Bilkent University

Department of Computer Engineering

Senior Design Project

T2522

CASSIE

Final Report

Ege Şirvan, 22102289, ege.sirvan@ug.bilkent.edu.tr
Eren Aslan, 22102329, eren.aslan@ug.bilkent.edu.tr
Arda Kırcı, 22002031, arda.kirci@ug.bilkent.edu.tr
Emre Can Yoloğlu, 22102542, can.yologlu@ug.bilkent.edu.tr
Osman Baktır, 22002553, osman.baktir@ug.bilkent.edu.tr

Supervisor

Can Alkan

1. Introduction

CASSIE (Cloud-based Assembly Streamlined Service Integration Engine) is a cloud-based genome assembly and annotation platform that automates complex genome analysis processes and makes large-scale genome assembly procedures more accessible to researchers. Modern genome assembly workflows require the integration of multiple complex tools, the management of highly compute-intensive steps, and deep proficiency in using those tools. Researchers without a background in bioinformatics infrastructure are frequently unable to exploit state-of-the-art assembly methods, even when those methods are freely available as open-source software.

CASSIE provides a streamlined environment where users can upload their sequencing data, select appropriate tools and pipeline steps, and run complete assembly workflows from a simple web interface with ease, effectively eliminating the complexity problem. CASSIE supports both human and non-human genomic data and appropriate tools, enabling the system to address a wide range of use cases. The system also has embedded privacy policies appropriate for sensitive genomic datasets.

CASSIE automates the fundamental steps of a standard genome assembly process: quality control, estimation of genomic properties, assembly, assembly quality assessment, and, optionally, repeat annotation and gene annotation. All of the required genomic tools are containerised using Docker, and workflow orchestration is managed by Nextflow and Kubernetes. By this architecture, CASSIE benefits from reproducibility, modularity, flexible scalability, and adaptability.

In the production deployment on AWS, EC2 instances serve as the scalable compute layer, AWS S3 provides cloud-based object storage, and Kubernetes (Minikube on the EC2 host) handles workflow orchestration. The EC2 instance is fronted by an AWS Application Load Balancer. Due to all these properties of the system, CASSIE abstracts all operational complexities from the end user, allowing researchers to focus directly on scientific results.

2. Requirement Details

2.1 Problem Statement and Motivation

The rapid expansion of genomics and the growing need for biological analyses have necessitated the development of accessible computational platforms capable of processing large-scale biological datasets without requiring deep toolchain proficiency. Web-based workflow management systems such as Galaxy [1] have become the industry standard for general-purpose bioinformatics, successfully opening the world of bioinformatics to a broader audience by providing a graphical interface for tools across many domains.

However, the one-platform-for-everything approach of such systems introduces significant complexities for specialised tasks like genome assembly. These systems are designed to support every conceivable pipeline, therefore they place a heavy orchestration work load entirely on the user. In order to use these aforementioned systems, researchers must set up confusing software options that must be manually selected, configured, and wired together - requiring deep infrastructure domain expertise to avoid compatibility errors. Furthermore, these platforms usually operate as black boxes. They provide little to no transparency about the computational cost or expected duration of a job before execution, leading to inefficient time management and budget uncertainty.

CASSIE addresses these important issues by being a system of not a general-purpose workbench but a specialised, automated workflow manager for genome assembly. Unlike traditional systems that require manual pipeline construction, CASSIE provides a goal-oriented interface where users define their objectives. Built on a cloud-native architecture using Docker, Nextflow, and Kubernetes, CASSIE automates pipelines covering quality control, assembly, assembly analysis, and annotation while ensuring full reproducibility. It uniquely introduces a pre-execution time and price estimation step, making the genome assembly pipeline more accessible and economically predictable.

2.2 Functional Requirements

Requirements below define the core functionality that the CASSIE system fulfills.

User and Access Requirements

The system provides an authentication mechanism in the web app that allows users to log in securely. Each submitted job is executed within its own dedicated Kubernetes namespace to isolate it from other processes and data from other jobs. Each user's data storage space on S3 is configured independently and isolated from other users. The system allows users to cancel or terminate a running workflow execution.

