiif_image(identifier, region, size, rotation, quality, fmt):** Returns a JPEG tile by parsing IIIF-standard region/size parameters, selecting the optimal pyramid level, and resizing.

Sharing Service: Enables slide sharing between users.

**share_slide(request):** Creates a SharedSlide

record linking sender, recipient, slide name, message, and optional ROI.

**get_shared_with_me():** Returns all slides shared with the current user, ordered by date.

Notes Service: Persists per-user annotations on slides.

**save_note(request):** Creates or updates a SlideNote record with labels and free-text note for the current user and slide.

**get_note(slide_name):** Returns the saved labels and note for a specific slide, or null if none exists.

Initialization Service: Tracks the multi-phase startup sequence and exposes status to the health endpoint.

**register_phases(names)**: Declares all initialization phase names upfront so progress percentage can be computed.

**start_phase / complete_phase / fail_phase / skip_phase(name):** Updates the status, timing, and detail of individual phases.

**mark_ready():** Signals that all phases are complete; sets global status to "ready".

**get_status():** Returns the current status object consumed by GET /health (status, progress, phases, duration).

## 4.3 ML Subsystem

The ML subsystem runs in-process as a set of singleton services initialized at startup. The Text Search Service loads a vision-language encoder (PLIP by default, with CONCH and UNI2 support), ingests pre-computed patch embeddings and clinical reports from disk, L2-normalizes all vectors, and builds a FAISS inner-product index for sub-second nearest-neighbor retrieval. It supports three search modes: text-only, image-text paired (blended with weight α), and fused multimodal (combining image, text-to-patch, and report similarity with weights α, β, γ, plus optional clinical filters). The Similarity Search Service operates at the slide level, loading one embedding per slide into a separate FAISS index to power the nearest-neighbor lookup. The Pattern Clustering Service groups patch embeddings via K-Means with automatic k selection (silhouette scoring), then labels each cluster through zero-shot classification by matching cluster centroids against candidate pathology terms encoded by the same PLIP text encoder. Cluster assignments are cached to disk with a dataset fingerprint to avoid recomputation on restart, and clustering can be toggled at runtime.

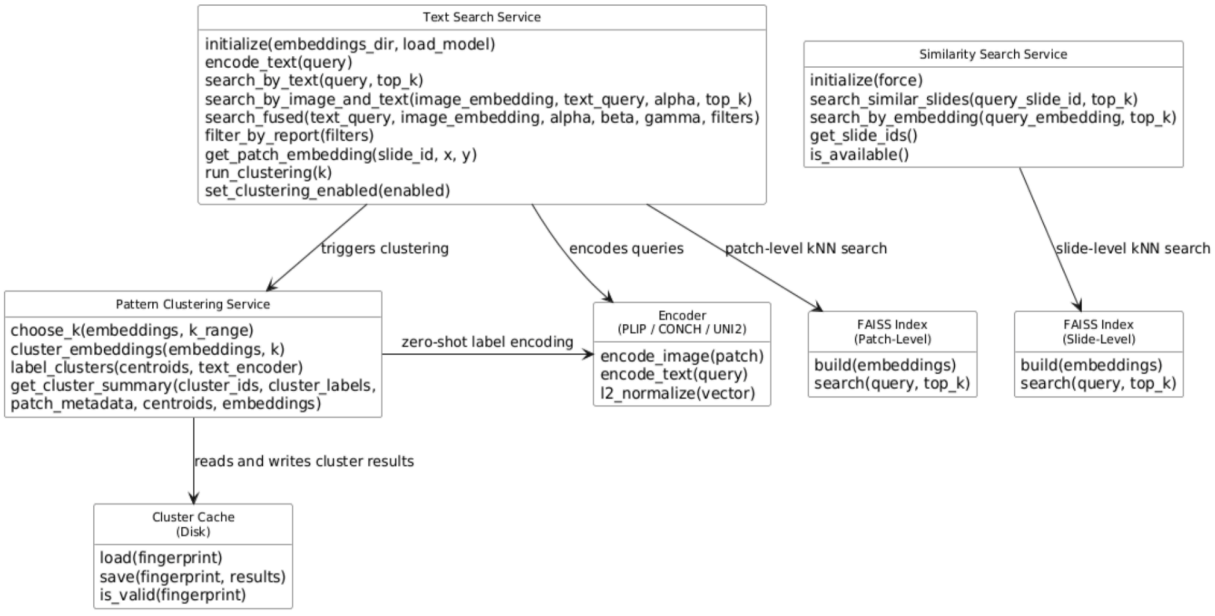


Figure 4. PatchMatch ML subsystem.

Text Search Service: Core retrieval engine that loads embeddings, encodes queries, and performs multimodal search.

**initialize(embeddings_dir, load_model):** Runs the six-phase startup: load encoder, ingest patch embeddings, load clinical reports, load filter schema, build FAISS index, run clustering.

**encode_text(query):** Encodes a natural-language string into a normalized embedding vector using the active encoder (PLIP/CONCH).

**search_by_text(query, top_k):** Ranks all patches by cosine similarity to the text embedding; returns top-k results with stretched scores.

**search_by_image_and_text(image_embedding, text_query, alpha, top_k):** Blends image-to-image and text-to-image similarity with weight α and returns ranked results.

**search_fused(text_query, image_embedding, alpha, beta, gamma, filters, ...):** Combines image (α), text-to-patch (β), and report (γ) similarity with optional clinical filters, cluster filtering, and region aggregation.

**filter_by_report(filters):** Returns the set of slide IDs whose clinical reports match the given filter criteria.

**get_patch_embedding(slide_id, x, y):** Retrieves the pre-computed embedding for a specific patch by slide ID and coordinates.

**run_clustering(k)**: Clusters all patch embeddings via K-Means, labels clusters with zero-shot classification, and caches results to disk.

**set_clustering_enabled(enabled):** Toggles clustering visibility at runtime; triggers computation if clusters were never generated.

Similarity Search Service: Provides slide-level nearest-neighbor retrieval for the "find similar slides" feature.

**initialize(force):** Loads slide-level embeddings from disk and builds a FAISS IndexFlatIP index.

**search_similar_slides(query_slide_id, top_k):** Finds the top-k slides most similar to the query slide by cosine similarity of their slide-level embeddings.

**search_by_embedding(query_embedding, top_k):** Finds the top-k slides most similar to an arbitrary embedding vector.

**get_slide_ids():** Returns the list of all indexed slide IDs.

**is_available():** Returns whether the service has loaded embeddings and is ready for queries.

Pattern Clustering Service: Groups patches into semantic clusters and auto-labels them.

**choose_k(embeddings, k_range):** Selects the optimal number of clusters by evaluating silhouette scores across candidate k values.

**cluster_embeddings(embeddings, k):** Runs K-Means clustering on the patch embedding matrix and returns cluster assignments and centroids.

**label_clusters(centroids, text_encoder):** Encodes candidate pathology terms via the text encoder, computes cosine similarity with each centroid, and assigns labels by greedy matching.

**get_cluster_summary(cluster_ids, cluster_labels, patch_metadata, centroids, embeddings):** Builds a summary for each cluster including label, patch count, and the representative patch (closest to centroid).

## 4.4 Sequence and Activity Diagrams

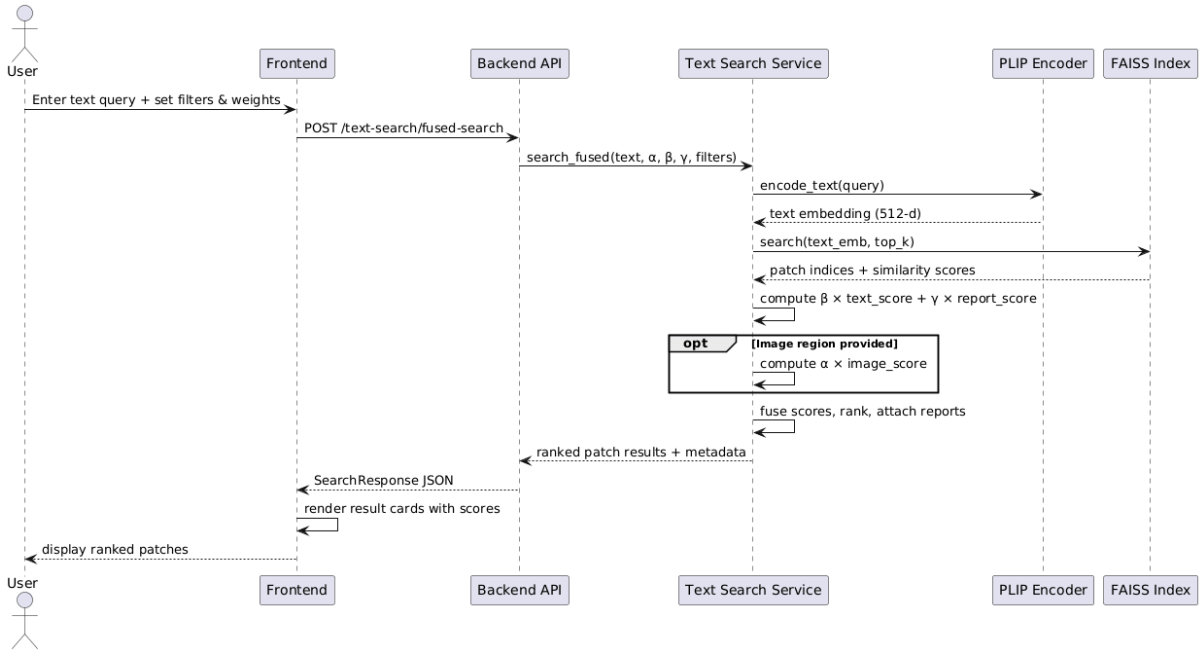


Figure 5. Multi modal fused search sequence diagram.

