Bilkent University
Department of Computer Engineering

Senior Design Project
T2504
Pathogenius

Detailed Design Report

Nazlı Apaydın 22202104
Ege Ateş 22201914
Yiğit Ali Doğan 22202329
Yunus Günay 22203758
Ata Uzay Kuzey 22203050

Supervisor: Can Alkan
Course Instructors: Mert Bıçakçı, İlker Burak Kurt

13.03.2026

Contents

1 Introduction

1.1 Purpose of the System

Rapid advances in sequencing technologies have enabled the analysis of genetic material obtained from clinical and environmental samples. Despite this progress, the software ecosystems required to interpret sequencing data remain complex, computationally demanding, and often dependent on high-performance computing infrastructure or continuous cloud connectivity. These requirements pose a significant barrier in field settings, emergency response scenarios, and resource-limited environments, where limited access to metagenomic analysis tools can hinder timely pathogen identification and critical decision-making.

Pathogenius addresses this challenge by providing a portable, offline-capable metagenomic analysis platform designed to operate on modest local hardware such as consumer-grade laptops. The system transforms raw long-read sequencing data into clear species-level identification results through a structured and automated analysis workflow. The classification stage utilizes the CLARK family of discriminative k-mer-based classifiers. When compatible NVIDIA GPUs are available, the pipeline leverages CuCLARK to accelerate classification. In environments where GPU resources are unavailable, the system automatically falls back to CLARK-lite, a memory-efficient CPU-based variant designed for operation on standard hardware. Both approaches rely on locally stored reference databases, enabling fast and reproducible taxonomic assignment without requiring external services or cloud connectivity.

To ensure accessibility for non-expert users, Pathogenius provides a locally running Electron.js-based graphical interface that abstracts complex command-line operations and offers real time feedback on system resources and workflow progress. A key innovation of the platform is the integration of a local AI assistant, implemented using small or quantized language models, which translates technical analysis results into readable and actionable summaries while preserving full offline operation. This design ensures that sensitive sequencing data remains transient and securely contained within the local environment.

1.2 Design Goals

Usability

- The system must provide a fully graphical user interface (GUI) implemented with Electron.js so that field personnel can perform all operations without using command-line tools.
- The interface must provide continuous visual feedback during long-running analyses through dynamic progress bars, workflow stage indicators, and status updates.
- The results dashboard should present classification outputs in a clear format, distinguishing high-confidence and low-confidence pathogen identifications using color-coding and visual markers.
- Navigation between major system components (analysis creation, monitoring, results visualization, and database management) should be intuitive and require minimal training for non-technical users.

Reliability

- The system must be offline-capable, allowing preprocessing, classification, and AI summarization functionality even when internet connectivity is unavailable.
- Required bioinformatics tools, reference databases, and software dependencies must be packaged with the deployment to ensure autonomous operation when needed.
- The system must ensure deterministic execution, producing identical analysis results when identical input data and reference databases are used.
- Raw sequencing files must be treated as read-only inputs.

Performance

- The analysis engine must be optimized to run on mid-range consumer laptops, avoiding the need for high-performance computing infrastructure.
- The system should efficiently utilize available hardware resources, including multi-core CPUs and GPU acceleration where available.

Supportability

- The system architecture should follow a modular design, separating the Electron GUI, workflow orchestration, and bioinformatics components to allow independent updates.
- The analysis pipeline should be implemented using Snakemake rules, enabling bioinformatics tools or workflow steps to be updated without modifying the entire system.
- The codebase should be well documented and structured, allowing new developers to easily understand, maintain, and extend the project.
- The system must generate persistent, human-readable session logs including timestamps, parameters, and hardware utilization to facilitate debugging and troubleshooting.

Marketability

- Pathogenius addresses an important need for portable and offline-capable pathogen detection, making it suitable for field laboratories and emergency response environments.
- The system provides a unique combination of bioinformatics analysis and AI-generated explanations, improving the interpretability of metagenomic results.
- The ability to run on consumer-grade laptops with both online and offline operation modes makes the system attractive for organizations operating in constrained environments.

Extendibility

- The system must be designed to support future expansion without requiring major architectural changes.
- Components such as workflow coordination, classification logic, reference data management, visualization, and AI interpretation should be separated into distinct subsystems.
- The modular structure should allow new bioinformatics tools, databases, and workflow stages to be added incrementally without disrupting existing functionality.
- The system should allow future extension of the reference database in order to support integration of new pathogen datasets and updated genomic resources.

Security

- The system must ensure secure handling of sequencing data, protecting sensitive biological information processed during analyses.
- All analysis operations should be designed to operate locally by default, preventing unintended data transmission to external services.

Scalability

- The system must support analysis of FASTQ datasets ranging from small samples to large sequencing datasets.
- The system should allow expansion of the reference database by importing additional FASTA files to support the detection of new or emerging pathogens.
- The application should automatically detect available hardware resources, including RAM and GPU capabilities, to optimize computational performance.

Maintainability

- The system must be supported by a modular architecture with clear separation of concerns.
- The backend workflow should be structured using independent Snakemake rules, allowing preprocessing, classification, and reporting stages to be modified without affecting the entire pipeline.
- The frontend should organize components such as dashboards, results visualization, database management, and system settings as separate functional units.
- The project should follow standard development practices, such as code documentation, version control, and consistent coding conventions to simplify maintenance.

Modularity

- Pathogenius should be designed as a set of distinct but cooperating architectural layers.
- These layers include sequencing input handling, reference database management, workflow execution, output analysis, and frontend presentation.
- Each layer should have a clearly defined responsibility.
- Communication between components should occur through structured data interfaces, allowing modules to be replaced or updated independently.

Aesthetics

- The interface design should prioritize clarity and usability for scientific and operational contexts.
- A clean and structured layout should support rapid interpretation of analysis results.
- Visual elements such as color coding, charts, cards, and layouts should improve readability and highlight important information.
- Critical information, including workflow status, confidence levels, and detected pathogens, should be visually emphasized.
- The interface should support decision-making in time-sensitive or stressful situations.

1.3 Definitions, Acronyms, and Abbreviations

- **AI:** Artificial Intelligence; computational methods used to generate human-readable interpretations of analysis results.
- **CLARK:** A discriminative k-mer based taxonomic classification tool used for identifying organisms from sequencing reads.
- **CLARK-lite:** A memory-efficient CPU-based variant of CLARK designed to operate on standard hardware with lower resource requirements.
- **CuCLARK:** A GPU-accelerated implementation of CLARK that uses NVIDIA CUDA to speed up metagenomic classification.
- **Electron.js:** A framework for building cross-platform desktop applications using web technologies such as JavaScript, HTML, and CSS.
- **FASTA:** A text-based file format used to represent nucleotide or protein sequences, where each sequence is preceded by a descriptive header line.
- **FASTQ:** A text-based file format used for storing biological sequence data along with corresponding quality scores.
- **GPU:** Graphics Processing Unit; a hardware accelerator capable of performing parallel computations, used in Pathogenius to accelerate classification tasks.
- **GUI:** Graphical User Interface; the visual interface through which users interact with the system.
- **NCBI:** National Center for Biotechnology Information; a source of genomic reference data used for building local databases.
- **k-mer:** A substring of length *k* extracted from a DNA sequence, commonly used in bioinformatics for sequence comparison and classification.
- **Metagenomics:** The study of genetic material recovered directly from environmental or clinical samples without isolating individual organisms.
- **PCR:** Polymerase Chain Reaction; a molecular biology technique used to amplify specific DNA segments.
- **Snakemake:** A workflow management system used to automate and coordinate the steps of the analysis pipeline.
- **Taxonomic Classification:** The process of assigning sequencing reads to known organisms based on reference genome databases.

1.4 Overview

Pathogenius is a portable, offline-capable metagenomic analysis platform designed to help users identify pathogens from long-read sequencing data on modest local hardware. The system combines an Electron.js-based graphical user interface, a Snakemake-managed analysis workflow, local reference databases, and optional GPU acceleration to transform raw FASTQ files into species-level classification results in a clear and accessible manner.

By integrating preprocessing, taxonomic classification, result visualization, and local AI-generated interpretation within a modular architecture, Pathogenius reduces the complexity of existing bioinformatics pipelines. This approach makes sequencing-based pathogen analysis more accessible to non-expert users in field laboratories, emergency response environments, and resource-limited settings, while ensuring that sensitive sequencing data remains entirely within the local system environment.

2 Current Software Architecture

Existing pathogen detection systems typically rely on architectures optimized for laboratory environments rather than portable or field-deployable workflows. Traditional PCR-based solutions such as RT-qPCR and ddPCR systems follow a tightly coupled laboratory pipeline in which sample preparation, amplification, and target detection are integrated with specialized hardware and reagent-dependent protocols [1]. As a result, these systems lack flexibility for discovering previously unknown pathogens and cannot dynamically adapt to new genomic targets without redesigning the assay.

Commercial pathogen detection platforms such as Bio-Rad's iQ-Check tests [2] and similar RT-qPCR kits follow this target-driven architecture as well. Their software components typically manage instrument configuration, amplification cycles, and result interpretation based on threshold values. While these systems can produce results within a relatively short time, they depend heavily on laboratory infrastructure and predefined pathogen panels. This architectural model prioritizes diagnostic reliability but sacrifices adaptability and portability, making it less suitable for exploratory pathogen detection scenarios.

More advanced systems such as Nanometa Live [3] represent an architectural shift toward real-time metagenomic analysis and visualization. These platforms integrate sequencing data streams with analysis dashboards that display classification results as sequencing progresses. However, their workflows still depend on complex toolchains and command-line driven execution. Many such systems rely on alignment-based methods such as BLAST for validation, which can introduce computational latency when operating on modest hardware. Additionally, these platforms primarily function as monitoring dashboards rather than fully integrated decision-support systems, requiring users to manually interpret complex analytical outputs.

Recent academic pipelines built around portable long-read sequencing platforms, particularly those based on Oxford Nanopore Technologies' MinION device [4], have demonstrated that on-site metagenomic pathogen detection is technically feasible using locally curated reference databases. However, these pipelines remain largely research-oriented. Their architectures typically consist of manually executed command-line workflows, locally managed reference databases, and loosely integrated analysis scripts. These solutions require significant bioinformatics expertise for workflow execution, database maintenance, and interpretation of results, limiting their accessibility in operational environments.

3 Proposed Software Architecture

3.1 Overview

The Pathogenius system is built using a 4-layer modular architecture designed to transform raw long-read sequencing data (FASTQ files) into actionable, species-level pathogen identification results through an automated analysis workflow. The architecture separates user interaction, workflow orchestration, computational analysis, data management, and optional hardware acceleration into distinct layers. This structure improves maintainability and scalability while enabling offline-capable operation on consumer-grade hardware and portable embedded platforms such as the NVIDIA Jetson Nano.

At the top of the architecture, the Frontend Layer provides an intuitive Electron.js-based graphical interface through which users can log in, upload FASTQ files, configure analysis parameters, monitor workflow progress in real time, and interpret pathogen identification results through interactive visualizations. This layer abstracts command-line complexity, allowing non-expert field personnel to operate the system without requiring bioinformatics expertise.