Data Upload and Management Requirements

The system supports uploading genomic sequencing files in formats such as FASTQ and FASTA. Uploaded data is automatically stored in the isolated storage area assigned to the corresponding user. The system allows users to transfer data directly from their own cloud storage services into CASSIE's storage without them having to download the data to their local machines.

Workflow Configuration Requirements

Users are able to choose which bioinformatics tools will be used in a workflow. The system automatically displays the required files, based on the selected tools. The allows multiple workflows with different configurations to be executed on the same data. The system is capable of asking high-level questions about the intended analysis goals and automatically selecting appropriate tools. Users are able to manually construct custom workflows without relying on automated tool selection. The system is able to estimate expected execution time and total cost for all available EC2 instance classes, each offering different CPU and memory capacities. Based on estimated requirements and execution time, the system should recommend a suitable EC2 instance class. The finalized workflow is executed on the EC2 instance class selected by the user.

Workflow Execution Requirements

The system submits configured workflows to Nextflow for execution using Kubernetes. Each workflow step is executed as a separate Kubernetes pod. Kubernetes schedules pods according to defined CPU and memory limits. The system records metadata such as execution duration and CPU/memory usage levels. Temporary and intermediate files, except for the tool outputs, are removed after execution.

Post-Execution Requirements

The system records and displays the execution status of each workflow step and notifies the user if they choose to. All output files are written to the user's designated storage folder in the bucket. After completion, users are able to download output files one by one or as a single compressed file. The system provides users with readable and accessible workflow execution logs.

Tool Requirements

The system supports quality control analyses for sequencing data. The system supports estimation of genome size, heterozygosity, and repeat content. The system supports genome assembly operations. The system supports scaffolding, polishing, and gap-filling steps. The system supports the assessment of assembled genomes. The system supports repeat and segmental duplication analysis. The system supports gene annotation workflows.

Community Workflow Requirements

The system allows users to publish workflows they have created to the community. The system ensures that publishing a workflow does not expose any user datasets or execution outputs. The system provides a community workflow list where users can browse shared workflows. The system allows users to search and filter community workflows. The system displays the vote count for community workflows. The system allows users to import a community workflow into their own pipeline list. Imported community workflows are editable independently of the original workflow. The system allows users to vote for community workflows.

Forum Requirements

The system allows users to publish forum posts they created to the forum. Forum posts allow users to publish up to 4 images. The system allows users to search and filter forum posts. The system displays vote count for forum posts. The system allows users to comment on forum posts. The system allows users to comment on forum post comments. The system allows users to vote for forum posts.

2.3 Non-Functional Requirements

This section defines the criteria that CASSIE meets in terms of usability, reliability, and scalability. These requirements include the core features that impact the system's user experience and operational stability.

Usability

The system provides a clear and simple web interface that enables users to run genome assembly workflows step-by-step with ease. The interface only shows the parameters for the related tools and reduce unnecessary complexity. The error messages are presented in a way that is readable, clear, and guides the user towards the correct solution. Workflow statuses are (ready, working, completed, error) outputted to the user via real-time or near-real-time updates. The users are able to access previous jobs, results, and logs from a single panel. The users are able to access and execute workflows from other users if they have shared them. The system provides contextual help and information about tools and parameters.

Reliability

The system gives readable error notifications in the event of a failure in the workflow steps. User data is stored in a durable storage layer to prevent data loss in the event of network outages or system failures. All workflows are designed to produce consistent and reproducible results when re-executed with the same parameters, tool versions, reference data versions, and random seeds.

Scalability

The system supports multiple workflows started by multiple users. The Kubernetes cluster provides horizontal scalability by adding new nodes when system load increases. User-specific namespace architecture guarantees resource isolation even under heavy load. The storage system (S3) is configured in a way that allows high volume and large numbers of concurrent read/write operations. The system architecture is designed with the flexibility to increase the number of tools if needed. The system is able to work with both small and large data.