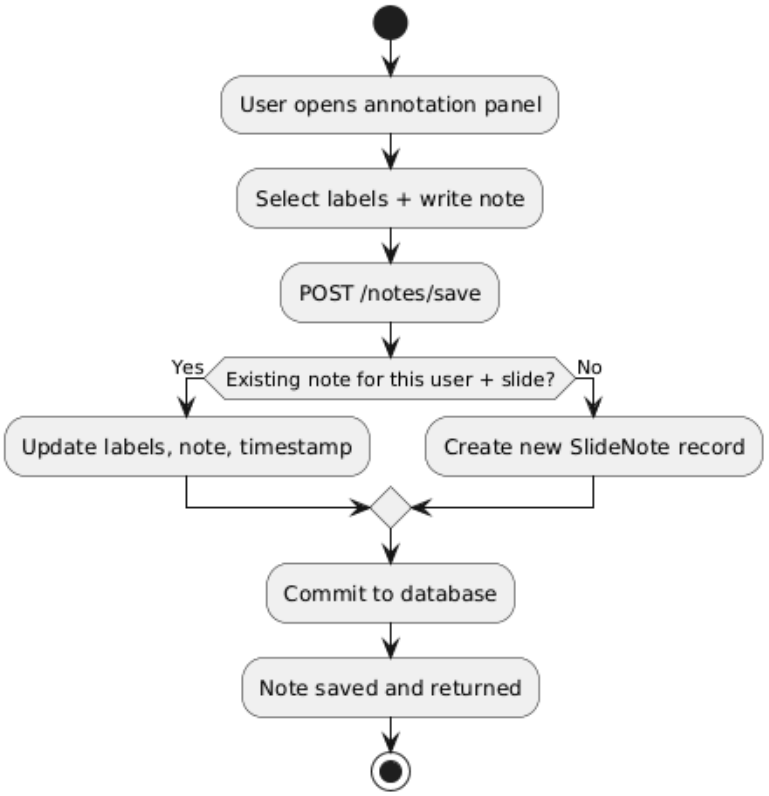


Figure 6. Slide annotation activity diagram.

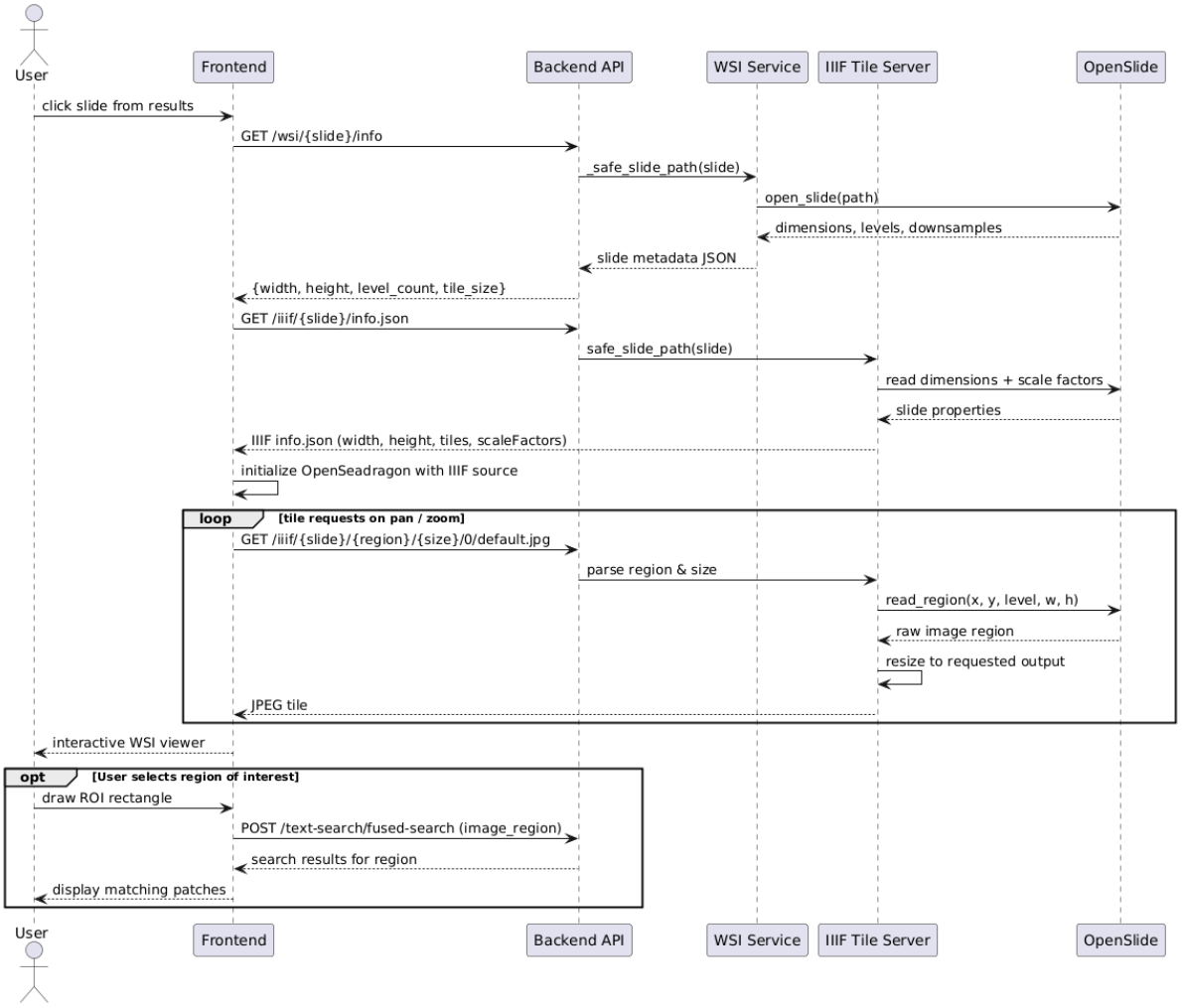


Figure 7. View WSI sequence diagram.

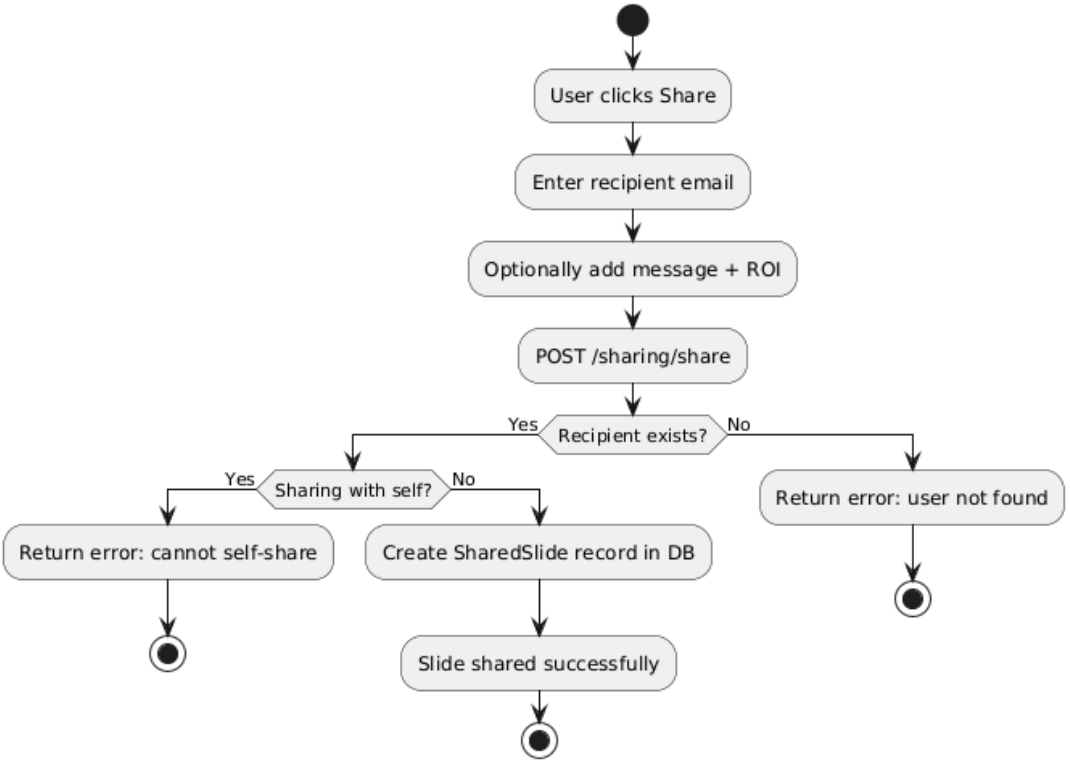


Figure 8. Slide sharing activity diagram.