The Backend Layer acts as the central orchestrator of the system. It handles API requests from the frontend, manages authentication and user sessions, coordinates task delegation to the analysis pipeline, and monitors system resources. During system initialization, this layer also determines whether NVIDIA CUDA-compatible GPUs are available and configures the pipeline execution accordingly to enable optional hardware acceleration.

The Workflow Layer is the computational core of the platform. It utilizes the Snakemake workflow engine to coordinate a reproducible analysis pipeline consisting of preprocessing and quality control, k-mer based taxonomic classification using CuClark and CLARK-lite, and result aggregation with confidence scoring against a locally stored reference database. The basecaller and signal generator will also be implemented within the workflows to generate signals, which will then be passed to the basecaller to produce FASTQ files in separate workflows. This is intended to simulate a real field environment, where FASTQ files are not readily provided but are instead generated from the samples themselves through the signal generator, with the resulting signals fed into the basecaller to produce the FASTQ files. Each pipeline stage is implemented as an independent modular rule, allowing individual components to be updated, replaced, or scaled without affecting the rest of the system.

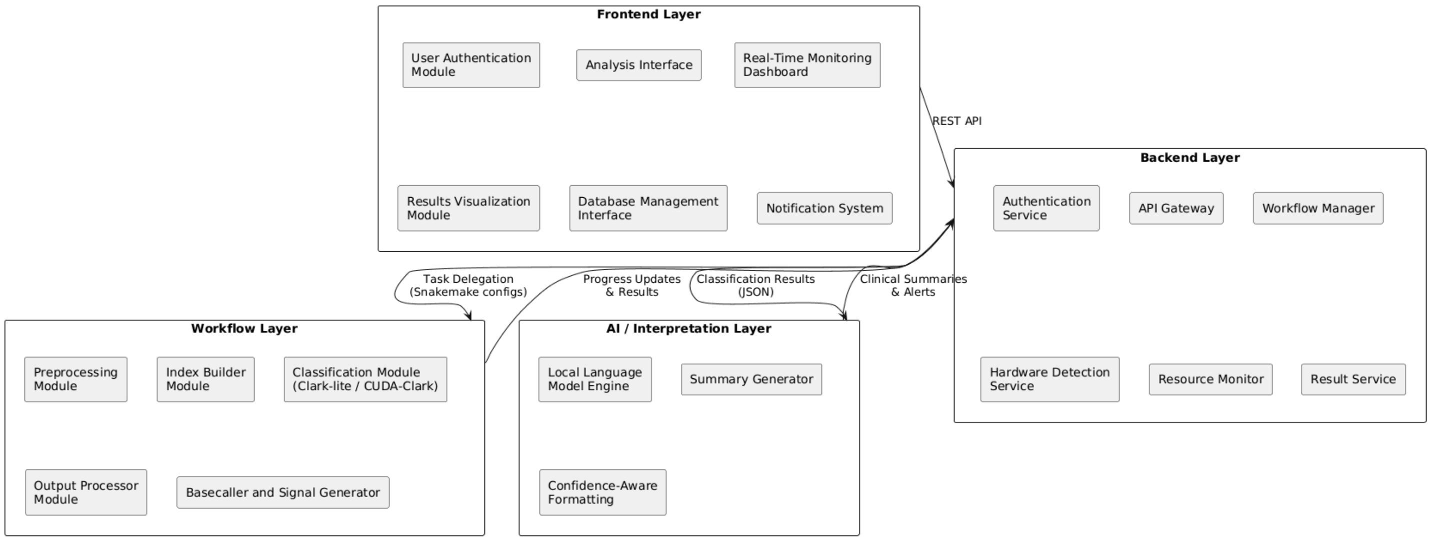
The AI / Interpretation Layer consumes the structured outputs produced by the analysis pipeline and generates human-readable summaries of pathogen detection results using a locally executed small or quantized language model. This layer is architecturally distinct from the workflow layer because it relies on different dependencies (model weights and inference engines) and has a different computational profile, being primarily memory-bound rather than I/O bound. All inference is performed locally to ensure that no analysis data leaves the device, preserving complete data sovereignty and offline operation.

Supporting these layers is a Local Data Storage subsystem accessed by all components of the architecture. This layer maintains FASTQ input files, FASTA reference databases, genomes, classification indices, analyses history records, user credentials, and AI model weights. The system is designed to operate entirely on local storage, without requiring external databases or cloud services. Cloud synchronization of analysis storage is completely optional.

Together, these four layers (Frontend, Backend, Workflow, and AI Interpretation) along with their supporting systems, form a cohesive architecture that supports the entire workflow, from FASTQ ingestion and quality-controlled classification to AI-powered result interpretation. This layered design ensures that the platform remains portable, reproducible, and robust for deployment in field and resource limited environments.

3.2 Subsystem Decomposition

The subsystem decomposition diagram (Figure 1) illustrates the modularity and data flow between the four layers of the Pathogenius system.



Frontend Layer
User Authentication Module
Analysis Interface
Real-Time Monitoring Dashboard
Results Visualization Module
Database Management Interface
Notification System
REST API
Backend Layer
Authentication Service
API Gateway
Workflow Manager
Hardware Detection Service
Resource Monitor
Result Service
Task Delegation (Snakemake configs)
Progress Updates & Results
Classification Results (JSON)
Clinical Summaries & Alerts
Workflow Layer
Preprocessing Module
Index Builder Module
Classification Module (Clark-lite / CUDA-Clark)
Output Processor Module
Basecaller and Signal Generator
AI / Interpretation Layer
Local Language Model Engine
Summary Generator
Confidence-Aware Formatting

Figure 1. *Modularity and data flow between the four layers of the Pathogenius.*

3.2.1 Frontend Layer

Purpose: The user facing interface that allows users to interact with the application. It enables FASTQ file uploads, analysis configuration, real-time workflow monitoring, result visualization with interactive charts, and user authentication, all without requiring any command line interaction.

Modules:

• **User Authentication Module:** Handles login, registration, guest mode access, and logout. Communicates with the backend's authentication service via REST API.

• **Analysis Interface:** Guides users through a three-step workflow: (1) FASTQ file selection via file dialog or drag-and-drop, (2) parameter configuration (analysis name, sample type, reference database, confidence threshold), and (3) review and launch.

• **Real-Time Monitoring Dashboard:** Displays dynamic progress bars with current Snakemake stage, completion percentage, system resource telemetry (CPU, RAM, GPU usage, storage), estimated time remaining, and reads-per-minute throughput.

• **Results Visualization Module:** Presents completed analysis results through species abundance treemaps, hierarchical taxonomy sunburst charts, read classification Sankey flow diagrams, pathogen detail cards with confidence scores and risk levels, and tabs for quality control metrics.

• **Database Management Interface:** Allows users to view current reference database information (version, species count, size, last update), import custom FASTA files, add/edit/remove species entries, and check for database updates when network is available.

• **Notification System:** Provides analysis completion banners, descriptive error messages, real-time progress indicators, and a persistent notification history.

Data Flow:

• **Frontend → Backend:** REST API calls for authentication, analysis initiation, result retrieval, and database operations.

• **Backend → Frontend:** API responses containing session tokens, analysis IDs, processing status updates, progress percentages, system resource metrics, and structured result data (JSON).

3.2.2 Backend Layer

Purpose: Acts as the central orchestrator of the system, handling all API requests from the frontend, delegating tasks to the pipeline and AI layers, managing user sessions, performing hardware detection, and coordinating data storage operations.

Modules:

• **Authentication Service:** Verifies user credentials against encrypted local storage, creates session tokens (containing user ID and role), manages role-based permissions (Admin, Registered User, Guest), and handles password reset flows.

• **API Gateway:** Serves as the single entry point for all frontend requests, routing them to the appropriate backend services and enforcing authentication requirements.

• **Workflow Manager:** Generates Snakemake configuration files based on user-selected parameters, initiates pipeline execution, monitors progress, manages the analysis lifecycle (Created → Queued → Preprocessing → Classifying → Processing → Finalizing → Completed/Failed/Cancelled), and supports checkpoint/resume for interrupted analyses.

• **Hardware Detection Service:** At system initialization, automatically detects available CPU cores, total RAM, presence of NVIDIA CUDA-compatible GPUs and their VRAM capacity, and available disk space. This information determines whether the pipeline will use Clark-lite (CPU) or CUDA-Clark (GPU) for classification, and is displayed on the dashboard.

• **Resource Monitor:** Tracks CPU, RAM, GPU, and storage utilization in real time during analysis. Enforces user-configurable memory caps to prevent exceeding physical RAM. Triggers warnings when resources fall below threshold levels.

• **Result Service:** Aggregates classification results and AI summaries, formats them for frontend consumption, and manages result export in PDF, JSON, and CSV formats.

Data Flow:

- **Backend → Pipeline Layer:** Sends Snakemake configuration (input files, parameters, selected database, classifier mode), receives progress updates and completion signals.
- **Backend → AI Layer:** Sends structured classification results for summarization, receives human-readable clinical summaries.

3.2.3 Workflow Layer

Purpose: The computational core of the system. Processes all bioinformatics tasks through a reproducible Snakemake workflow. Each stage operates as an independent, modular rule that can be developed, tested, and updated without affecting other stages.

Modules:

- **Preprocessing Module:** Performs quality control on raw FASTQ input: filters low-quality reads, handles compressed formats (.gz), and generates preprocessing quality reports. Original FASTQ files are treated as strictly read-only throughout all stages.
- **Index Builder Module:** Constructs classifier-compatible k-mer indices from locally stored FASTA reference genomes. Generated indices are cached and reused across analyses, and are rebuilt only when the reference database is modified.
- **Classification Module (Clark-lite / CUDA-Clark):** The central classification stage, implemented as a single Snakemake rule with two execution paths. When no GPU is available or GPU mode is disabled by the user, the rule invokes Clark-lite for CPU-based k-mer classification using multi-threaded processing. When NVIDIA CUDA hardware is detected and enabled, the rule invokes CUDA-Clark for GPU-accelerated classification with batched processing to manage VRAM constraints. If CUDA-Clark fails at runtime, the rule falls back transparently to Clark-lite. Both paths produce identical output formats, so downstream modules are unaffected by the classifier choice. It should be noted that on Jetson Nano, CUDA-Clark operates within shared memory constraints and batch sizing is critical.
- **Output Processor Module:** Aggregates raw classification results, computes abundance statistics, total/classified/unclassified read counts, confidence scores for each species identification, taxonomy breakdowns (bacteria, viruses, archaea, etc.), and generates structured JSON and CSV result files.
- **Basecaller and Signal Generator Module:** Calls the signal generator to output raw signal data, the basecaller ingests the raw signal data and outputs the base sequence (FASTQ file). This step will be implemented as a separate part of the workflow to simulate realistic field conditions, where FASTQ files are not directly available but are generated from the raw sequencing signals obtained from samples originating from water and raw organic materials. These signals are then processed by the basecaller to produce FASTQ sequences.

Data Flow:

• **Backend → Pipeline:** Task configurations (input files, parameters, selected database, classifier mode).

• **Pipeline → Backend:** Progress updates (current stage, completion percentage), quality reports, structured classification results (JSON/CSV).

3.2.4 AI / Interpretation Layer

Purpose: Translates complex, structured classification output into readable, actionable clinical summaries for non-expert users. This layer is architecturally separate from the pipeline because it has fundamentally different dependencies (language model weights, inference engine), different resource characteristics (memory-bound, not I/O-bound), and can be updated or swapped independently of the bioinformatics tools.