3. Final Architecture and Design Details

3.1 Architectural Overview

CASSIE has a three-layer cloud-native structure. The browser-facing layer is a React single-page app delivered by nginx, which also forwards /api/ requests to the backend. The application layer is a FastAPI service running on uvicorn. The data layer includes PostgreSQL 15 for relational metadata and AWS S3 for storing large data. Kubernetes (Minikube) provides container orchestration. In the AWS production deployment, the backend uses AWS S3, Minikube runs on the EC2 host. An AWS Application Load Balancer fronts the instance on port 3000.

3.2 Subsystem Decomposition

The five subsystems discussed in previous reports are implemented as described below. Figure 3 shows the inter-subsystem communication flows as documented in the DDR subsystem diagram.

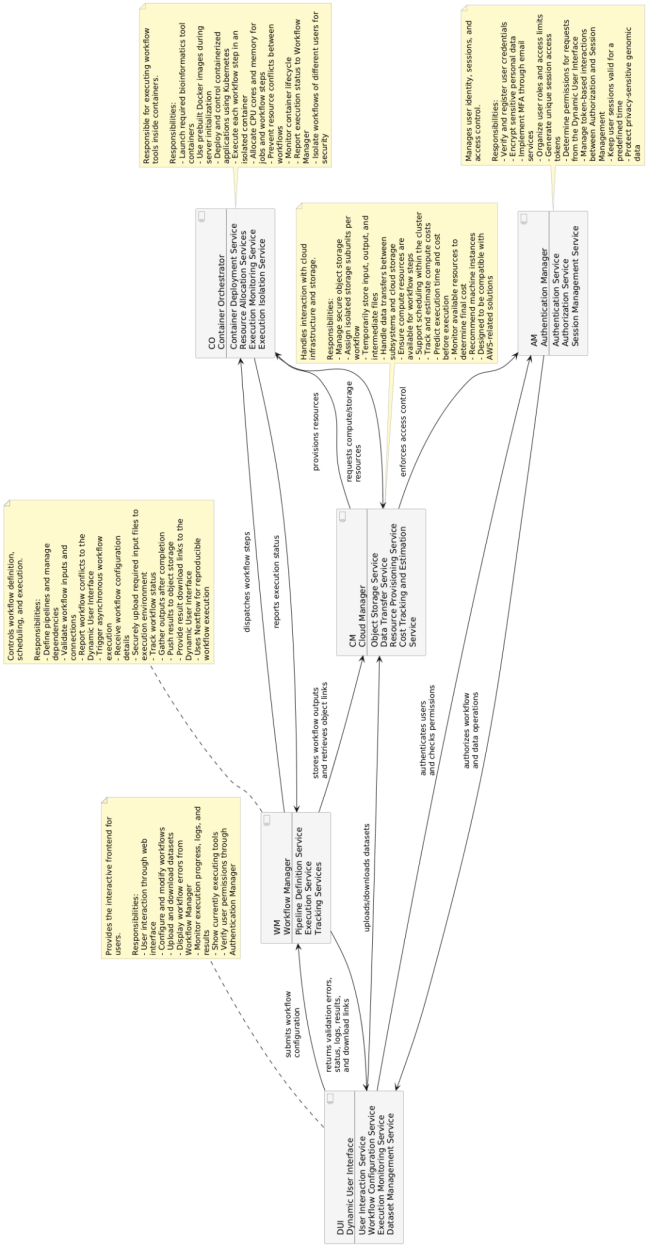

Figure 1 - Subsystem Decomposition

3.2.1 Dynamic User Interface

The Dynamic User Interface subsystem provides the interaction layer between users and the CASSIE platform. It enables users to configure workflows, upload datasets, monitor execution progress, and access analysis results. It is implemented as a React 18 / TypeScript 5 single-page application built with Vite 5, using ReactFlow 11 for the visual pipeline builder, React Router 6

for navigation, and Axios for API calls. nginx serves the built SPA and proxies /api/ requests to the FastAPI backend.