# 5. Test Cases

## 5.1 Functional test cases

T-1: Signing In Successfully

| Test ID | T-1 | Category | Functional | Severity | |
|---|---|---|---|---|---|
| Objective | Verify that a user can sign in with valid credentials | | | | |
| Steps | 1. Navigate to the sign in page<br>2. Enter a valid email and a password<br>3. Click the Sign In button | | | | |
| Expected | • Session cookie or token is set<br>• User is directed to the home page | | | | |
| Date-Result | | | | | |

T-2: Signing In with Invalid Credentials

| Test ID | T-2 | Category | Functional | Severity | |
|---|---|---|---|---|---|
| Objective | Verify that signing in fails if the credentials are invalid | | | | |
| Steps | 1. Navigate to the sign in page<br>2. Enter an invalid email or a password<br>3. Click the Sign In button | | | | |
| Expected | • Display an error message indicating that the credentials are wrong<br>• No session or token is created<br>• User is not directed to the home page | | | | |
| Date-Result | | | | | |

T-3: Signing In with Missing Field

| Test ID | T-3 | Category | Functional | Severity | |
|---|---|---|---|---|---|
| Objective | Verify that signing in fails if the credentials are invalid | | | | |
| Steps | 1. Navigate to the sign in page<br>2. Enter an invalid email or a password<br>3. Click the Sign In button | | | | |
| Expected | • Display an error message indicating that the credentials are wrong<br>• No session or token is created<br>• User is not directed to the home page | | | | |
| Date-Result | | | | | |

T-4: Signing In with Invalid Email Format

| Test ID | T-4 | Category | Functional | Severity | |
|---|---|---|---|---|---|
| Objective | Verify that signing in fails and triggers a validation error when the email entered is missing the '@' symbol and domain. | | | | |
| Steps | 1. Navigate to the sign in page<br>2. Enter an email string missing the '@' symbol<br>3. Enter any password<br>4. Click the Sign In button | | | | |
| Expected | • A frontend validation error message "Please include an '@' in the email address" is displayed<br>• The form does not submit to the server<br>• No session or token is created<br>• User remains on the sign in page | | | | |
| Date-Result | | | | | |

T-5: Viewing a WSI

| Test ID | T-5 | Category | Functional | Severity | |
|---|---|---|---|---|---|

| | |
|---|---|
| Objective | Verify that a user can successfully open and view the base layer of a selected WSI from the library |
| Steps | 1. Log into the system and navigate to the WSI Library<br>2. Select a valid, fully processed WSI thumbnail<br>3. Click to open the slide in the main viewer interface |
| Expected | • The viewer interface launches without errors<br>• The lowest magnification level (base layer) of the WSI renders fully on the screen<br>• Default viewer controls (zoom, pan, toggle annotations) become active and visible |
| Date-Result | |

T-6: Panning Across WSI

| | | | | | |
|---|---|---|---|---|---|
| Test ID | T-6 | Category | Functional | Severity | |
| Objective | Verify that the user can smoothly pan across the WSI to view different tissue regions at the current magnification | | | | |
| Steps | 1. Open a WSI in the main viewer<br>2. Click and hold the left mouse button on the image<br>3. Drag the cursor in multiple directions to pan across the slide<br>4. Release the mouse button | | | | |
| Expected | • The image tracks accurately with the cursor movement without snapping back<br>• Panning stops immediately when the mouse button is released | | | | |
| Date-Result | | | | | |

T-7: Valid ROI Selection

| | | | | | |
|---|---|---|---|---|---|
| Test ID | T-7 | Category | Functional | Severity | |

| | |
|---|---|
| Objective | Verify that a user can successfully draw and lock a bounding box over a specific region of the WSI |
| Steps | 1. Log into the application with valid credentials<br>2. Open a WSI in the viewer<br>3. Select the ROI mode<br>4. Click and drag the cursor over a specific tissue area to draw an ROI<br>5. Release the cursor |
| Expected | • A visible bounding box is accurately drawn where the user dragged the cursor<br>• The selected ROI coordinates are captured by the system for querying |
| Date-Result | |

T-8: Out-of-Bounds ROI Selection

| Test ID | T-8 | Category | Functional | Severity | |
|---|---|---|---|---|---|
| Objective | Verify that attempting to select an ROI on a blank background or outside slide boundaries prevents an invalid query | | | | |
| Steps | 1. Open a WSI in the viewer.<br>2. Attempt to draw a bounding box completely outside the tissue area<br>3. Click the "Find Similar Slides" button | | | | |
| Expected | • The system prevents the query execution<br>• A helpful validation message "Please select an ROI containing tissue" is displayed<br>• No blank image queries are sent to the backend | | | | |
| Date-Result | | | | | |

T-9: Increasing Magnification (Zooming In)

| Test ID | T-9 | Category | Functional | Severity | |
|---|---|---|---|---|---|

| | |
|---|---|
| Objective | Verify that increasing the magnification successfully loads and renders the higher-resolution WSI tiles for a targeted tissue area |
| Steps | 1. Open a WSI in the main viewer and locate a specific tissue structure at the base magnification<br>2. Position the cursor over the structure<br>3. Use the viewer controls to increase the magnification level to a high power |
| Expected | • The viewer successfully zooms into the targeted spatial coordinates<br>• The system fetches and renders the high-resolution image tiles for that specific area without remaining permanently blurry or pixelated<br>• The UI's magnification indicator correctly updates to reflect the new zoom level |
| Date-Result | |

T-10: Decreasing Magnification (Zooming Out)

| | | | | | |
|---|---|---|---|---|---|
| Test ID | T-10 | Category | Functional | Severity | |
| Objective | Verify that decreasing the magnification smoothly zooms out to reveal the broader tissue context without losing spatial orientation | | | | |
| Steps | 1. Open a WSI and zoom in to a high magnification level on a specific area<br>2. Use the viewer controls to decrease the magnification back to the base layer | | | | |
| Expected | • The viewer smoothly zooms out to display the broader area of the slide<br>• The system successfully unloads the high-resolution tiles and renders the lower-resolution overview tiles<br>• The UI's magnification indicator correctly updates to reflect the new zoom level | | | | |
| Date-Result | | | | | |

T-11: Annotating Slides

| Test ID | T-11 | Category | Functional | Severity | |
|---|---|---|---|---|---|
| Objective | Verify that a Pathologist can draw and save a new annotation on a WSI | | | | |
| Steps | 1. Open WSI in the viewer<br>2. Select one of the selection modes<br>3. Draw a shape around a tissue or a specific group of cells<br>4. Enter text in the annotation label field and click "Save" | | | | |
| Expected | • The drawn annotation remains persistently visible on the slide<br>• The label is correctly associated with the shape<br>• The annotation data is saved to the database | | | | |
| Date-Result | | | | | |

T-12: Displaying Annotations

| Test ID | T-12 | Category | Functional | Severity | |
|---|---|---|---|---|---|
| Objective | Verify that clicking the "View Annotations" toggle to the "ON" position allows a user to display all annotations associated with a slide | | | | |
| Steps | 1. Open a WSI containing one or more existing annotations<br>2. Ensure the "View Annotations" toggle is in the "OFF" position and annotations are currently hidden<br>3. Click the "View Annotations" toggle in the viewer toolbar<br>4. Select the “Pan” mode and zoom across the WSI to inspect the annotated areas<br>5. Click on the annotated areas to view the associated text labels | | | | |
| Expected | • All saved annotation shapes become visible and overlay correctly on the WSI.<br>• The annotations accurately align with the specific tissue coordinates they were originally drawn on, regardless of the zoom level.<br>• Associated text labels are displayed correctly when an annotated shape is selected | | | | |
| Date-Result | | | | | |

T-13: Hiding Annotations

| Test ID | T-13 | Category | Functional | Severity | |
|---|---|---|---|---|---|
| Objective | Verify that clicking the "View Annotations" toggle to the "OFF" position allows a user to hide all annotations associated with a slide | | | | |
| Steps | 1. Open a WSI containing one or more existing annotations<br>2. Ensure the "View Annotations" toggle is in the "ON" position and annotations are currently visible<br>3. Click the "View Annotations" toggle in the viewer toolbar<br>4. Select the “Pan” mode and zoom across the WSI to inspect the annotated areas | | | | |
| Expected | • All saved annotation shapes and associated text labels become hidden on the WSI<br>• The underlying whole-slide image is completely unobstructed<br>• Panning or zooming the image does not cause the hidden annotations to reappear | | | | |
| Date-Result | | | | | |

T-14: Editing Shape of an Annotation

| Test ID | T-14 | Category | Functional | Severity | |
|---|---|---|---|---|---|
| Objective | Verify that the "Edit Annotations" extension allows a user to modify the shape and boundaries of an existing saved annotation. | | | | |
| Steps | 1. Open a WSI containing an existing annotation<br>2. Ensure the "View Annotations" toggle is in the "ON" position and the annotation is currently visible<br>3. Select the existing annotation shape on the slide<br>4. Drag the nodes to alter the shape's boundaries<br>5. Click "Save" | | | | |
| Expected | • The system successfully updates the geometric coordinates of the shape<br>• The new shape boundaries are accurately displayed on the WSI<br>• The changes persist after refreshing the page | | | | |
| Date-Result | | | | | |