Modules:

• **Local Language Model Engine:** Executes a small or quantized open-source language model locally. All inference is performed on-device; no data is sent to external APIs or cloud services. When internet connectivity is available, the system optionally allows selecting or updating which local model is used, but inference always remains local.

• **Summary Generator:** Takes structured JSON classification results (species names, abundances, confidence scores, risk levels, taxonomy breakdowns) and produces human-readable summary paragraphs. Generates clinical recommendations where appropriate, with clear disclaimers that results are for decision support only.

• **Confidence-Aware Formatting:** Ensures that low-confidence findings are clearly distinguished in the generated text, and that uncertainty markers are preserved in all summaries to prevent over-interpretation by field users.

Data Flow:

• **Backend → AI Layer:** Structured classification results (JSON) from the pipeline, along with analysis metadata (sample type, database version).

• **AI Layer → Backend:** Human-readable clinical summary text, critical alerts (e.g., multi-drug resistant pathogens detected), and clinical recommendations.

3.3 Hardware/Software Mapping

Pathogenius is designed to run entirely on a single machine, targeting mid-range consumer-grade laptops. Unlike cloud based bioinformatics platforms, no external servers are required for core system operation. The hardware/software mapping describes how the four architectural layers are deployed onto available hardware resources.

Hardware Component	Software Deployed	Description
Host CPU	Electron.js Frontend, Backend API, Snakemake Engine, Clark-lite (CPU classifier), preprocessing tools, local AI model inference	Primary compute resource. The system detects available CPU cores at startup and dynamically scales thread usage. All four architectural layers execute on the CPU when GPU acceleration is not available.
Host RAM	Classification index, Snakemake working memory, AI model weights	Clark-lite or CUDA-CLARK loads the k-mer classification index into RAM during classification. User-configurable memory limits ensure that the system does not exceed available physical memory.
NVIDIA GPU (optional)	CUDA-CLARK classifier	When available and enabled, CUDA-CLARK offloads k-mer classification tasks to the GPU to accelerate processing. Batched processing is used to manage VRAM constraints. GPU availability is automatically detected during system startup.

3.4 Persistent Data Management

Pathogenius does not require a custom filesystem or a dedicated database engine. All persistent data, including FASTQ input files, FASTA reference genomes, analysis results, and user credentials, are stored locally on the host machine. The system is designed to operate in a fully offline-capable environment, although optional cloud storage can be used for backing up or sharing analysis results if desired.

The sizes of stored data vary depending on the type of the file. FASTQ input files range from 10MB to 4GB, while the locally stored reference genome database may occupy 8-15 GB, depending on the number of organisms included in the reference set. Analysis outputs and reports are comparatively small, only requiring kilobytes to a few megabytes of storage, and are stored in structured formats such as JSON or CSV.

Therefore, Pathogenius does not require extraordinary amounts of data management or a dedicated database, and it does not rely on a custom filesystem or database system.

3.5 Access Control and Security

The access control framework of Pathogenius is designed with the protection of sensitive genomic data, including potential human host DNA from clinical samples, as a primary concern. Because the system operates locally on shared devices in field settings, the framework ensures data isolation between users while maintaining ease of access during emergency situations. The system defines three distinct user roles with different permissions and responsibilities.

Role	Permissions	Description
Admin	Full system access: create/view/delete all analyses, manage users, reset passwords, manage databases, configure system settings	System administrator responsible for device setup, user management, and database maintenance
Registered User	Create/view/delete own analyses, export results, optional cloud synchronization, manage personal settings, change own password	Authenticated user with persistent analysis history and isolated workspace
Guest User	Create and view analyses within the current session; read and analysis permissions only	Emergency offline access without authentication; session-scoped with no persistent history

As shown above, the system defines distinct roles for the use of Pathogenious, with each role assigned specific permissions and workspace access. Data protection is particularly important because the system may process sensitive information, including human host DNA when analyzing clinical samples.

Clinical samples that may contain human host DNA are processed transiently. Sensitive intermediate data is not stored beyond the duration of the analysis, and no genomic information is transmitted to external services. All AI summarization is performed by locally executed language models, and the generated summaries are derived solely from locally produced analysis results. User passwords are hashed using cryptographic algorithms before storage, and plaintext passwords are never stored.

Each authenticated user operates within a separate workspace with protected data boundaries, ensuring that analysis data from one user is not accessible to another on the same device. Guest users do not have persistent data storage and are limited to operations within their current session.

4 Subsystem Services

4.1 Frontend Subsystem

The Frontend Subsystem is a desktop application built using Electron.js and serves as the primary graphical interface through which users interact with Pathogenius. It enables users to upload FASTQ files, configure analyses, monitor workflow execution in real time, visualize pathogen identification results through interactive charts, and manage authentication, all without requiring command-line interaction.

The subsystem is composed of several functional modules. The User Authentication Module manages login, registration, and guest-mode access. The Analysis Interface guides users through file selection, parameter configuration, and the initiation of analysis workflows. The Real-Time Monitoring Dashboard provides live feedback on workflow progress, displaying progress bars, system telemetry, and estimated time remaining. The Results Visualization Module presents analysis outputs through interactive visualizations such as species abundance treemaps, taxonomy sunburst charts, Sankey flow diagrams, and pathogen detail cards.

Additionally, the subsystem includes a Database Management Interface, which allows users to view and modify the locally stored reference database; and a Notification System that provides completion banners, error alerts, and a persistent notification history. The architecture of the frontend subsystem is illustrated in Figure 2.

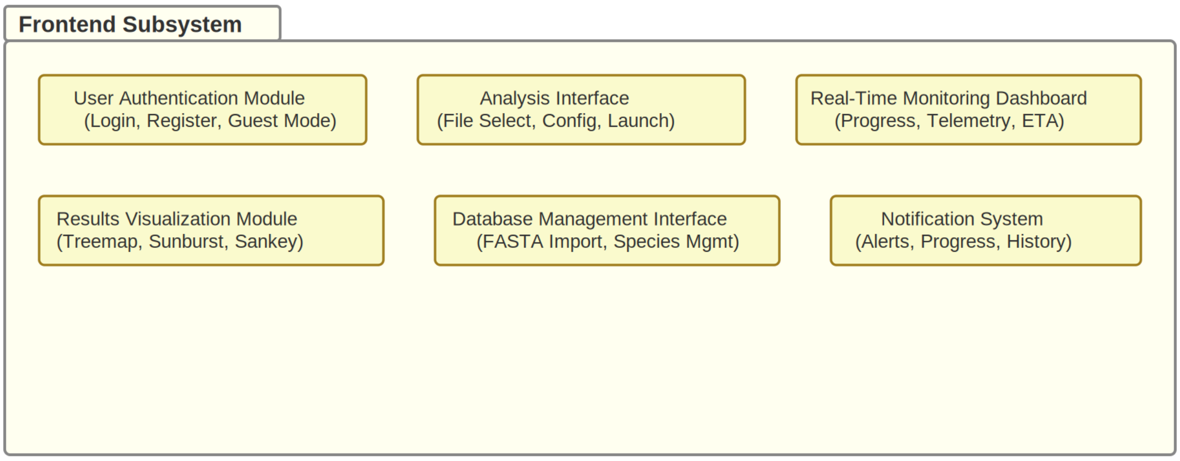

Frontend Subsystem
User Authentication Module (Login, Register, Guest Mode)
Analysis Interface (File Select, Config, Launch)
Real-Time Monitoring Dashboard (Progress, Telemetry, ETA)
Results Visualization Module (Treemap, Sunburst, Sankey)
Database Management Interface (FASTA Import, Species Mgmt)
Notification System (Alerts, Progress, History)

Figure 2. *Frontend Subsystem internal structure*

4.2 Backend Subsystem

The Backend Subsystem acts as the central orchestrator of the system. It handles API requests from the frontend, delegates tasks to the Workflow and AI subsystems, manages user sessions, performs hardware detection, and coordinates data storage operations. This subsystem serves as the control layer that connects the user interface with the computational analysis pipeline.

The Authentication Service verifies user credentials and manages role-based access control. The API Gateway serves as the single entry point for all frontend requests, routing them to the appropriate backend components. The Workflow Manager is responsible for generating Snakemake configurations, initiating pipeline execution, and managing the analysis lifecycle.

In addition, the backend contains a Hardware Detection Service, which probes the system during initialization to determine available CPU cores, RAM, NVIDIA CUDA-compatible GPUs, and VRAM capacity. Based on this detection, the system decides whether the classification pipeline will run using Clark-lite (CPU) or CUDA-CLARK (GPU). The Resource Monitor tracks system utilization in real time and enforces memory limits to prevent resource exhaustion. Finally, the Result Service aggregates classification results and AI-generated summaries, preparing them for visualization, export, and storage. The architecture of the backend subsystem is illustrated in Figure 3.

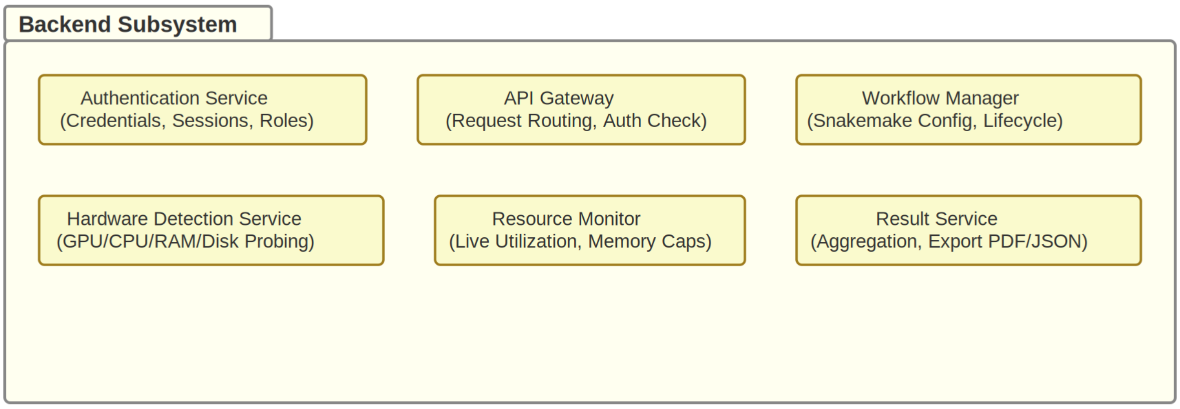

Backend Subsystem
Authentication Service (Credentials, Sessions, Roles)
API Gateway (Request Routing, Auth Check)
Workflow Manager (Snakemake Config, Lifecycle)
Hardware Detection Service (GPU/CPU/RAM/Disk Probing)
Resource Monitor (Live Utilization, Memory Caps)
Result Service (Aggregation, Export PDF/JSON)

Figure 3. *Backend Subsystem internal structure*

4.3 Workflow Subsystem

The Workflow Subsystem is the computational core of the system, executing all bioinformatics tasks through a reproducible Snakemake-based workflow in which each stage operates as an independent modular rule. The pipeline begins with the Signal Generator and Basecaller, which together simulate realistic sequencing conditions. The signal generator produces simulated sequencing signals, and the basecaller converts these signals into FASTQ-format reads within a separate workflow stage.