User Interaction Service: This subsystem allows users to interact with the platform through a web interface. These services allow users to configure workflows, manage datasets, start various tasks, and interact with other users.

It consists of authenticated dashboard, public landing page, profile page consisting profile editing, notification preferences, and 2FA toggle, balance page, login and register pages with an email-based password reset page and email verification page.

Workflow Configuration Service: Users can create and modify workflows through the interface. The subsystem collects workflow parameters and configuration details and forwards them to the Workflow Manager subsystem. It also shows the error messages (if any exist) that were received from the Workflow Manager in case of a parameter mismatch.

It consists of a ReactFlow canvas with drag-and-drop tool blocks, a list of saved pipelines which allow editing, deleting and using a pipeline in a job, a starter templates page which allows users to use pre-defined pipelines and a create job page which user makes configurations for their job.

Execution Monitoring Service: This subsystem retrieves workflow execution status, logs, and results from the Workflow Manager and presents them to users in a readable format. It also shows which tools are currently executing in their workflow session.

It consists of a job list with status filters and progress indicators, job based tool status, inputs, outputs, and logs.

Dataset Management Service: The interface supports dataset upload and download operations. Data transfers are coordinated with the Cloud Manager subsystem. The Dynamic User Interface subsystem utilizes the Authentication Manager to verify user access permissions before allowing any workflow or data operations.

It consist of a storage page where user can upload files to their storage according to their storage limit and view outputs of previous jobs.

Community and Forum: The subsystem allows users to publish their pipelines and questions on their respective pages. It also allows users to view and interact with other users pipelines and forum posts; these interactions are voting, commenting, visiting profile pages of the users and reporting users.

It consists of a pipeline list which allows browse, import, and vote on shared pipelines with search and tool filtering, a forum post list with search and tag filtering, a forum post creation page with title, body, tags, and image attachment, a forum post view with answers, nested comments, votes, and moderation reporting.

3.2.2 Container Orchestrator

The Container Orchestrator subsystem manages the execution of genomic tools inside containerized environments. In CASSIE, this functionality is implemented using Kubernetes to deploy and control containerized applications.

Container Deployment Service: This subsystem launches containers of the required bioinformatics tools. Containers get their image instances from the already existing Docker images of genomic tools in the system, which are built during server initialization. Genomic tool images are built using publicly available links. Each workflow step is executed inside an isolated container environment.

It renders the Kubernetes Job manifest for each tool, substituting the tool image, CPU requests, memory limits, and shared-volume paths. It builts tool images via buildtools.sh during server initialisation.

Resource Allocation Service: Maximum computational resources, such as the number of available CPU cores and the amount of available memory, are allocated for each job. This service also allocates an appropriate number of CPU cores and amount of memory for each workflow step according to the maximum computational resources. This ensures efficient use of system resources and prevents resource use conflicts between workflows.

It specifies CPU and memory requests and limits derived from the selected VM tier and per-tool defaults for each job manifest.

Execution Monitoring Service: This subsystem monitors the lifecycle of running containers and reports execution status to the Workflow Manager subsystem. So that the Workflow Manager Subsystem can redirect the current workflow status to the Execution Monitoring Services in the Dynamic User Interface Subsystem.

It polls the Kubernetes API at pre-defined intervals to track pod status, streams the log lines from pod stdout and stores in the filesystem for display in the Dynamic User Interface.

Execution Isolation Service: Different users' workflows are executed in separate, isolated containers, ensuring security and preventing interference between workflows.

It executes each workflow in a dedicated Kubernetes namespace derived from the user ID and job execution ID, ensuring CPU allocations, intermediate files, and workflow results from different users cannot interfere.

3.2.3 Workflow Manager

This subsystem is responsible for handling workflows using Nextflow, which provides consistent executions across different environments. While ensuring full reproducibility, it automates complex pipelines, from quality control to assembly, analysis, and annotation.