T-15: Editing Label of an Annotation

| Test ID | T-15 | Category | Functional | Severity | |
|---|---|---|---|---|---|
| Objective | Verify that the "Edit Annotations" extension allows a user to modify the text label of an existing saved annotation. | | | | |
| Steps | 1. Open a WSI containing an existing annotation<br>2. Ensure the "View Annotations" toggle is in the "ON" position and the annotation is currently visible<br>3. Select the existing annotation on the slide to reveal its current text label<br>4. Change the text label to a new value<br>5. Click "Save" | | | | |
| Expected | • The system successfully updates the text label associated with the annotation<br>• The new label is accurately displayed when the annotated shape is selected<br>• The geometric coordinates of the shape remain completely unchanged<br>• The changes persist after refreshing the page | | | | |
| Date-Result | | | | | |

T-16: Sharing a WSI

| Test ID | T-16 | Category | Functional | Severity | |
|---|---|---|---|---|---|
| Objective | Verify that the "Edit Annotations" extension allows a user to modify the text label of an existing saved annotation. | | | | |
| Steps | 1. Navigate to the Library.<br>2. Open a WSI and click the "Share" button<br>3. Enter the exact email of an existing registered user<br>4. Click "Send" button | | | | |
| Expected | • A success confirmation is displayed<br>• The receiving user sees the shared WSI in their "Shared with Me" tab<br>• Both users can access the same WSI record | | | | |

| | |
|---|---|
| Date-Result | |

T-17: Sharing a WSI with Invalid Email

| | | | | | |
|---|---|---|---|---|---|
| Test ID | T-17 | Category | Functional | Severity | |
| Objective | Verify that the system handles sharing attempts with non-existent users gracefully | | | | |
| Steps | 1. Navigate to the Library.<br>2. Open a WSI and click the "Share" button<br>3. Enter an email address that is not registered in the system<br>4. Click "Send" button | | | | |
| Expected | • The system prevents the share action<br>• An error message is displayed | | | | |
| Date-Result | | | | | |

T-18: Content-based Image Retrieval

| | | | | | |
|---|---|---|---|---|---|
| Test ID | T-18 | Category | Functional | Severity | |
| Objective | Verify that submitting an ROI successfully queries the database and returns visually similar case thumbnails | | | | |
| Steps | 1. Define a valid ROI on the WSI<br>2. Click the "Find Similar Slides" button | | | | |
| Expected | • The system processes the image query successfully<br>• Visually similar WSI patches/thumbnails are displayed in the “Retrieval Results” panel. | | | | |
| Date-Result | | | | | |

T-19: Valid Text-based Image Retrieval

| Test ID | T-19 | Category | Functional | Severity | |
|---|---|---|---|---|---|
| Objective | Verify that entering clinical keywords into the search bar successfully returns relevant results | | | | |
| Steps | 1. Navigate to the Library<br>2. Enter a valid clinical keyword into the text search bar<br>3. Click the "Search" button | | | | |
| Expected | • The system processes the text query successfully<br>• Slides matching the provided clinical keyword are retrieved<br>• Slides are displayed correctly in the Library | | | | |
| Date-Result | | | | | |

T-20: Invalid Text-based Image Retrieval

| Test ID | T-20 | Category | Functional | Severity | |
|---|---|---|---|---|---|
| Objective | Verify that entering unsupported characters or invalid text queries is handled gracefully | | | | |
| Steps | 1. Navigate to the Library<br>2. Enter a string of special characters (e.g., *&^%$#) into the text search bar<br>3. Click the "Search" button | | | | |
| Expected | • The system does not crash or throw an unhandled exception<br>• Specific format validation message is displayed to the user | | | | |
| Date-Result | | | | | |

T-21: Viewing Retrieved Results

| Test ID | T-21 | Category | Functional | Severity | |
|---|---|---|---|---|---|
| Objective | Verify that the retrieval results interface correctly displays the returned WSI patches along with their similarity scores and clinical metadata | | | | |

| | |
|---|---|
| Steps | 1. Execute a successful search query<br>2. Wait for the retrieval process to complete<br>3. Inspect the loaded results grid on the dashboard |
| Expected | • The system displays a list of the retrieved WSI patch thumbnails in the “Retrieval Results” panel<br>• Each thumbnail clearly displays its similarity score<br>• Key clinical metadata is visible alongside or below each result |
| Date-Result | |

T-22: Opening Retrieved Image

| | | | | | |
|---|---|---|---|---|---|
| Test ID | T-22 | Category | Functional | Severity | |
| Objective | Verify that clicking a specific retrieval result successfully navigates the user to the full WSI view at the exact matched coordinates | | | | |
| Steps | 1. Execute a search query and wait for the results to load<br>2. Click on the thumbnail of a specific retrieved result<br>3. Observe the system's navigation and the resulting viewer state | | | | |
| Expected | • The system seamlessly transitions to the main WSI viewer<br>• The viewer automatically loads the correct Whole-Slide Image associated with that result<br>• The viewer automatically pans and zooms to the exact tissue coordinates (ROI) that matched the search query | | | | |
| Date-Result | | | | | |

T-23: Fetching Associated Reports Successfully

| | | | | | |
|---|---|---|---|---|---|
| Test ID | T-23 | Category | Functional | Severity | |
| Objective | Verify that retrieving WSIs via text search successfully fetches and displays the associated clinical reports for the results | | | | |

| | |
|---|---|
| Steps | 1. Navigate to the Search dashboard and select the "Text Search" mode<br>2. Enter a valid clinical keyword and click "Search"<br>3. Select one of the retrieved WSI thumbnails from the results list<br>4. Open or expand the "Associated Reports" panel for that specific WSI |
| Expected | • The system successfully queries the database for the selected WSI<br>• The associated clinical report text is fetched and displayed correctly in the UI panel<br>• The report data accurately matches the selected WSI case |
| Date-Result | |

T-24: Uploading Single WSI

| | | | | | |
|---|---|---|---|---|---|
| Test ID | T-24 | Category | Functional | Severity | |
| Objective | Verify that a System Administrator can successfully upload a single supported WSI file format | | | | |
| Steps | 1. Log in using System Administrator credentials<br>2. Navigate to the "Admin" page<br>3. Click the "Upload Slide" button<br>4. Select a single valid whole-slide image file<br>5. Click "Submit" and monitor the upload progress | | | | |
| Expected | • The individual file uploads successfully without timing out<br>• A success message is displayed to the user confirming the single upload<br>• The newly uploaded WSI appears correctly in the associated user's Library panel | | | | |
| Date-Result | | | | | |

T-25: Uploading Multiple WSIs

| Test ID | T-25 | Category | Functional | Severity | |
|---|---|---|---|---|---|
| Objective | Verify that a Pathologist can successfully upload multiple supported WSI files simultaneously in a single batch | | | | |
| Steps | 1. Log in using System Administrator credentials<br>2. Navigate to the "Admin" page<br>3. Click the "Upload Slide" button<br>4. Select multiple valid whole-slide image files simultaneously<br>5. Click "Submit" and monitor the overall batch upload progress | | | | |
| Expected | • The system successfully handles the concurrent upload of multiple files without crashing or timing out<br>• The UI displays an accurate progress indicator for the entire batch<br>• A final success message is displayed once all files are uploaded<br>• All uploaded WSIs appear correctly in the user's Library panel | | | | |
| Date-Result | | | | | |

T-26: Extracting Valid Tissue Patches

| Test ID | T-26 | Category | Functional | Severity | |
|---|---|---|---|---|---|
| Objective | Verify that the system successfully extracts and tiles valid tissue patches from a newly uploaded WSI while discarding the blank background | | | | |
| Steps | 1. Trigger the patch extraction service (via API or background job) for a valid WSI file<br>2. Monitor the extraction pipeline logs<br>3. Inspect the output directory or database for the generated patches | | | | |
| Expected | • The service successfully divides the WSI into smaller tiles<br>• The background filtering algorithm successfully discards pure white/blank glass tiles<br>• Only tiles containing actual tissue are passed to the next stage of the pipeline | | | | |
| Date-Result | | | | | |

T-27: Handling Corrupted WSI Patch Extraction

| Test ID | T-27 | Category | Functional | Severity | |
|---|---|---|---|---|---|
| Objective | Verify that the extraction service handles corrupted or fully blank WSI files gracefully without crashing the pipeline | | | | |
| Steps | 1. Trigger the patch extraction service (via API or background job) for a completely blank WSI or a corrupted file<br>2. Monitor the extraction pipeline logs<br>3. Check the status of the background worker | | | | |
| Expected | • The service detects the lack of valid tissue or the file corruption<br>• It logs an appropriate error or warning message<br>• The background worker safely terminates the job for that specific file without crashing the entire service | | | | |
| Date-Result | | | | | |

T-28: Computing Patch Embeddings

| Test ID | T-28 | Category | Functional | Severity | |
|---|---|---|---|---|---|
| Objective | Verify that the PLIP model successfully generates numerical feature vectors (embeddings) for a batch of tissue patches. | | | | |
| Steps | 1. Pass a known batch of extracted WSI patches to the embedding computation endpoint<br>2. Await the response from the PLIP model<br>3. Inspect the resulting data payload | | | | |
| Expected | • The model returns a complete set of numerical vectors<br>• The number of output vectors perfectly matches the number of input patches<br>• The vectors contain valid floating-point numbers | | | | |
| Date-Result | | | | | |

T-29: Memory Management in Computing Embeddings

| Test ID | T-29 | Category | Functional | Severity | |
|---|---|---|---|---|---|
| Objective | Verify that the embedding service manages memory correctly when fed an extremely large WSI containing thousands of patches | | | | |
| Steps | 1. Pass an exceptionally large dataset of patches to the embedding service<br>2. Monitor the GPU/RAM usage on the processing server | | | | |
| Expected | • The service processes the patches in manageable batches (chunking) rather than loading everything into memory at once<br>• The server does not throw an Out-of-Memory error<br>• All patches are eventually processed successfully | | | | |
| Date-Result | | | | | |