Following data generation, the Preprocessing Module performs quality filtering on the raw FASTQ input, handles compressed formats, and generates quality reports. During this stage, original FASTQ files are treated as strictly read-only. The Index Builder Module constructs classifier-compatible k-mer indices from locally stored FASTA reference genomes. These indices are cached after construction so they can be reused across multiple analyses. The Classification Module implements a single Snakemake rule with two execution paths. When no compatible GPU is available, the system performs CPU-based classification using Clark-lite. When NVIDIA CUDA-compatible hardware is detected, the pipeline switches to CUDA-CLARK for GPU-accelerated classification. If GPU execution fails at runtime, the system automatically falls back to the CPU-based Clark-lite classifier to ensure workflow completion.

Finally, the Output Processor Module aggregates the raw classification outputs, calculates abundance statistics and confidence scores, and generates structured JSON or CSV result files. The architecture of the workflow subsystem is illustrated in Figure 4.

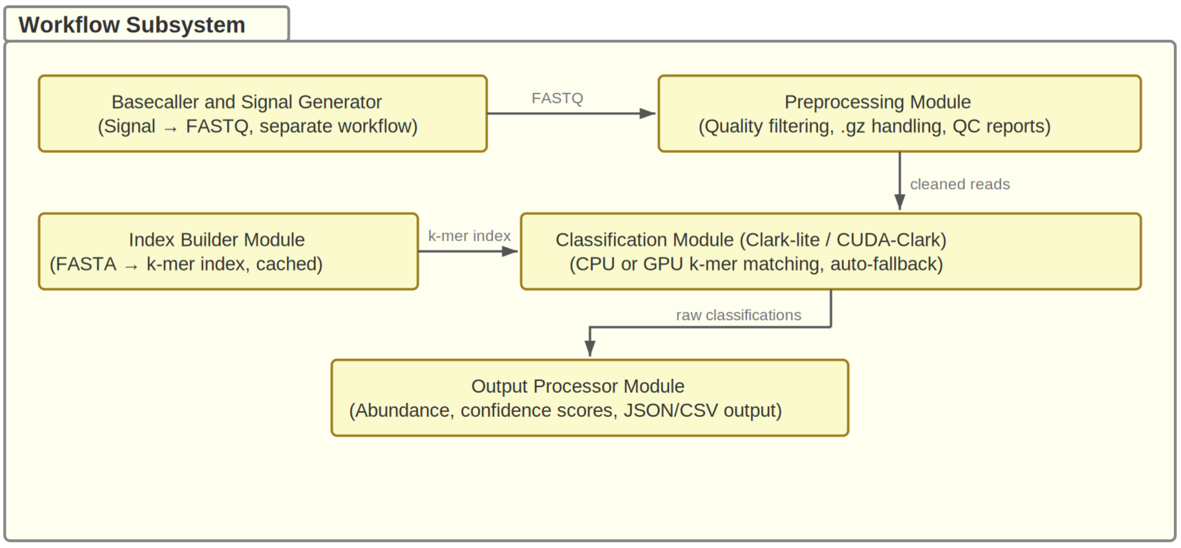

Workflow Subsystem
Basecaller and Signal Generator
(Signal → FASTQ, separate workflow)
FASTQ
Preprocessing Module
(Quality filtering, .gz handling, QC reports)
cleaned reads
Index Builder Module
(FASTA → k-mer index, cached)
k-mer index
Classification Module (Clark-lite / CUDA-Clark)
(CPU or GPU k-mer matching, auto-fallback)
raw classifications
Output Processor Module
(Abundance, confidence scores, JSON/CSV output)

Figure 4. *Workflow Subsystem internal structure*

4.4 AI / Interpretation Subsystem

The AI / Interpretation subsystem translates complex, structured classification outputs into readable and actionable clinical summaries for non-expert users. This subsystem is architecturally separate from the workflow layer because it has fundamentally different dependencies (language model weights and inference engines), different resource characteristics (being primarily memory-bound rather than I/O-bound), and can be updated or replaced independently of the bioinformatics tools.

The Local Language Model Engine executes a quantized open-source language model entirely on-device without transmitting any data to external services. It also includes the Summary Generator, which consumes structured JSON classification results and produces human-readable summary paragraphs containing clinical interpretations and decision-support disclaimers. Additionally, the Confidence-Aware Formatting module ensures that low-confidence findings are clearly distinguished and that uncertainty markers are preserved in all summaries to prevent over-interpretation by field users. The architecture of the AI / Interpretation subsystem is illustrated in Figure 5.

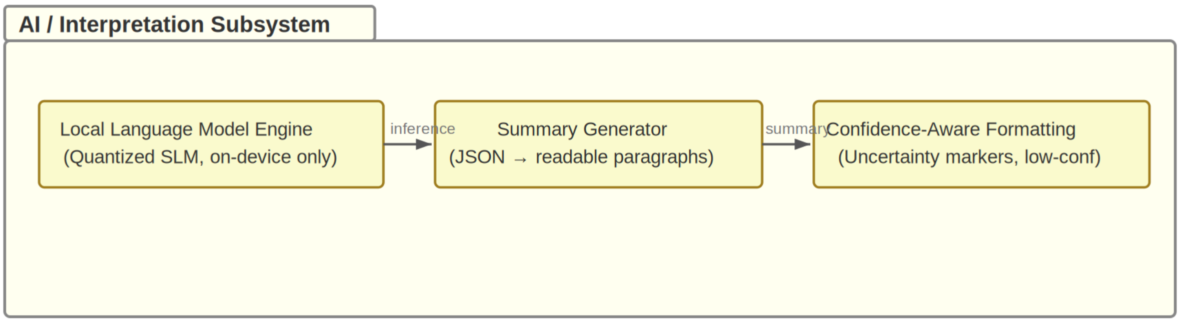

AI / Interpretation Subsystem
Local Language Model Engine
(Quantized SLM, on-device only)
inference
Summary Generator
(JSON → readable paragraphs)
summary
Confidence-Aware Formatting
(Uncertainty markers, low-conf)

Figure 5. *AI / Interpretation Subsystem internal structure*

5 Test Cases

The prefix **AM** represents *Analysis and Workflow Management*, **FP** refers to *FASTQ Processing and Classification*, **DM** represents *Dataset Management*, **NT** refers to *Notifications*, **UA** represents *User Authentication*, **RT** refers to *Realtime Data Display*, **GPU** represents *GPU Acceleration*, **USE** refers to *Usability*, **REL** represents *Reliability*, **PER** refers to *Performance*, **SUP** represents *Supportability*, and **SCA** refers to *Scalability*. These prefixes are used in test case identifiers to indicate the subsystem or requirement category that each test case validates.

Test ID	**TC-AM-001**
Category	Functional
Severity	Critical
Objective	Verify that a completed analysis is stored in the persistent local database for an authenticated user with analysis name, date, and status.
Steps	1. Log in as an authenticated user. 2. Navigate to the New Analysis page and start a new analysis with a valid FASTQ input file. 3. Enter a custom analysis name. 4. Allow the analysis to complete successfully. 5. Navigate to the Results page's Analysis History section. 6. Locate the completed analysis entry. 7. Restart the application and log in again with the same user account. 8. Navigate back to the Analysis History section.
Expected	• The completed analysis is stored and remains visible after application restart. • The stored entry includes the analysis name, date, and status. • The status is shown as "Completed." • The data is associated with the same authenticated user.
Date-Result	

Test ID	**TC-AM-002**
Category	Functional
Severity	High
Objective	Verify that the analysis history table displays analysis name, date, processing status, and its report for authenticated users.
Steps	1. Log in as an user with at least one completed analysis in the history. 2. Navigate to the Results page's Analysis History section. 3. Inspect the columns of the history table. 4. Compare the displayed data with the stored analysis details.
Expected	• The history table is displayed. • Each analysis row shows the analysis name, date, report, and processing status. • The displayed values match the stored analysis information.
Date-Result	

Test ID	TC-AM-003
Category	Functional
Severity	Medium
Objective	Verify that the system allows the user to assign a custom name and description to a new analysis.
Steps	1. Navigate to the New Analysis page. 2. Select a valid FASTQ input file. 3. Enter a custom analysis name and description. 4. Complete the remaining configuration steps and start the analysis. 5. After completion, navigate to the Results page's Analysis History section. 6. Open the saved analysis details.
Expected	• The system accepts the custom analysis name and description. • The analysis is saved with the entered name, stored, and displayed in the detailed analysis view. • No default or incorrect value replaces the user-provided inputs.
Date-Result	

Test ID	TC-AM-004
Category	Functional
Severity	Medium
Objective	Verify that the system prevents proceeding when the analysis name field is empty.
Steps	1. Navigate to the New Analysis page. 2. Select a valid FASTQ input file. 3. Leave the analysis name field empty. 4. Attempt to continue or start the analysis.
Expected	• The system displays a validation message indicating that the analysis name is required. • The user remains on the current step. • The analysis is not started until a valid name is entered.
Date-Result	

Test ID	TC-AM-005
Category	Functional
Severity	High
Objective	Verify that the system search and filter functionality allows users to locate analyses by analysis name, date, or detected pathogen.
Steps	1. Log in as an user with multiple analyses in the history table containing different names, dates, and detected pathogens. 2. Navigate to the Results page's Analysis History section. 3. In the search field, enter the exact or partial name of an existing analysis and observe the results. 4. Clear the search field and apply a date filter corresponding to one of the analyses. 5. Clear the filters and enter the number of detected pathogens into the search field. 6. Observe the displayed results for each filtering criterion.

Expected	• The table dynamically updates according to the entered filter criteria. • When filtering by analysis name, only analyses matching the entered name or partial name are displayed. • When filtering by date, only analyses corresponding to the specified date criterion are displayed. • When filtering by detected pathogens, only analyses containing the specified number of pathogens are displayed. • Analyses that do not match the applied criteria are hidden.
Date-Result	

Test ID	**TC-AM-006**
Category	Functional
Severity	High
Objective	Verify that a user can reopen a completed analysis and view its results.
Steps	1. Log in as an user with at least one completed analysis. 2. Navigate to the Results page's Analysis History section. 3. Locate a completed analysis entry. 4. Click the "View Report" action.
Expected	• The system opens the selected completed analysis. • The detailed results report is displayed. • Analysis summary, metrics, and result visualizations are shown correctly. • The opened report corresponds to the selected analysis.
Date-Result	

Test ID	**TC-AM-007**
Category	Functional
Severity	Low
Objective	Verify that the system allows deletion of an analysis only after confirmation.
Steps	1. Log in as an user with at least one completed analysis. 2. Navigate to the Results page's Analysis History page. 3. Locate an analysis entry. 4. Click the "Delete" button. 5. Observe the confirmation prompt. 6. Confirm the deletion.
Expected	• A confirmation dialog is displayed before deletion. • After confirmation, the selected analysis is removed from history. • The deleted analysis cannot be reopened from the history table. • The system updated the analysis list immediately.
Date-Result	

Test ID	**TC-AM-008**
Category	Functional

Severity	Medium
Objective	Verify that the system exports a completed analysis in PDF format.
Steps	1. Log in as an user with at least one completed analysis. 2. Navigate to the Results page, click on “View Reports,” and open the detailed results page of the analysis. 3. Click the “Export Report” button. 4. Choose a save location and confirm.
Expected	• The system generates a PDF file successfully. • The file is saved to the selected location. • The PDF contains the analysis results in readable form.
Date-Result	

Test ID	**TC-AM-09**
Category	Functional
Severity	High
Objective	Verify that the system generates a human-readable summary paragraph using the language model for a completed analysis.
Steps	1. Log in as an authenticated user. 2. Start and complete an analysis with valid input data. 3. Open the Results page and view the report of the completed analysis. 4. Locate the “Overview & Summary” section.
Expected	• A summary paragraph is displayed in readable natural language. • The summary elaborates on classification metrics and findings. • The summary is related to the selected analysis results.
Date-Result	

Test ID	**TC-AM-010**
Category	Functional
Severity	High
Objective	Verify that the offline AI summary is still available when the system is operated without network connectivity.
Steps	1. Disconnect the test machine from the internet. 2. Log in as an authenticated user. 3. Open a completed analysis with available result data from the Results page’s Analysis History section. 4. Access the AI summary from the “Overview & Summary” section.
Expected	• The AI-generated summary is displayed successfully in offline mode. • No network-required warning is shown for this feature. • The system does not depend on external APIs to produce the summary.
Date-Result	

Test ID	**TC-FP-001**
Category	Functional
Severity	High
Objective	Verify that the system accepts valid long-read FASTQ files in both compressed and uncompressed formats as input.
Steps	1. Log in as an authenticated user. 2. Navigate to the New Analysis page. 3. Select a valid uncompressed .fastq file and upload it. 4. Start the analysis process. 5. Repeat the process using a compressed .fastq.gz file.
Expected	• The system accepts both .fastq and .fastq.gz files. • The files are successfully uploaded without validation errors. • The analysis can proceed with both formats.
Date-Result	

Test ID	**TC-FP-002**
Category	Functional
Severity	Critical
Objective	Verify that the system rejects invalid input files that are not in FASTQ format.
Steps	1. Log in as an authenticated user. 2. Navigate to the New Analysis page. 3. Attempt to upload a file with an unsupported format. 4. Attempt to start the analysis.
Expected	• The system detects that the file format is invalid. • An error message indicating unsupported file type is displayed. • The analysis cannot proceed with the invalid file.
Date-Result	

Test ID	**TC-FP-003**
Category	Functional
Severity	Medium
Objective	Verify that the system validates uploaded FASTQ files and displays file metadata before processing.
Steps	1. Log in as an authenticated user. 2. Navigate to the New Analysis page. 3. Upload a valid FASTQ file. 4. Wait for file validation to complete. 5. Observe the displayed file metadata.
Expected	• The system validates the file structure successfully. • Metadata such as file name, file size, number of reads, and format is displayed.

	• The user can review the metadata before initiating processing.
Date-Result	

Test ID	**TC-FP-004**
Category	Functional
Severity	Critical
Objective	Verify that the system calculates and reports confidence scores for each species identification.
Steps	1. Navigate to the New Analysis page. 2. Upload a valid FASTQ file and start an analysis. 3. Allow the analysis to complete. 4. Navigate to the Results page and open the results by clicking the "View Report" button. 5. Inspect the classification results.
Expected	• Each identified species entry includes a confidence score. • The system clearly distinguishes between high-confidence and low-confidence classifications. • Confidence scores correspond to the classification results.
Date-Result	

Test ID	**TC-FP-005**
Category	Functional
Severity	Critical
Objective	Verify that the workflow engine coordinates preprocessing, classification, and output stages.
Steps	1. Navigate to the New Analysis page. 2. Upload a valid FASTQ file and start an analysis. 3. Monitor the workflow progress display. 4. Observe the execution of pipeline stages.
Expected	• The workflow executes preprocessing, classification, and output stages sequentially. • Each stage completes successfully before the next stage begins. • The workflow execution follows a deterministic order.
Date-Result	

Test ID	**TC-FP-006**
Category	Functional
Severity	Medium
Objective	Verify that interrupted analyses can resume from the last completed workflow stage.
Steps	1. Navigate to the New Analysis page. 2. Upload a valid FASTQ file and start an analysis. 3. During processing, simulate an interruption. 4. Restart the application.

	5. Reopen the interrupted analysis. 6. Resume the workflow.
Expected	• The system detects the previously completed workflow stages. • Processing resumes from the last successful stage rather than restarting the entire workflow. • The analysis eventually completes successfully.
Date-Result	

Test ID	**TC-FP-007**
Category	Functional
Severity	High
Objective	Verify that the system generates preprocessing and classification quality reports.
Steps	1. Navigate to the New Analysis page. 2. Upload a valid FASTQ file and start an analysis. 3. Allow the analysis to complete. 4. Navigate to the Results page. 5. Open the quality report section via the “View Report” button.
Expected	• A preprocessing quality report is generated. • A classification quality report is generated. • The reports contain relevant metrics and are accessible from the Results page.
Date-Result	

Test ID	**TC-FP-008**
Category	Functional
Severity	Medium
Objective	Verify that the system supports batch processing of multiple FASTQ files.
Steps	1. Navigate to the New Analysis page. 2. Upload a valid FASTQ file and start an analysis. 3. Start the batch analysis process. 4. Allow the analyses to complete. 5. Navigate to the Results page and inspect its Analysis History section.
Expected	• The system accepts multiple FASTQ files for a single batch run. • Each input file is processed through the analysis pipeline. • Results are generated and displayed for each file separately. • The results for all input files are accessible in the results interface.
Date-Result	

Test ID	**TC-DM-001**
Category	Functional

Severity	High
Objective	Verify that the system displays current reference database metadata including version, species count, last update date, and database size.
Steps	1. Navigate to the Database page via the sidebar. 2. Locate the “Current Reference Database” panel. 3. Read the displayed metadata fields: Version, Index Size, Total Genomes, Last Updated. 4. Verify each field is populated with a non-empty value.
Expected	• Database version number is displayed (e.g., 2025.01). • Index size is shown in GB. • Total genomes count is displayed. • The last updated date is shown in a recognizable format.
Date-Result	

Test ID	**TC-DM-002**
Category	Functional
Severity	High
Objective	Verify that a user can import a valid custom FASTA file and add a new species to the reference database.
Steps	1. Navigate to the Database page. 2. Scroll to the Custom Species section and click the “Add FASTA” button. 3. Select a valid .fasta file from the local filesystem. 4. Enter required metadata. 5. Confirm the submission.
Expected	• The system validates the FASTA format without errors. • The new species entry appears in the Custom Species list. • Species name, Tax ID, and type are correctly shown. • A success notification is displayed.
Date-Result	

Test ID	**TC-DM-003**
Category	Functional
Severity	Medium
Objective	Verify that the system prevents importing an invalid (non-FASTA) file and displays an appropriate error message.
Steps	1. Navigate to the Database page. 2. Click “Add FASTA” under the Custom Species section. 3. Select a non-FASTA file. 4. Attempt to proceed.
Expected	• The system displays an error message indicating the file format is invalid. • The invalid file is rejected and not added to the database. • The user remains on the same page and can select a new file.
Date-Result	

Test ID	TC-DM-004
Category	Functional
Severity	Medium
Objective	Verify that the system warns the user and provides options when attempting to add a species that already exists in the database.
Steps	1. Navigate to the Database page. 2. Click “Add FASTA” and select a FASTA file for a species already in the database. 3. Enter species metadata matching an existing entry. 4. Confirm submission.
Expected	• The system detects the duplicate and displays a warning message. • The user is presented with options to update the existing entry or cancel. • No duplicate entry is silently created.
Date-Result	

Test ID	TC-DM-005
Category	Functional
Severity	Medium
Objective	Verify that a user can remove a custom species from the reference database with confirmation.
Steps	1. Navigate to the Database page and scroll to the Custom Species section. 2. Locate an existing custom species entry. 3. Click the “Remove” button next to the species. 4. Confirm the removal in the confirmation dialog.
Expected	• The confirmation dialog is shown before deletion. • After confirmation, the species entry is removed from the Custom Species list. • A success notification is displayed. • The database index is updated to reflect the removal.
Date-Result	

Test ID	TC-DM-006
Category	Functional
Severity	Low
Objective	Verify that canceling a remove action leaves the species entry unchanged in the database.
Steps	1. Navigate to the Database page. 2. Click the “Remove” button next to a custom species entry. 3. When the confirmation dialog appears, click “Cancel.”
Expected	• The dialog closes without removing the species. • The species entry remains visible and unchanged in the Custom Species list.
Date-Result	

Test ID	TC-DM-007
Category	Functional
Severity	Medium
Objective	Verify that a user can edit taxonomic information of an existing custom species.
Steps	1. Navigate to the Database page and scroll to the Custom Species section. 2. Click the “Edit” button next to an existing custom species entry. 3. Modify the species name, taxonomic ID, or type fields in the edit dialog. 4. Confirm the changes.
Expected	• The updated information is saved and displayed correctly in the Custom Species list. • A success notification is shown. • No data loss occurs for unchanged fields.
Date-Result	

Test ID	TC-DM-008
Category	Functional
Severity	Medium
Objective	Verify that the system rejects an invalid taxonomic ID format during species edit and prompts the user to correct it.
Steps	1. Navigate to the Database section. 2. Click the “Edit” button next to a custom species. 3. Enter an invalid taxonomic ID (e.g., letters instead of numeric ID). 4. Attempt to confirm changes.
Expected	• The system displays a validation error message for the invalid Tax ID. • Changes are not saved. • The user can correct the value and resubmit.
Date-Result	

Test ID	TC-DM-009
Category	Functional
Severity	High
Objective	Verify that the system checks for database updates when the network is available and correctly reports the update status.
Steps	1. Ensure the system has an active internet connection. 2. Navigate to the Database page. 3. Click the “Check for Updates” button. 4. Observe the displayed update status.
Expected	• The system contacts the update server and retrieves version information. • If an update is available, the new version and size are shown. • If already up-to-date, a message such as “Database is up to date.” is shown. • If the network is unavailable, the system displays “Network connection required.”

Date-Result	

Test ID	**TC-NT-001**
Category	Functional
Severity	High
Objective	Verify that the system provides a visual notification when an analysis workflow completes successfully.
Steps	1. Navigate to the Results page's Running Analyses section. 2. Select an analysis that is currently running. 3. Wait for the workflow to complete.
Expected	• The analysis status updates to "Completed" in the analysis history. • A visual completion notification or banner appears on the dashboard. • The notification clearly indicates that the analysis finished successfully.
Date-Result	

Test ID	**TC-NT-002**
Category	Functional
Severity	High
Objective	Verify that the system displays descriptive error notifications when a processing failure occurs.
Steps	1. Navigate to the New Analysis page. 2. Start an analysis using a corrupted input file. 3. Allow the system to attempt processing. 4. Navigate to the Results page's Running Analyses section.
Expected	• The system detects the failure during processing. • A visible error notification appears. • The notification explains the cause of the error and provides corrective guidance.
Date-Result	

Test ID	**TC-NT-003**
Category	Functional
Severity	Medium
Objective	Verify that the system displays real-time progress indicators during long-running workflow operations.
Steps	1. Navigate to the Results page's Running Analyses section. 2. Observe an analysis currently running in the system.
Expected	• A real-time progress indicator or progress bar is displayed. • The current workflow stage is shown. • The completion percentage updates dynamically as the workflow proceeds.