T-30: Storing Embeddings

| Test ID | T-30 | Category | Functional | Severity | |
|---|---|---|---|---|---|
| Objective | Verify that computed embeddings and their associated metadata are correctly saved to the database | | | | |
| Steps | 1. Trigger the storage function with a payload of computed embeddings, patch coordinates, and the parent WSI ID<br>2. Query the database directly for that specific WSI ID<br>3. Retrieve the stored records | | | | |
| Expected | • The database successfully inserts the records without schema validation errors<br>• The queried data exactly matches the payload sent (embeddings, X/Y coordinates, and WSI ID map correctly) | | | | |
| Date-Result | | | | | |

T-31: Handling Duplicate Embedding Insertions

| Test ID | T-31 | Category | Functional | Severity | |
|---|---|---|---|---|---|
| Objective | Verify that the storage service prevents or safely handles duplicate embedding insertions for the same WSI | | | | |
| Steps | 1. Identify a WSI ID that already has embeddings stored in the database.<br>2. Re-trigger the storage payload for that exact same WSI ID and embedding set<br>3. Check the database row count for that WSI ID | | | | |
| Expected | • The system detects the duplicate data<br>• It rejects the new insertion with a "Duplicate entry" log<br>• The database does not bloat with duplicate rows for the same spatial coordinates | | | | |
| Date-Result | | | | | |

T-32: Updating Similarity Index

| Test ID | T-32 | Category | Functional | Severity | |
|---|---|---|---|---|---|
| Objective | Verify that new embeddings are successfully added to the similarity index (FAISS) making them retrievable in search. | | | | |
| Steps | 1. Ensure a new set of embeddings has just been stored<br>2. Trigger the "Update Similarity Index" job<br>3. Perform a region-of-interest (ROI) search from the frontend using a patch from the newly indexed WSI | | | | |
| Expected | • The indexing job completes successfully without errors<br>• The frontend search retrieves the newly indexed WSI patch in the top results<br>• This confirms the index mapping is perfectly synced with the database | | | | |
| Date-Result | | | | | |

T-33: Updating Similarity Index without Crashing

| Test ID | T-33 | Category | Functional | Severity | |
|---|---|---|---|---|---|
| Objective | Verify that an ongoing index update does not block or crash active search queries from pathologists | | | | |
| Steps | 1. Initiate a large "Update Similarity Index" job in the background<br>2. While the index is actively building/updating, execute multiple image retrieval searches from the frontend UI<br>3. Observe the frontend search results and response times | | | | |
| Expected | • The frontend searches execute successfully against the old index without throwing a timeout or connection error<br>• The background index update finishes seamlessly<br>• The system hot-swaps to the new index without dropping user traffic | | | | |
| Date-Result | | | | | |

T-34: Registering User with Valid Credentials

| Test ID | T-34 | Category | Functional | Severity | |
|---|---|---|---|---|---|
| Objective | Verify that a System Administrator can successfully register a new user with valid credentials. | | | | |
| Steps | 1. Log in using System Administrator credentials<br>2. Navigate to the "Admin" page<br>3. Click the “Register User” button<br>4. Enter the name of the user<br>5. Enter a valid, unique email address<br>6. Enter a secure password that meets system requirements<br>7. Click the "Create Account" button | | | | |
| Expected | • The system successfully creates the new user account<br>• A success message is displayed on the screen<br>• The new user is visible in the active users list<br>• The new user can successfully log into the system with the provided credentials | | | | |
| Date-Result | | | | | |

T-35: Registering User with Existing Email

| Test ID | T-35 | Category | Functional | Severity | |
|---|---|---|---|---|---|
| Objective | Verify that the system prevents the registration of a new user if the provided email address already exists in the database. | | | | |
| Steps | 1. Log in using System Administrator credentials<br>2. Navigate to the "Admin" page<br>3. Click the “Register User” button<br>4. Enter the name of the user<br>5. Enter an email address that is already registered to an existing user<br>6. Enter a valid password<br>7. Click the "Create Account" button | | | | |
| Expected | • The system rejects the registration attempt<br>• A validation error message is displayed<br>• No duplicate account is created in the database | | | | |
| Date-Result | | | | | |

T-36: Changing Password

| Test ID | T-36 | Category | Functional | Severity | |
|---|---|---|---|---|---|
| Objective | Verify that the System Administrator can successfully update the password for an existing user account. | | | | |
| Steps | 1. Log in using System Administrator credentials<br>2. Navigate to the "Admin" page<br>3. Click the "Change Password" button in one of the “Registered Users” table entries.<br>4. Enter a new, valid password and confirm it.<br>5. Click "Update Password"<br>6. Attempt to log in as that user with the old password, then the new password | | | | |
| Expected | • The system successfully updates the user's password<br>• Logging in with the old password fails<br>• Logging in with the new password succeeds<br>• If the user was currently logged in during the change, their active session is terminated | | | | |

| | |
|---|---|
| Date-Result | |

T-37: Granting Admin

| Test ID | T-37 | Category | Functional | Severity | |
|---|---|---|---|---|---|
| Objective | Verify that a System Administrator can successfully grant admin access to a standard user | | | | |
| Steps | 1. Log in using System Administrator credentials<br>2. Navigate to the "Admin" page<br>3. Locate a standard user in the "Registered Users" table<br>4. Click the "Grant Admin" button for that user<br>5. Log out of the current Admin account<br>6. Log in as the newly modified user and check for access to the "Admin" panel | | | | |
| Expected | • The system successfully updates the user's role to Administrator in the database<br>• The modified user can now view the "Admin" tab in the navigation menu<br>• The modified user can successfully access and interact with the administrative functions on the Admin page | | | | |
| Date-Result | | | | | |

T-38: Revoking Admin

| Test ID | T-38 | Category | Functional | Severity | |
|---|---|---|---|---|---|
| Objective | Verify that a System Administrator can successfully revoke admin access from an existing administrator account | | | | |
| Steps | 1. Log in using System Administrator credentials<br>2. Navigate to the "Admin" page<br>3. Locate another user who currently has Admin privileges in the "Registered Users" table<br>4. Click the "Revoke Admin" button for that user<br>5. Log out of the current Admin account<br>6. Log in as the modified user and attempt to access the "Admin" | | | | |

| | |
|---|---|
| | panel |
| Expected | • The system successfully reverts the user's role back to a standard user<br>• The "Admin" tab is no longer visible in the modified user's navigation menu<br>• Attempting to manually navigate to the Admin page URL redirects the user or displays an "Access Denied" message |
| Date-Result | |

T-39: Removing User

| Test ID | T-39 | Category | Functional | Severity | |
|---|---|---|---|---|---|
| Objective | Verify that the System Administrator can successfully remove an existing user from the system | | | | |
| Steps | 1. Log in using System Administrator credentials<br>2. Navigate to the "Admin" page<br>3. Click the "Delete User" button in one of the "Registered Users" table entries<br>4. Confirm the deletion prompt<br>5. Attempt to log in using the removed user's credentials | | | | |
| Expected | • The user account is successfully removed from the "Registered Users" list.<br>• The removed user's subsequent login attempts fail with an appropriate "Invalid credentials" message<br>• The user's previously created annotations and uploaded WSIs remain in the system, attributed to a deleted user, ensuring medical records are not lost | | | | |
| Date-Result | | | | | |

T-40: Logging Out

| Test ID | T-40 | Category | Functional | Severity | |
|---|---|---|---|---|---|
| Objective | Verify that a logged-in user can successfully log out, terminating their active session and restricting access to protected pages. | | | | |

| | |
|---|---|
| Steps | 1. Log into the application with valid credentials.<br>2. Click the "Log Out" button.<br>3. Attempt to navigate back to the previous protected page using the browser's "Back" button<br>4. Attempt to access a protected URL directly |
| Expected | • User is redirected to the Sign In page<br>• The authentication token/session is destroyed<br>• The user cannot access protected pages via the "Back" button.<br>• Direct navigation to a protected URL forces a redirect back to the Sign In page. |
| Date-Result | |

## 5.2 Non functional test cases

T-41: Multimodal Search Response Time

| | | | | | |
|---|---|---|---|---|---|
| Test ID | T-41 | Category | Non-Functional | Severity | |
| Objective | Verify that the system retrieves and renders multimodal search results within an acceptable clinical workflow timeframe | | | | |
| Steps | 1. Execute a complex multimodal query<br>2. Measure the round-trip time from the button click to the results rendering on screen | | | | |
| Expected | • The PLIP model computes the query embedding, searches the index, and returns results in under 7.0 seconds<br>• The UI remains responsive (no freezing) while waiting for the payload | | | | |
| Date-Result | | | | | |

T-42: WSI Viewer Client Rendering & Memory Management

| | | | | | |
|---|---|---|---|---|---|
| Test ID | T-42 | Category | Non-Functio | Severity | |

| | | | | | |
|---|---|---|---|---|---|
| | | | nal | | |
| Objective | Verify that deep zooming and rapid panning on a WSI maintain a smooth framerate without causing memory leaks in the browser | | | | |
| Steps | 1. Open a multi-gigabyte WSI in the viewer<br>2. Rapidly pan and zoom in/out continuously for 2 minutes<br>3. Monitor browser tab RAM usage and visual framerate | | | | |
| Expected | • Tile rendering occurs smoothly without severe visual tearing or stuttering<br>• Browser RAM usage stabilizes due to effective garbage collection of off-screen tiles | | | | |
| Date-Result | | | | | |