Date-Result	

Test ID	**TC-UA-001**
Category	Functional
Severity	Critical
Objective	Verify that a new user can successfully register with a valid unique username and a strong password.
Steps	1. Navigate to the Sign Up page. 2. Enter a unique username that does not exist in the system. 3. Enter a strong password meeting the minimum security requirements. 4. Re-enter the same password in the confirmation field. 5. Click “Create Account.”
Expected	• The system validates that the username is available and passwords match. • A new user account is created. • The user is redirected to the login page. • A success message is displayed.
Date-Result	

Test ID	**TC-UA-002**
Category	Functional
Severity	High
Objective	Verify that sign-up fails when the username already exists in the system.
Steps	1. Navigate to the Sign Up page. 2. Enter a username that is already registered in the system. 3. Enter any valid strong password and confirm it. 4. Click “Create Account.”
Expected	• The system detects the duplicate username. • An error message is displayed (e.g., “Username already taken.”). • No new user is created. • The user remains on the Sign Up page.
Date-Result	

Test ID	**TC-UA-003**
Category	Functional
Severity	High
Objective	Verify that sign-up fails when the two password fields do not match.
Steps	1. Navigate to the Sign Up page. 2. Enter a valid unique username. 3. Enter a strong password in the password field. 4. Enter a different string in the confirm password field.

	5. Click the “Create Account” button.
Expected	• The system displays an error message indicating passwords do not match. • No account is created. • The form remains on the Sign Up page for correction.
Date-Result	

Test ID	**TC-UA-004**
Category	Functional
Severity	Critical
Objective	Verify that a registered user can log in successfully with correct credentials.
Steps	1. Navigate to the Login page. 2. Enter the correct registered username. 3. Enter the correct password. 4. Click the “Login” button.
Expected	• The system validates credentials successfully. • A session token is created. • The user is redirected to the Dashboard page. • The user’s display name and role are shown in the sidebar.
Date-Result	

Test ID	**TC-UA-005**
Category	Functional
Severity	High
Objective	Verify that login fails and an appropriate error is shown when incorrect credentials are provided.
Steps	1. Navigate to the Login page. 2. Enter a registered username with an incorrect password. 3. Click “Login.”
Expected	• The system returns an unauthorized response. • An error message is displayed. • The user remains on the login page. • No session is created.
Date-Result	

Test ID	**TC-UA-006**
Category	Functional
Severity	High
Objective	Verify that a user can enter Guest Mode without any credentials and receives limited permissions.
Steps	1. Navigate to the Login page.

	2. Click the “Guest Mode (Offline Access)” button without entering any credentials.
Expected	• A guest session is created. • The user is redirected to the Dashboard as “Guest User.” • The user can access read and analyze functions. • Account management features (history persistence, cloud sync) are not accessible.
Date-Result	

Test ID	**TC-UA-007**
Category	Functional
Severity	High
Objective	Verify that user credentials are stored securely and that passwords are hashed in the database.
Steps	1. Register a new user account with a known password. 2. Inspect the user database record (direct database inspection by a test engineer). 3. Attempt to retrieve the plain-text password.
Expected	• The password field in the database stores a hashed value, not plain text. • The hash is consistent with a recognized algorithm. • The plain-text password cannot be retrieved from the stored hash.
Date-Result	

Test ID	**TC-UA-008**
Category	Functional
Severity	Medium
Objective	Verify that separate user workspaces are maintained and that one user cannot access another user's analyses.
Steps	1. Log in as User A and create an analysis. 2. Log out from User A. 3. Log in as User B. 4. Navigate to the Results page.
Expected	• User B’s Results page does not display User A’s analyses. • Each user’s analysis history is isolated and private. • No cross-user data leakage occurs.
Date-Result	

Test ID	**TC-UA-009**
Category	Functional
Severity	Medium
Objective	Verify that a registered user can successfully change their password through the settings interface.
Steps	1. Log in as a registered user.

	2. Navigate to the password change option in the Settings page. 3. Enter the correct old password. 4. Enter a new valid password and confirm it. 5. Submit the change.
Expected	• The system validates the old password and the new password match. • The password is updated in the database. • A success notification is displayed. • The user can log in with the new password on the next attempt.
Date-Result	

Test ID	**TC-RT-001**
Category	Functional
Severity	High
Objective	Verify that the system displays a real-time progress bar indicating the active Snakemake stage and completion percentage.
Steps	1. Navigate to the New Analysis page. 2. Upload a valid FASTQ file and start an analysis. 3. Navigate to the Results page's Running Analyses section (workflow monitoring interface) during an ongoing analysis. 4. Observe the progress bar and stage indicator while the workflow is running.
Expected	• A progress bar is visible during analysis. • The current Snakemake workflow stage is displayed. • The completion percentage updates dynamically as the workflow progresses.
Date-Result	

Test ID	**TC-RT-002**
Category	Functional
Severity	Medium
Objective	Verify that the preliminary results table updates dynamically as species are identified.
Steps	1. Navigate to the New Analysis page and upload a valid FASTQ file to start an analysis. 2. Navigate to the Results page's Running Analyses section. 3. Observe the results table during classification.
Expected	• The table dynamically updates as new species are detected. • Each row displays species name and associated read counts. • Newly identified species appear without requiring page refresh.
Date-Result	

Test ID	**TC-RT-003**
Category	Functional

Severity	Medium
Objective	Verify that the system displays live telemetry metrics for CPU/GPU usage and available RAM.
Steps	1. Navigate to the Dashboard page's System Resources and Storage sections. 2. Observe the telemetry cards displaying system resource usage.
Expected	• CPU utilization is displayed and updated periodically. • GPU usage (if available) is displayed. • Available RAM is displayed.
Date-Result	

Test ID	**TC-RT-004**
Category	Functional
Severity	Low
Objective	Verify that classification statistics are updated dynamically during processing.
Steps	1. Navigate to the New Analysis page and upload a valid FASTQ file to start an analysis. 2. Navigate to the Results page's Running Analyses section. 3. Observe the classification statistics section.
Expected	• Classification statistics update dynamically during processing. • Metrics such as number of classified reads and detected species increase as analysis progresses. • Updates occur without requiring page refresh.
Date-Result	

Test ID	**TC-RT-005**
Category	Functional
Severity	Low
Objective	Verify that the system displays the number of reads processed per minute during analysis.
Steps	1. Navigate to the New Analysis page and upload a valid FASTQ file to start an analysis. 2. Navigate to the Results page's Running Analyses section. 3. Observe the throughput or performance metrics section.
Expected	• The interface displays the number of reads processed per minute. • The value updates periodically during the analysis. • The metric provides feedback on analysis efficiency.
Date-Result	

Test ID	**TC-RT-006**
Category	Functional
Severity	Medium

Objective	Verify that the system displays preliminary results from completed workflow stages while subsequent stages continue processing.
Steps	1. Navigate to the New Analysis page and upload a valid FASTQ file to start an analysis. 2. Navigate to the Results page's Running Analyses section. 3. Observe the results panel after one workflow stage completes while others continue.
Expected	• Preliminary results from completed workflow stages are displayed. • These results remain visible while later stages continue processing. • The interface continues updating without hiding previously generated results.
Date-Result	

Test ID	**TC-GPU-001**
Category	Functional
Severity	Medium
Objective	Verify that the system correctly detects an available NVIDIA CUDA-compatible GPU and displays its availability in the UI.
Steps	1. Ensure the test machine has an NVIDIA CUDA-compatible GPU installed with appropriate drivers. 2. Launch the Pathogenius application. 3. Navigate to the Dashboard and observe the System Status panel.
Expected	• GPU status is shown as "Available" in the System Status panel. • GPU utilization metric is displayed during active analysis. • No error message is shown regarding GPU detection.
Date-Result	

Test ID	**TC-GPU-002**
Category	Functional
Severity	Medium
Objective	Verify that the system displays "Not Available" for GPU status when no CUDA-compatible GPU is present and notifies the user.
Steps	1. Ensure the test machine does NOT have a CUDA-compatible GPU (or disable it). 2. Launch the Pathogenius application. 3. Navigate to the Dashboard.
Expected	• GPU status is shown as "Not Available." • The system displays a notification informing the user that analyses may run slower without GPU acceleration. • The application does not crash and continues to function normally using CPU.
Date-Result	

Test ID	**TC-GPU-003**
Category	Functional

Severity	High
Objective	Verify that the user can enable GPU acceleration from the settings and that the system uses GPU resources during classification.
Steps	1. Ensure a CUDA-compatible GPU is available. 2. Navigate to the Settings page. 3. Locate the GPU acceleration toggle and enable it. 4. Navigate to the New Analysis page and start a new analysis with a FASTQ file. 5. Monitor the GPU utilization metric during classification.
Expected	• The GPU acceleration option can be toggled on. • During the classification stage, GPU utilization metrics increase visibly. • The analysis completes successfully. • Results are identical to CPU-mode analysis.
Date-Result	

Test ID	**TC-GPU-004**
Category	Functional
Severity	High
Objective	Verify that the system automatically falls back to CPU-only processing when GPU acceleration fails during a running analysis.
Steps	1. Enable GPU acceleration in the Settings page. 2. Simulate a GPU failure or remove CUDA availability during an active analysis (e.g., via driver disruption in a test environment). 3. Observe system behavior.
Expected	• The system detects the GPU failure and automatically switches to CPU-only processing. • The analysis continues to completion without crashing. • The user is notified of the fallback to CPU mode. • The final results are produced correctly.
Date-Result	

Test ID	**TC-GPU-005**
Category	Functional
Severity	Low
Objective	Verify that the user can disable GPU acceleration after it has been enabled, and that subsequent analyses use CPU only.
Steps	1. Enable GPU acceleration in Settings. 2. Disable GPU acceleration by toggling the option off. 3. Navigate to the New Analysis page and start a new analysis. 4. Observe GPU utilization metrics during the analysis.
Expected	• GPU acceleration is successfully disabled. • During the analysis, GPU utilization remains at baseline (no GPU workload assigned). • The analysis completes successfully using CPU only.
Date-Result	

Test ID	**TC-USE-001**
Category	Non-Functional
Severity	High
Objective	Verify that all system operations can be performed through the graphical user interface without requiring command-line interaction.
Steps	1. Launch the application. 2. Navigate through the GUI (from Dashboard to New Analysis page) to start a new analysis. 3. Upload a FASTQ file and configure analysis parameters through the interface. 4. Start the analysis and observe the workflow execution through the GUI (Results page).
Expected	• All operations are accessible through the graphical interface. • No command-line or terminal interaction is required. • The user can complete the full workflow using only GUI controls.
Date-Result	

Test ID	**TC-USE-002**
Category	Non-Functional
Severity	High
Objective	Verify that the results dashboard visually differentiates high-confidence and low-confidence species identifications.
Steps	1. Navigate to the New Analysis page, upload a valid FASTQ file, and start an analysis. 2. After completion, navigate to the Results page. 3. Observe the species identification results table.
Expected	• High-confidence species identifications are visually distinguished. • Low-confidence identifications are clearly marked or color-coded. • The visual indicators help users easily interpret classification certainty.
Date-Result	

Test ID	**TC-REL-001**
Category	Non-Functional
Severity	Critical
Objective	Verify that the system maintains full functionality in environments without internet connectivity.
Steps	1. Disconnect the system from the internet. 2. Launch the application. 3. Navigate to the New Analysis page, upload a valid FASTQ file, and start an analysis. 4. Allow the workflow to proceed through preprocessing, classification, and result generation.
Expected	• The analysis workflow runs successfully without internet connectivity. • Preprocessing, classification, and AI summarization complete normally. • No external network services are required during the process.
Date-Result	

Test ID	TC-REL-002
Category	Non-Functional
Severity	Medium
Objective	Verify that the system handles malformed or truncated FASTQ entries without terminating the entire analysis.
Steps	1. Navigate to the New Analysis page. 2. Upload a FASTQ file containing one or more malformed or truncated reads. 3. Start the analysis workflow. 4. Observe system logs and workflow execution.
Expected	• The system logs an error indicating the malformed read. • The malformed read is skipped during processing. • The rest of the analysis continues normally without termination.
Date-Result	

Test ID	TC-REL-003
Category	Non-Functional
Severity	Critical
Objective	Verify that the system produces deterministic outputs when identical inputs and databases are used.
Steps	1. Navigate to the New Analysis page, upload a specific FASTQ file to run an analysis. 2. Export or save the generated identification report. 3. Run the same analysis again using the same FASTQ file and reference database. 4. Compare the two generated reports.
Expected	• Both runs produce identical identification reports. • No variation exists between outputs generated from identical inputs. • The workflow execution is deterministic.
Date-Result	

Test ID	TC-REL-004
Category	Non-Functional
Severity	Medium
Objective	Verify that raw FASTQ files remain read-only during the analysis workflow.
Steps	1. Upload a FASTQ file and record its file checksum or last modified timestamp. 2. Start and complete an analysis. 3. Inspect the original FASTQ file after the workflow finishes.
Expected	• The original FASTQ file remains unchanged. • No modification, deletion, or overwriting occurs. • File checksum or timestamp remains identical to its original state.
Date-Result	

Test ID	**TC-PER-001**
Category	Non-Functional
Severity	High
Objective	Verify that the Pathogenius application remains operational and responsive on a mid-range consumer-grade laptop without requiring HPC resources.
Steps	1. Deploy Pathogenius on a mid-range consumer laptop (e.g., Intel Core i5, 8 GB RAM, integrated or entry-level GPU). 2. Launch the application. 3. Run a full end-to-end analysis with a medium-size FASTQ file (e.g., 500 MB). 4. Monitor UI responsiveness during analysis.
Expected	• The application launches successfully on the target hardware. • The analysis completes without memory overflow or crash. • The UI remains navigable and responsive while analysis runs in the background. • No requirement for HPC cluster or cloud computing.
Date-Result	

Test ID	**TC-PER-002**
Category	Non-Functional
Severity	High
Objective	Verify that the user can configure a memory cap for the classification algorithm and that the system does not exceed the specified RAM limit.
Steps	1. Open the Settings page in Pathogenius. 2. Set a memory cap for the classification algorithm (e.g., 4 GB). 3. Start an analysis with a FASTQ file large enough to stress memory usage. 4. Monitor RAM consumption throughout the analysis.
Expected	• The classification process respects the configured memory cap. • RAM usage does not exceed the specified limit during classification. • The analysis completes (possibly more slowly) without crashing or throwing an out-of-memory error.
Date-Result	

Test ID	**TC-PER-003**
Category	Non-Functional
Severity	High
Objective	Verify that computationally intensive analysis tasks run as background processes and do not freeze or block the Electron frontend.
Steps	1. Navigate to the New Analysis page and start a new analysis with a large FASTQ file. 2. While the analysis is running, attempt to navigate between sections (Dashboard, Results, Database). 3. Interact with UI elements such as search filters and buttons during the analysis.

Expected	• The frontend remains interactive and navigable throughout the analysis. • No UI freeze, hang, or unresponsive state is observed. • Navigation between pages completes without delay attributable to the background computation.
Date-Result	

Test ID	**TC-PER-004**
Category	Non-Functional
Severity	Low
Objective	Verify that the system provides a reasonable estimated time-to-completion for an ongoing analysis and updates the estimate as processing progresses.
Steps	1. Navigate to the New Analysis page. 2. Start a new analysis with a valid FASTQ file. 3. Navigate to the Results page and observe the running analysis banner. 4. Wait for at least two workflow stages to complete (e.g., Preprocessing then Classifying). 5. Observe changes in the estimated time remaining.
Expected	• An estimated time remaining is displayed at the start of the analysis. • The estimate updates as workflow stages complete. • The estimate decreases progressively and does not display a static or incorrect value.
Date-Result	

Test ID	**TC-SUP-001**
Category	Non-Functional
Severity	High
Objective	Verify that session logs are generated for each analysis run and contain timestamps, analysis parameters, and hardware utilization data.
Steps	1. Start and complete a new analysis. 2. Locate the generated session log file in the application's log directory. 3. Open and inspect the log file contents.
Expected	• A log file is created for the analysis session. • The log includes timestamps for each major event. • Analysis parameters (FASTQ file, database version, confidence threshold) are recorded. • Hardware utilization data (CPU %, RAM usage) is present in the log.
Date-Result	

Test ID	**TC-SUP-002**
Category	Non-Functional
Severity	High
Objective	Verify that the application can be deployed as a containerized or managed environment and produces consistent results across Windows and Linux.

Steps	1. Deploy Pathogenius on a Windows machine using the provided installer or container. 2. Deploy Pathogenius on a Linux machine using the same package. 3. Run the same analysis with identical FASTQ input and configuration on both platforms. 4. Compare the output results and check for functional parity.
Expected	• The application deploys and launches successfully on both platforms. • Analysis results (classified species, confidence scores) are identical on both platforms. • No platform-specific errors or missing features are observed.
Date-Result	

Test ID	**TC-SUP-003**
Category	Non-Functional
Severity	Medium
Objective	Verify that an individual Snakemake workflow rule can be updated or replaced without requiring changes to other pipeline modules.
Steps	1. Identify the Snakemake rule for preprocessing in the source code. 2. Replace the preprocessing tool reference with an alternative tool while keeping input/output interfaces unchanged. 3. Run a full analysis pipeline. 4. Verify that downstream steps proceed normally.
Expected	• The updated preprocessing rule executes without errors. • Downstream pipeline steps receive valid input from the updated module. • The full analysis completes successfully. • No other Snakemake rules require modification.
Date-Result	

Test ID	**TC-SCA-001**
Category	Non-Functional
Severity	High
Objective	Verify that the system can process a small FASTQ file (1 MB) and a large FASTQ file (10 GB) and scales its resource usage accordingly.
Steps	1. Start an analysis with a 1 MB FASTQ file and record CPU usage, RAM usage, and processing time. 2. Start an analysis with a 10 GB FASTQ file and record the same metrics. 3. Compare resource usage between both runs.
Expected	• Both analyses complete without errors. • Resource consumption (CPU, RAM, disk I/O) is higher for the 10 GB file, demonstrating dynamic scaling. • Neither run crashes due to resource exhaustion. • Results are produced for both inputs.
Date-Result	

Test ID	**TC-SCA-002**

Category	Non-Functional
Severity	High
Objective	Verify that the system automatically detects the number of available CPU cores and NVIDIA CUDA-compatible GPUs upon initialization.
Steps	1. Launch the Pathogenius application on a machine with a known hardware configuration (e.g., 8-core CPU, 1 CUDA GPU). 2. Navigate to the Dashboard and observe the System Resources panel. 3. Navigate to Settings and check hardware detection information.
Expected	• The correct number of CPU cores is detected and displayed. • If a CUDA GPU is present, it is detected and shown as “Available.” • Detection occurs automatically on startup without user intervention.
Date-Result	

Test ID	**TC-SCA-003**
Category	Non-Functional
Severity	Medium
Objective	Verify that the classification module utilizes multi-threaded CPU processing on a multi-core machine, improving throughput compared to a single-thread baseline.
Steps	1. Configure the system to run with all available CPU cores (default setting). 2. Start an analysis with a medium FASTQ file and record total classification time. 3. Configure the system to use a single CPU thread. 4. Run the same analysis and record total classification time. 5. Compare the two results.
Expected	• Multi-threaded mode completes classification in less time than single-threaded mode. • No correctness degradation is observed (same classification results). • System logs confirm multi-core utilization during the multi-threaded run.
Date-Result	

6 Consideration of Various Factors in Engineering Design

6.1 Constraints

The design of Pathogenius was influenced not only by technical requirements, but also by the context in which the system is intended to be used. Since the project aims to support offline pathogen analysis on modest hardware, factors such as public health impact, data security, accessibility, deployment conditions, and cost had a direct effect on design decisions. In our earlier analysis, we had already identified offline operation, low hardware requirements, privacy-preserving local processing, and non-expert usability as core constraints, and these continued to guide the detailed design stage as well.

6.1.1 Public Health

Public health is one of the most relevant factors for Pathogenius. The system is designed for settings such as field hospitals, emergency response situations, and resource-constrained clinics, where fast and local pathogen screening may be valuable. This is why the project emphasizes offline execution, rapid automated workflows, and readable output for non-expert users. At the same time, Pathogenius is not presented as a definitive diagnostic tool; it is a decision-support system that helps users interpret sequencing-based evidence more easily and more quickly.

6.1.2 Public Safety and Security

Public safety and security considerations influenced several architectural decisions in Pathogenius. Since biological sequencing data may contain sensitive information, including human genetic material, protecting user data and minimizing unnecessary exposure were important design goals. For this reason, the system supports local offline execution, allowing users to run analyses entirely on their own machines without transferring data externally. At the same time, the system also provides an optional cloud-based execution mode for users who prefer scalable computation resources or centralized deployment. This dual design allows institutions to choose the deployment model that best fits their security policies and operational needs. In both modes, the system emphasizes transparency in results and clear reporting of uncertainty to prevent misinterpretation of probabilistic outputs.

6.1.3 Public Welfare

Pathogenius supports public welfare by making advanced pathogen analysis more accessible. Existing bioinformatics workflows often assume expert users, powerful hardware, and reliable connectivity. Our system tries to reduce these barriers through a graphical interface, automated workflow management, and human-readable summaries. In that sense, the project aims to make useful analysis capabilities available to a wider range of users and institutions.

6.1.4 Global Factors

Global factors were relevant because healthcare infrastructure is not equally distributed. Many existing pathogen analysis solutions depend on cloud access, expensive infrastructure, or specialized personnel. Pathogenius was designed with a different assumption: it should still function in places with poor connectivity or limited computational resources. The decision to run core workflows fully offline after installation was directly shaped by this concern.

6.1.5 Cultural Factors

Cultural factors had a more limited effect compared to the other categories. Still, trust and acceptability matter, especially when handling sensitive biological data. Keeping the system local and avoiding unnecessary cloud dependency can make the platform more acceptable in environments where data sovereignty and external data transfer are sensitive concerns. In addition, the interface is designed to be straightforward and neutral rather than overly technical or specialized in tone.