T-43: Backend Similarity Index Memory Efficiency

| Test ID | T-43 | Category | Non-Functio nal | Severity | |
|---|---|---|---|---|---|
| Objective | Verify that the backend Similarity Index (FAISS) efficiently manages RAM when holding a massive dataset of patch embeddings | | | | |
| Steps | 1. Load the backend vector database with 100,000+ computed patch embeddings<br>2. Monitor the active RAM utilization of the indexing service | | | | |
| Expected | • The indexing service memory footprint remains within the server's allocated limits and does not cause an Out-Of-Memory crash<br>• Search times do not degrade exponentially as the index grows | | | | |
| Date-Result | | | | | |

T-44: Concurrent Search Scalability (Load Testing)

| Test ID | T-44 | Category | Non-Functional | Severity | |
|---|---|---|---|---|---|
| Objective | Verify that PLIP model can handle multiple concurrent search requests from different pathologists without dropping connections | | | | |
| Steps | 1. Use a load-testing tool to simulate 20 users executing multimodal searches simultaneously<br>2. Monitor the API error rate and average response times | | | | |
| Expected | • The system successfully queues and processes all requests<br>• 0% of requests return a 500 Internal Server Error or timeout<br>• Response times may increase under load, but remain within defined acceptable limits | | | | |
| Date-Result | | | | | |

T-45: Background Processing Resource Isolation

| Test ID | T-45 | Category | Non-Functional | Severity | |
|---|---|---|---|---|---|
| Objective | Verify that intensive background tasks (like WSI patch extraction) do not degrade the performance of the frontend search features | | | | |
| Steps | 1. Initiate a massive batch upload of 10 WSIs to trigger heavy backend patch extraction and embedding computation<br>2. While extraction is running, have a user perform standard text and image searches on the frontend | | | | |
| Expected | • Frontend search requests execute normally without significant lag<br>• The system properly isolates resources between background workers and user-facing APIs | | | | |
| Date-Result | | | | | |

T-46: Automatic Session Timeout

| Test ID | T-46 | Category | Non-Functional | Severity | |
|---|---|---|---|---|---|
| Objective | Verify that the system enforces an automatic session timeout after a period of user inactivity, adhering to medical data security standards | | | | |
| Steps | 1. Log in as a Pathologist<br>2. Leave the application completely idle (no mouse clicks, scrolling, or API calls) for the defined timeout period (15 minutes)<br>3. Attempt to navigate to a new page or execute a search | | | | |
| Expected | • The user is automatically logged out and redirected to the login screen<br>• The active session token is invalidated on the backend<br>• A message explains that the session expired due to inactivity is displayed | | | | |
| Date-Result | | | | | |

T-47: Backend API Role-Based Access Control

| Test ID | T-47 | Category | Non-Functional | Severity | |
|---|---|---|---|---|---|
| Objective | Verify that standard users cannot access backend administrative APIs directly via URL or API manipulation | | | | |
| Steps | 1. Log in as a standard Pathologist.<br>2. Use a tool like Postman to attempt a direct API call to an admin-only endpoint | | | | |
| Expected | • The backend strictly rejects the request<br>• A 403 Forbidden or 401 Unauthorized HTTP status code is returned<br>• The action is logged as an unauthorized access attempt | | | | |
| Date-Result | | | | | |

T-48: Clinical Audit Logging

| Test ID | T-48 | Category | Non-Functional | Severity | |
|---|---|---|---|---|---|
| Objective | Verify that the system actively logs crucial user actions to maintain an audit trail for clinical accountability | | | | |
| Steps | 1. Log in, open a specific WSI, execute a multimodal search, and log out<br>2. Access the system's backend database or log management console | | | | |
| Expected | • Logs accurately record the user ID, timestamp, the ID of the WSI accessed, and the parameters of the executed search<br>• Passwords and sensitive personal data are strictly excluded from or hashed in the logs | | | | |
| Date-Result | | | | | |

T-49: UI Contrast for Annotations and Heatmaps

| Test ID | T-49 | Category | Non-Functional | Severity | |
|---|---|---|---|---|---|
| Objective | Verify that the explainability heatmaps and annotation colors remain clearly visible against standard H&E tissue stains | | | | |
| Steps | 1. Open a densely stained H&E WSI<br>2. Draw several annotations and toggle the similarity heatmap<br>3. Visually inspect the contrast and readability | | | | |
| Expected | • The default colors for annotations and heatmaps contrast sufficiently with the underlying tissue<br>• The pathologist does not have to strain to distinguish the highlighted regions from the physical cell structures | | | | |
| Date-Result | | | | | |

T-50: Long-Running Process UI Feedback

| Test ID | T-50 | Category | Non-Functional | Severity | |
|---|---|---|---|---|---|
| Objective | Verify that the system provides clear, continuous feedback during long-running processes like WSI uploads or index updates | | | | |
| Steps | 1. Initiate a 20GB WSI file upload<br>2. Observe the UI during the upload process | | | | |
| Expected | • A persistent, accurate progress bar or percentage indicator is displayed<br>• The user is not left wondering if the system has frozen | | | | |
| Date-Result | | | | | |

T-51: Viewer Network Interruption Recovery

| Test ID | T-51 | Category | Non-Functional | Severity | |
|---|---|---|---|---|---|
| Objective | Verify that the WSI viewer handles sudden network drops gracefully without crashing the browser tab | | | | |
| Steps | 1. Open a WSI in the viewer<br>2. Disable the device's internet connection<br>3. Attempt to pan to an unloaded region of the slide | | | | |
| Expected | • The application does not crash.<br>• A non-intrusive error message "Network connection lost, reconnecting..." appears<br>• The viewer successfully resumes loading tiles once the connection is restored without requiring a full page refresh | | | | |
| Date-Result | | | | | |

T-52: Background Worker Error Recovery (Corrupted File Handling)

| Test ID | T-52 | Category | Non-Functional | Severity | |
|---|---|---|---|---|---|
| Objective | Verify that a corrupted WSI file does not crash the entire backend extraction queue | | | | |
| Steps | 1. Upload a deliberately corrupted file disguised as a WSI<br>2. Upload a valid WSI immediately afterward<br>3. Monitor the background worker queue | | | | |
| Expected | • The background worker fails gracefully on the corrupted file, marking it as "Failed" in the database<br>• The worker does not permanently die; it successfully picks up and processes the next valid WSI in the queue | | | | |
| Date-Result | | | | | |

# 6. Consideration of Various Factors in Engineering Design

## 6.1 Constraints

PatchMatch operates at the intersection of artificial intelligence, medical imaging, and clinical practice. The following sections discuss how public health, safety, security, welfare, global, cultural, social, environmental, and economic factors were considered during both the analysis and design phases of the project.

**Public Health** Effect Level: 9

PatchMatch directly serves the public health domain by supporting pathologists in identifying and comparing tissue patterns. During the analysis phase, interviews with a pathologist confirmed that enabling faster case comparison could meaningfully accelerate the diagnostic workflow, reducing both the time required to reach a conclusion and the likelihood of ordering additional tests due to uncertainty. The decision to design PatchMatch strictly as a decision-support tool rather than an autonomous diagnostic system was also driven by public health considerations; an incorrect automated diagnosis could directly harm a patient. Explainable retrieval through heatmaps and similarity scores was designed into the system specifically so that clinicians can critically evaluate results rather than blindly accept them. The 99% uptime requirement during clinical hours was set because system unavailability during active diagnostic sessions could delay patient care.

**Safety** Effect Level: 9

Any automated diagnostic output is deliberately excluded from the system. PatchMatch retrieves similar cases and presents similarity scores, but it does not classify tissue or suggest diagnoses. This boundary was established during the analysis phase and is enforced throughout the design through the absence of any diagnostic label generation in the ML layer. Score stretching was designed partly for safety reasons as well: raw cosine similarities clustering between 0.2 and 0.7 could be misread as low confidence across the board, potentially causing a clinician to dismiss relevant results. Stretching these to an interpretable 0–1 range reduces the risk of misinterpretation. The system also displays clinical report information alongside retrieval results so that pathologists have full context when comparing cases, reducing the risk of visually similar but clinically dissimilar cases being treated as equivalent.

**Security** Effect Level: 10

Security had maximum effect on the design because the system handles sensitive patient medical data. JWT-based authentication with bcrypt password hashing was chosen to ensure that credentials are never stored or transmitted in plaintext. Role-based access control was implemented so that regular users can only access their own data and explicitly shared slides, while administrative functions are gated behind a separate admin flag. Patient data anonymization is a stated requirement for any data entering the system, and the note on clinical deployment explicitly addresses that in a hospital environment the storage layer must reside on institution-controlled servers rather than public cloud infrastructure. Audit logging is built into the backend to support traceability of data access, which is a compliance requirement under both GDPR and KVKK.

**Welfare** Effect Level: 7

The welfare of both pathologists and patients was considered during design. For pathologists, the usability requirements, 30-minute onboarding, no more than 5 interactions to retrieve results, immediate feedback within a few seconds, were designed to reduce cognitive load and prevent the system from adding friction to an already demanding workflow. The analytics and system status pages were included to give users visibility into system state, reducing anxiety about whether the system is functioning correctly. The anonymization requirement, the decision-support framing, and the explainability features all serve patient welfare by reducing the risk of harm from misuse or overreliance on the system.

**Global Factors** Effect Level: 6

Digital pathology is a globally practiced discipline and PatchMatch was designed with international deployment in mind. The dataset used during development is drawn from TCGA, a publicly available US-based cancer genomics archive, which introduces a known geographic and demographic bias. This was acknowledged as a limitation during the analysis phase and documented in the system. The compliance framework was designed to satisfy both GDPR for European deployments and KVKK for Turkish deployments, with the architecture being flexible enough to accommodate other regional data protection regimes. The use of open standards such as IIIF for tile serving and support for common WSI formats including SVS and NDPI ensures that the system is not locked to any particular scanner manufacturer or geographic market.

**Cultural Factors** Effect Level: 3

Cultural factors had a limited but non-zero effect on the design. The user interface terminology was aligned with standard pathological vocabulary used internationally, rather than region-specific clinical language, to support adoption across different institutional contexts. The system was designed to avoid making clinical recommendations that could conflict with locally established diagnostic protocols or medical culture. Beyond these interface and framing considerations, the core retrieval and ML pipeline is not materially affected by cultural factors.

**Social Factors** Effect Level: 7

PatchMatch has meaningful social implications in terms of access to diagnostic quality. One of the stated motivations of the project is that pathologists in under-resourced institutions could use the system to compare their cases against a curated reference archive, effectively extending the reach of expert knowledge. The collaborative features, slide sharing, annotations, and notes, were designed to support knowledge transfer between pathologists across institutions. The educational use case was also explicitly considered during analysis: medical students and trainees can use the system to explore visual patterns and build case familiarity without requiring access to a physical slide library. These social motivations influenced the decision to support multiple user roles and to design the interface to be learnable without formal training.

**Environmental Factors** Effect Level: 5

Environmental factors had a moderate indirect effect on design decisions. The embedding generation pipeline is computationally intensive, requiring GPU acceleration

and significant processing time for large slide collections. During the analysis phase, the decision to pre-compute all patch embeddings offline rather than on demand was motivated partly by the desire to avoid repeated energy-intensive inference at query time. Storing embeddings as compressed NumPy files and caching clustering results to disk with fingerprint-based invalidation also reduces unnecessary recomputation. In a large-scale clinical deployment, the energy consumption of running continuous GPU inference servers would be a legitimate environmental consideration, and the architecture's support for batch preprocessing rather than live inference keeps this footprint lower than an on-demand design would.

**Economic Factors** Effect Level: 8

Economic considerations shaped several design and technology choices. The system was built entirely on open-source components, FastAPI, React, PyTorch, FAISS, PostgreSQL, OpenSeadragon, to eliminate licensing costs and maximize accessibility for institutions with limited budgets. PLIP was chosen over proprietary vision-language models for the same reason. The use of pre-computed embeddings stored as flat files rather than a commercial vector database such as Pinecone eliminates ongoing subscription costs during development. The split storage strategy between PostgreSQL and flat files was partly an economic decision. The note on clinical deployment acknowledges that at production scale, infrastructure costs including GPU servers, storage, and backup systems would be substantial, and the modular architecture was designed to allow institutions to scale components independently rather than requiring wholesale infrastructure investment.

Table 1. Various factors in engineering design and their effects.

| Factor | Effect Level (0-10) | Effect |
|---|---|---|
| Public Health | 9 | PatchMatch directly supports clinical diagnostic workflows. Retrieval latency targets, uptime requirements during clinical hours, and the decision-support framing were all shaped by the need to avoid disrupting or endangering patient care. |
| Safety | 9 | The exclusion of automated diagnosis, the design of explainable similarity scores, and score stretching for interpretability were all driven by the need to prevent clinical misuse or overreliance on system outputs. |
| Security | 10 | Authentication architecture, role-based access |

| | | |
|---|---|---|
| | | control, bcrypt password hashing, audit logging, and the clinical deployment note requiring on-premise storage were all direct consequences of handling sensitive patient medical data. |
| Welfare | 7 | Usability requirements including 30-minute onboarding, 5-interaction retrieval, and one-second feedback were designed to protect pathologist workflow. Patient welfare was addressed through anonymization requirements and the decision-support framing. |
| Global Factors | 6 | Multi-regulation compliance covering both GDPR and KVKK, open format support for SVS and NDPI, and IIIF adoption were all motivated by the need to support international deployment without geographic lock-in. |
| Cultural Factors | 3 | The system uses internationally standardized pathology terminology in the interface. No further cultural localization was required given that English is the established language of medical communication globally. |
| Social Factors | 7 | Collaboration features, multi-role user support, and the educational use case were motivated by the goal of making expert-level case comparison accessible to pathologists, trainees, and researchers across institutions with varying resources. |
| Environmental Factors | 5 | Pre-computed embeddings, batch inference rather than on-demand inference, and fingerprint-based clustering cache were designed to reduce unnecessary computation and energy use. Environmental impact is limited but was considered in pipeline design. |
| Economic Factors | 8 | The open-source stack including FastAPI, React, PyTorch, FAISS, and PostgreSQL, flat file embedding storage, and modular architecture were all chosen to minimize infrastructure costs and make the system accessible to institutions with limited budgets. |

## 6.2 Standards

**Medical Software Development Standards**

- The system follows IEC 62304:2006/AMD1:2015 [8] to ensure safe medical software development. As a decision-support system, PatchMatch aligns with the principles of safety class-based development and traceability.
- The system follows IEC 82304-1:2016 [9], which focuses on usability, validation, and release practices for health software.

**Medical Imaging and Data Exchange Standards**

- The system considers DICOM Supplement 145 [10] to support future interoperability with hospital PACS systems.
- The system is designed to be compatible with commonly used digital pathology formats including SVS and NDPI through the use of OpenSlide, enabling integration with existing clinical infrastructures.
- HL7 standards [11] are considered for the exchange of clinical metadata and pathology reports when integrating with hospital information systems.
- The WSI tile serving component is implemented using an IIIF-compatible interface [12], which is an open standard for serving large image collections in an interoperable and resolution-independent manner. This ensures that the viewer layer can be integrated with other IIIF-compliant platforms in the future.

**Quality and Risk Management Standards**

- ISO 13485:2016 [13] is considered for development practices as it emphasizes quality management, documentation, and controlled change processes in medical software.
- ISO 14971:2019 [14] guides the management of risks such as incorrect retrieval results, misinterpretation of similarity scores, and overreliance on system outputs in clinical decision-making.

**Regulatory and Compliance Considerations**

- The system is reviewed in line with the EU Medical Device Regulation (EU) 2017/745 [15], which covers clinical decision-support software.
- The EU Artificial Intelligence Act (EU) 2024/1689 [16] is considered to promote transparent and responsible use of AI in medical settings. PatchMatch's explainability features, score transparency, and decision-support framing directly address the transparency obligations outlined in this regulation.
- Regulatory pathways in other regions, such as the FDA Class II 510(k) process [17] in the United States, are taken into account for future deployment.

**Data Protection and Privacy Standards**

- PatchMatch follows GDPR (EU) 2016/679 [18] and KVKK Law No. 6698 [19] requirements to process medical images and patient data lawfully.
- To protect patient data, the system implements JWT-based authenticated access, role-based authorization, bcrypt password hashing, and audit logging. Anonymization is required for any data used in research or educational contexts.

**Documentation and Modeling Standards**

- UML 2.5.1 [20] is used for modeling system architecture, subsystem interactions, navigational flows, and dynamic behavior throughout this report.
- IEEE 830-1998 [21] principles guide the structure and clarity of the software requirements specification, with functional and non-functional requirements clearly separated and written in a testable and verifiable form.

**Software Engineering and API Standards**

- The backend API is structured following REST architectural principles [22], with clearly separated endpoints, stateless request handling, and standard HTTP status codes, ensuring predictable and interoperable client-server communication.
- OpenAPI Specification 3.0 [23] is used to document the backend API, enabling automatic generation of interactive API documentation and supporting future integration with external systems or hospital middleware.
- The frontend is built following React component architecture conventions and TypeScript strict typing [24], which enforces interface contracts between components and reduces the risk of integration errors during development.
- Docker and Docker Compose [25] are used following containerization best practices to ensure reproducible builds and consistent deployment environments across development, testing, and production targets.

# 7. Teamwork Details

## 7.1 Contributing and functioning effectively on the team

The PatchMatch team approached task distribution by mapping each member's existing technical strengths and areas of interest to the specific demands of the project, ensuring that no member worked in isolation and that every component received cross-team input.