6.1.6 Social Factors

Social factors were important mainly because the project tries to reduce the expertise gap in metagenomic analysis. Many current tools are difficult for non-bioinformatics users to operate and interpret. By hiding command-line complexity behind a GUI and supplementing technical outputs with readable summaries, Pathogenius aims to make this type of analysis more usable for a broader group of people. This can help smaller institutions or local teams benefit from tools that would otherwise remain accessible only to specialists.

6.1.7 Environmental Factors

Environmental factors were considered, although they were not the primary driver of the design. The system is designed to run on mid-range laptops and to avoid unnecessary dependence on cloud infrastructure or high-performance clusters. This keeps the computational footprint lower and fits the project's goal of efficient local analysis. The reuse of local databases and reference indices also helps avoid repeated heavy computation.

6.1.8 Economic Factors

Economic factors had a strong effect on the design. One of the project goals is to avoid recurring usage costs and reduce dependence on expensive infrastructure. This is why the system uses open-source tools such as Snakemake and CLARK, runs locally after installation, and targets modest commercial hardware instead of specialized computing systems. These decisions make the project more practical for resource-constrained settings.

6.1.9 Summary Table of Factors and Effects

Table 1: Design Factors and Their Effects.

Factor	Effect Level (0–10)	Explanation
Public Health	10	Core motivation of the project; shaped offline screening and decision-support design.
Public Safety / Security	9	Strongly influenced local processing, privacy protection, and confidence-aware result presentation.
Public Welfare	8	Encouraged accessible design for users without advanced bioinformatics expertise.
Global Factors	8	Motivated offline capability and suitability for low-resource settings.
Cultural Factors	4	Less central, but relevant in terms of trust, data sovereignty, and neutral interface design.

Social Factors	8	Influenced the goal of democratizing access to advanced pathogen analysis tools.
Environmental Factors	5	Encouraged efficient local execution on modest hardware.
Economic Factors	9	Strongly influenced the low-cost, open-source, non-cloud-dependent design.

6.2 Standards

6.2.1 IEEE 830

IEEE 830 guided the way we defined and documented the project requirements. It helped us write clearer functional and non-functional requirements and made it easier to connect design decisions back to actual system needs. This was especially useful for a project like Pathogenius, where workflow, usability, reliability, and offline operation all needed to stay aligned from analysis to design.

6.2.2 ISO/IEC 25010

ISO/IEC 25010 [5] was important in shaping the quality goals of the system. Our non-functional requirements were already organized around qualities such as usability, reliability, performance, supportability, and scalability, and this standard gave us a structured framework for those decisions. In practice, it influenced choices such as modular workflow design, responsive GUI behavior, reliable offline functionality, and maintainable subsystem separation.

6.2.3 UML 2.5.1 - Unified Modeling Language

UML 2.5.1 [6] was used to model the system structure and behavior through use case, class, activity, state, and sequence diagrams. This helped us represent subsystem boundaries, workflow execution, and user interactions in a standard and readable way. Since Pathogenius combines frontend interaction, workflow management, local storage, and AI-based explanation, UML was useful in keeping the architecture understandable and well-documented.

6.2.4 ISO 9241-210

ISO 9241-210 [7] influenced the user-centered side of the project. Since Pathogenius is meant to be used by people who may not be comfortable with command-line bioinformatics tools, the system needed to be designed around usability, clarity, and error reduction. This standard supports choices such as a GUI-only workflow, clear status updates, readable error messages, and simplified explanations of technical results.

7 Teamwork Details

7.1 Contributing and Functioning Effectively on the Team

The Pathogenius project requires expertise in several areas, including bioinformatics pipelines, user interface design, workflow orchestration, system architecture, and testing. To ensure effective collaboration, responsibilities were distributed according to each member's technical strengths while maintaining regular coordination among all members.

- **Yiğit Ali Doğan** is responsible for the overall system design and architecture, including subsystem decomposition and high-level UML modeling. He also contributes to backend development by designing the Snakemake workflow and integrating services. In addition, he focuses on hardware/software mapping, particularly GPU acceleration using NVIDIA CUDA.
- **Ege Ateş** primarily focuses on backend development, including implementing the Snakemake workflow and integrating databases. He also works on the integration of signal simulators for sequencing reads and contributes to improving the scalability of the system.
- **Nazlı Apaydın** focuses on frontend development, designing a user-friendly graphical interface that allows users to run analyses and interpret results easily. She also works on data visualization, converting classification results into clear and understandable graphs and reports.
- **Ata Uzay Kuzey** contributes mainly to frontend development and reporting components, including generating web-based reports and visual representations of analysis outputs. In addition, he manages the integration of backend services with the local AI assistant that produces human-readable explanations of the results.
- **Yunus Günay** acts as the lead for testing and integration, ensuring that the different system components function correctly together. He is responsible for preparing dockerized environments and managing demo setups. As a cross-functional member, he helps integrate frontend components and contributes to resolving interface issues during development.

7.2 Helping Creating a Collaborative and Inclusive Environment

To maintain effective collaboration, the team established regular communication and coordination mechanisms throughout the project. Weekly meetings were held either face-to-face or through Zoom, where members shared progress updates, discussed design decisions, and addressed technical challenges. These meetings allowed members responsible for different subsystems to explain their work and receive feedback from the rest of the team.

Outside of scheduled meetings, the team primarily communicated through a WhatsApp group, which allowed quick coordination on smaller issues such as debugging problems, integration questions, or scheduling decisions. This helped maintain continuous communication and ensured that problems could be addressed quickly without waiting for formal meetings.

Version control through GitHub was used to share code, track updates, and review each other’s contributions. Integration tasks required close coordination between frontend, backend, and workflow components, which encouraged collaboration across different parts of the system.

The team environment encouraged open discussion and participation from all members. During meetings, members were able to propose ideas, raise concerns, and contribute to architectural or implementation decisions. This collaborative approach helped maintain an inclusive working environment and ensured that all members were actively involved in the project development process.

7.3 Taking Lead Role and Sharing Leadership on the Team

Leadership responsibilities in the Pathogenius project are distributed among team members depending on the subsystem being developed. Instead of having a single centralized leader, different members take the lead for different aspects of the system.

- **Yiğit Ali Doğan** leads the system architecture and design, guiding the overall structure of the system and ensuring consistency between subsystems.
- **Yunus Günay** leads testing and integration, coordinating the merging of frontend and backend components and ensuring that the system functions correctly in integrated environments.
- **Ata Uzay Kuzey** takes the lead in documentation and deliverables, coordinating the preparation of project documentation, technical materials, and the project website.
- **Nazlı Apaydın** leads requirements and analysis, and the user interface design, ensuring that the application remains accessible and easy to use for non-expert users.
- **Ege Ateş** takes a leadership role in backend workflow development, ensuring the proper integration of Snakemake workflows and classification tools.

By distributing leadership across different components of the project, the team ensures that each subsystem benefits from focused guidance while still maintaining overall coordination. Major architectural decisions are discussed collectively during team meetings, allowing all members to participate in decision-making while still maintaining clear responsibility for each subsystem.

8 Glossary

1. **FASTQ:** A sequencing read file format that stores nucleotide sequences with per-base quality scores.
2. **FASTA:** A sequence file format typically used for storing reference genome sequences.
3. **Read:** A single nucleotide sequence produced by a sequencing device and stored in a FASTQ file.
4. **Sequencing:** The process of generating nucleotide read data from biological samples.
5. **Long-Read Sequencing:** Sequencing that produces long read fragments (used as the system's primary input style).
6. **k-mer:** A substring of length *k* extracted from reads/genomes, used for matching and classification.
7. **Index (k-mer index):** A CLARK-compatible database structure enabling fast k-mer lookups during classification.
8. **Snakemake:** A workflow engine used to coordinate and execute the analysis pipeline reproducibly.
9. **Workflow:** An ordered set of steps (rules) executed to process inputs into final outputs.
10. **Pipeline:** The end-to-end chain of preprocessing, classification, and post-processing stages.
11. **Electron.js:** The framework used to build the application interface.
12. **Offline Execution:** System operation without requiring internet connectivity during analysis.
13. **Reference Database:** Locally stored genomes used by CuCLARK and CLARK-lite to classify reads.
14. **Metagenomic Analysis:** Sequencing-based analysis of genetic material from mixed samples to identify organisms.
15. **Taxonomic Classification:** Assigning reads to taxa such as species based on sequence similarity evidence.
16. **NVIDIA CUDA:** The GPU computing platform referenced for future acceleration support.
17. **NCBI (National Center for Biotechnology Information):** A source of curated genomic resources used for building local reference databases.
18. **GPU:** The graphics processor targeted for acceleration in later project stages.
19. **CPU:** The host processor used for analysis when GPU acceleration is unavailable or disabled.
20. **Sequencing Depth:** The total number of reads representing a sample, influencing detection sensitivity.
21. **False Positive:** An incorrect classification where a read is assigned to a taxon not actually present in the sample.
22. **Confidence Score:** A numerical measure indicating the reliability of a taxonomic classification result.
23. **Preprocessing:** Initial analysis steps applied to raw FASTQ data, such as quality filtering and cleanup, before classification.
24. **Deterministic Execution:** Pipeline behavior that produces identical outputs when given the same inputs and reference data.

9 References

[1] Y. L. Oon, Y. S. Oon, M. Ayaz, M. Deng, L. Li, and K. Song, “Waterborne pathogens detection technologies: advances, challenges, and future perspectives,” *Frontiers in Microbiology*, vol. 14, Art. no. 1286923, Nov. 2023, doi: 10.3389/fmicb.2023.1286923.

[2] Bio-Rad Laboratories, “Pathogen detection,” Bio-Rad Laboratories, Online. Available: https://www.bio-rad.com/en-tr/a/ls/pathogen-detection. Accessed: Dec. 18, 2025.

[3] K. Sandås, J. Lewerentz, E. Karlsson, L. Karlsson, D. Sundell, K. Simonyté-Sjödin, and A. Sjödin, “Nanometa Live: a user-friendly application for real-time metagenomic data analysis and pathogen identification,” *Bioinformatics*, vol. 40, no. 3, Art. no. btae108, Mar. 2024, doi: 10.1093/bioinformatics/btae108.

[4] L. E. Braley, J. B. Jewell, J. Figueroa, J. L. Humann, D. Main, G. A. Mora-Romero, N. Moroz, J. W. Woodhall, R. A. White III, and K. Tanaka, “Nanopore sequencing with GraphMap for comprehensive pathogen detection in potato field soil,” *Plant Disease*, vol. 107, no. 8, pp. 2288–2295, Aug. 2023.

[5] International Organization for Standardization, *Systems and software engineering—Systems and software Quality Requirements and Evaluation (SQ*International Organization for Standardization, *Systems and software engineering—Systems and software Quality Requirements and Evaluation (SQuaRE)—System and software quality models*, ISO/IEC 25010:2011, 2011*uaRE)—System and software quality models*, ISO/IEC 25010:2011, 2011.

[6] Object Management Group, *Unified Modeling Language (UML) Specification*, Version 2.5.1, Dec. 2017. [Online]. Available: https://www.omg.org/spec/UML/2.5.1

[7] International Organization for Standardization, *Systems and software engineering—Life cycle processes—Requirements engineering*, ISO/IEC/IEEE 29148:2018, 2018.