- Weekly group meetings were conducted and important decisions were taken after consulting all members of the project.
- Project reports and presentations were prepared with a fairly distributed workload among every group member.
- Research and analysis of existing digital pathology systems, academic literature on WSI retrieval, and state-of-the-art deep learning methods were conducted collectively by all members during the early phases of the project.
- The project demo was presented with the attendance of each group member.
- The ML pipeline including patch extraction, PLIP-based embedding generation, FAISS indexing, score fusion, and multimodal retrieval was designed and implemented by Ekin Köylü and Emre Yazıcıoğlu with contributions from all members.
- The frontend including the WSI viewer, search interface, filter panel, analytics page, and result visualization was led by Elif Lara Oğuzhan with contributions from İlke Latifoğlu and Emre Yazıcıoğlu.
- The backend including the API gateway, authentication service, WSI tile server, slide sharing, annotations, and ML pipeline integration was led by Emre Yazıcıoğlu with contributions from Elif Lara Oğuzhan, İlke Latifoğlu, and Bertan Uran.
- The database layer including schema design, embedding storage, metadata management, and data access controls was implemented by Bertan Uran with contributions from İlke Latifoğlu.
- System testing including functional, integration, performance, and scalability testing will be led by İlke Latifoğlu with contributions from Emre Yazıcıoğlu.
- Demo preparation, system architecture presentations, and project launch activities were coordinated by İlke Latifoğlu and Bertan Uran respectively, with the participation of all members.

## 7.2 Helping creating a collaborative and inclusive environment

Building a collaborative and inclusive environment was a conscious priority throughout the project. The following practices were adopted to support this:

- Goals and milestones were defined collectively at the start of each project phase, with workload estimates reviewed by all members to ensure that no single person carried a disproportionate share of the effort. This upfront alignment reduced the risk of bottlenecks and kept expectations realistic across the team.
- Decision-making was treated as a shared responsibility. Technical choices such as the selection of the PLIP model, the flat file storage strategy were discussed

openly in team meetings rather than being decided unilaterally, ensuring that all members understood and could defend the rationale behind key design decisions.

- The team maintained a multi-channel communication structure suited to different types of interaction. WhatsApp was used for quick coordination and daily updates, Zoom for structured meetings and supervisor sessions, GitHub for code reviews and version tracking, and Google Drive for collaborative document editing and report drafting.
- Weekly internal meetings served as a consistent checkpoint where each member reported progress, flagged blockers, and aligned on the next steps.
- Supervisor meetings held at most every two weeks provided an external perspective on the project's direction.
- Meetings with a practicing pathologist were also organized to ground design decisions in real clinical needs, and all team members were expected to attend and engage rather than delegating this responsibility to a single representative.

## 7.3 Taking lead role and sharing leadership on the team

Leadership in PatchMatch was distributed across the team by work package, with each member owning a distinct technical domain while remaining accountable to the broader system. This structure meant that leadership shifted naturally depending on which phase of the project was active, rather than being concentrated in a single person throughout.

- Ekin Köylü anchored the research and ML pipeline work. Beyond conducting the literature review and benchmarking existing WSI retrieval systems, Ekin was responsible for translating academic findings into implementable decisions, selecting PLIP over generic vision-language models, designing the patch extraction strategy, and building the FAISS-based similarity search, score fusion.
- Elif Lara Oğuzhan drove the frontend from initial mockups through to the final implementation. Her ownership extended beyond writing components, she defined the navigational structure of the application, maintained visual and interaction consistency across all pages, and served as the primary point of contact for resolving mismatches between what the interface expected and what the backend provided.
- Emre Yazıcıoğlu took responsibility for the backend as a whole, which in practice meant coordinating the most cross-cutting part of the system. He directed the implementation of the API gateway, authentication, tile serving, ML integration, and sharing service, and was the person who ensured that changes in one backend module did not silently break another.

- Bertan Uran led both the database layer and the project launch phase, two responsibilities that required different kinds of leadership. The database work demanded careful technical design around schema structure and embedding storage, while the launch phase required coordination across all members to prepare deployment, onboarding materials, and user documentation.
- İlke Latifoğlu led testing and the demo phase, which placed her in a cross-cutting role during the final months of the project. Verifying that all subsystems worked correctly together required deep familiarity with every part of the system, and she was also responsible for ensuring the demonstration environment was stable and that the team's presentation materials accurately reflected what had been built.

## 8. Glossary

- **Patch:** A small fixed-size image crop extracted from a WSI, used as the unit of analysis for feature extraction and retrieval.
- **Multi-Resolution Embedding:** Feature representations computed at multiple magnification levels to capture both cellular-level and tissue-level visual characteristics.
- **Score Fusion:** The process of combining image similarity, text similarity, and clinical report similarity scores into a single ranked result using configurable weights.
- **Score Stretching:** A post-processing step that re-maps raw cosine similarity scores from a narrow range into a full 0–1 scale to make results interpretable for end users.
- **Region Aggregation:** The process of grouping spatially adjacent patch-level retrieval results into coherent tissue regions, producing slide-level retrieval results from patch-level scores.
- **Pattern Clustering:** The process of grouping patch embeddings using K-Means clustering and assigning human-readable pathology labels to each cluster via zero-shot classification.
- **Tile Serving:** The process of splitting a large WSI into small image tiles and delivering them to the viewer on demand, enabling smooth pan and zoom without loading the entire image into memory.
- **Tissue Masking:** A preprocessing step that identifies and excludes background, blurred, and non-tissue regions from a WSI before patch extraction, improving embedding quality.

- **FastAPI**: A modern Python framework for building APIs that handle requests between the frontend and backend services, like retrieval and embedding generation.
- **React**: A JavaScript library for building user interfaces that is used in this project to create the interactive image viewer, where users can navigate and annotate WSIs, and view retrieval results.
- **PostgreSQL**: An open-source relational database. Stores user accounts, WSI metadata, annotations, and session information for PatchMatch.
- **OpenSeadragon:** An open-source JavaScript library used to implement the interactive WSI viewer in PatchMatch, supporting smooth pan and zoom over large images.
- **Docker:** A containerization platform used to package and deploy the PatchMatch backend and frontend as isolated, reproducible containers.

# 9. References


1. Huron Digital Pathology, "Lagotto — content-based image retrieval solution for pathology," Huron Digital Pathology. [Online]. Available: https://www.hurondigitalpathology.com/lagotto/
2. Sectra, "Sectra Digital Pathology Solution," Sectra Medical Systems. [Online]. Available: https://medical.sectra.com/product/sectra-digital-pathology-solution/
3. Iron Mountain, "Digital Pathology Solutions," Iron Mountain. [Online]. Available: https://www.ironmountain.com/industries/healthcare-services/pathology-solutions
4. N. Hegde et al., "Similar image search for histopathology: SMILY," NPJ Digital Medicine, vol. 2, no. 1, p. 56, 2019.
5. S. Kalra, H. R. Tizhoosh, C. Choi, S. Shah, P. Diamandis, C. J. V. Campbell, and L. Pantanowitz, "Yottixel – An Image Search Engine for Large Archives of Histopathology Whole Slide Images," Medical Image Analysis, vol. 65, p. 101757, 2020. doi: 10.1016/j.media.2020.101757.
6. A. Pedersen, M. Valla, A. M. Bofin, J. P. de Frutos, I. Reinertsen, and E. Smistad, "FastPathology: An open-source platform for deep learning-based research and decision support in digital pathology," arXiv preprint arXiv:2011.06033, 2020.
7. Ibex Medical Analytics (Ibex), "Home – Ibex Medical Analytics," Ibex AI. [Online]. Available: https://ibex-ai.com/

8. IEC 62304:2006/AMD1:2015, *Medical device software — Software life cycle processes*, IEC, 2015.
9. IEC 82304-1:2016, *Health software — Part 1: General requirements for product safety*, IEC, 2016.
10. DICOM Standards Committee, *DICOM Supplement 145: Whole Slide Microscopic Image IOD and SOP Classes*, 2010.
11. Health Level Seven International, "HL7 Standards." [Online]. Available: https://www.hl7.org/
12. International Image Interoperability Framework, "IIIF Image API 3.0." [Online]. Available: https://iiif.io/api/image/3.0/
13. ISO 13485:2016, *Medical devices — Quality management systems — Requirements for regulatory purposes*, ISO, 2016.
14. ISO 14971:2019, *Medical devices — Application of risk management to medical devices*, ISO, 2019.
15. European Parliament and Council, *Regulation (EU) 2017/745 on medical devices (MDR)*, 2017.
16. European Parliament and Council, *Regulation (EU) 2024/1689 (Artificial Intelligence Act)*, 2024.
17. U.S. Food and Drug Administration, "Premarket Notification 510(k)." [Online]. Available: https://www.fda.gov/medical-devices/premarket-submissions
18. European Parliament and Council, *Regulation (EU) 2016/679 (General Data Protection Regulation — GDPR)*, 2016.
19. Kişisel Verileri Koruma Kanunu (KVKK), *Law No. 6698*, 2016. [Online]. Available: https://www.kvkk.gov.tr/
20. Object Management Group, *OMG Unified Modeling Language (UML), Version 2.5.1*, OMG, Dec. 2017. [Online]. Available: https://www.omg.org/spec/UML/2.5.1/
21. IEEE, *IEEE Recommended Practice for Software Requirements Specifications (IEEE Std 830-1998)*, IEEE Computer Society, 1998. [Online]. Available: https://standards.ieee.org/standard/830-1998.html
22. R. T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," Ph.D. dissertation, University of California, Irvine, 2000. [Online]. Available: https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

23. OpenAPI Initiative, *OpenAPI Specification Version 3.0.3*, 2020. [Online]. Available: https://spec.openapis.org/oas/v3.0.3
24. Microsoft, "TypeScript Documentation." [Online]. Available: https://www.typescriptlang.org/docs/
25. Docker Inc., "Docker Documentation." [Online]. Available: https://docs.docker.com/