Pipeline Definition Service: This service defines workflows and manages the dependencies required for biological analyses. To establish a compatible pipeline between Workflow Configuration Services of the Dynamic User Interface Subsystem and the Workflow Manager Subsystem, it supports a workflow creation system that utilizes text-based inputs. It also evaluates the inputs and connections that are defined in workflows and reports any conflicts to the Dynamic User Interface Subsystem.

It persists visual pipelines as ReactFlow JSONB in the pipelines table, detects required input file types, and validates the compatibility of tool connections and returns error messages to the Dynamic User Interface in case of an error, and can suggest a pipeline from high-level questions the system presents to the user.

Execution Service: This subsystem triggers the workflow execution. To ensure a responsive user interface, tasks are executed asynchronously in the background. It receives workflow configuration details from the Pipeline Definition Services. It securely uploads required input files from the user into the execution environment.

It starts pipeline execution, translates ReactFlow JSONB into an executable plan, copies input files from the storage into the execution environment before the job submission.

Tracking Services: This subsystem tracks the status of workflows. After a successful completion of a workflow, it locates the result directory, gathers the generated outputs, and pushes them to the object storage. It provides access links to download workflow results to the Execution Monitoring Service of the Dynamic User Interface Subsystem.

It tracks per-tool status and job status, uploads outputs to the user's S3 bucket upon completion, creates a ZIP archive consisting all output files before flagging the job as completed, provides download URLs to the Dynamic User Interface, and wires billing settlement.

3.2.4 Cloud Manager

This subsystem manages the interactions between the other CASSIE subsystems and the cloud infrastructure on which it runs. The Cloud Manager Subsystem is designed to be compatible with AWS and related cloud solutions.

Object Storage Service: This subsystem is responsible for managing secure object storage. Each workflow will be assigned to an isolated storage subunit, which will be used to store input/output/intermediate files.

It implements the folder per user model in an S3 bucket, and a folder is created automatically at user registration. Downloads are served via boto3-generated URLs.

Data Transfer Service: This subsystem is responsible for handling the data transfers between the other CASSIE subsystems and the cloud storage services.

It mediates all file record operations, and large genomic files are transferred in multipart mode for datasets exceeding the boto3 single-part limit.

Resource Provisioning Service: This subsystem manages the cloud resources. When a workflow step is ready to execute, the service ensures that appropriate compute resources are available and that the task can be scheduled within the cluster.

It implements a multi-tier VM partition scheduler, each having a different number of possible jobs and different price-per-minute values.

Cost Tracking and Estimation Service:

This subsystem is a deterministic, transparent heuristic that combines baseline runtimes for each tool with VM speed, input size, and compression penalties. Runtime is adjusted by VM density, input size and compression. In sequential workflows, individual runtimes are summed up for the final runtime.

3.2.5 Authentication Manager

This subsystem is responsible for handling authentication-related issues in the system. It manages user-access protocols, password/MFA actions, and user roles. It grants secure access to and control of privacy-sensitive genomic data.

Authentication Service: This subsystem verifies and registers user credentials in compliance with the existing user data. It encrypts sensitive personal data using novel methods and implements Multifactor Authentication to ensure personal access security via email services.

It issues and validates JWTs using PyJWT, hashes passwords with bcrypt, implements a unified one-time-code for email verification, 2FA, password reset, and account-deletion confirmation.

Authorization Service: This subsystem organizes in-system user-roles and their access limits. Each user is given a unique access token in each session by the Session Management Service, which is then used to determine user permissions whenever a request is sent from the Dynamic User Interface Subsystem. The Authorization Service manages the access token-related interactions between the Dynamic User Interface and the Session Management Service.

It makes each protected API route to validate the JWT from the Bearer header of the request, encodes user roles (user, admin) in the token payload, enforces resource-level authorisation, enforces resource caps per user.

Session Management Service: This subsystem generates unique access tokens whenever a user logs in, which are valid for a predetermined amount of time. These tokens are used to provide a continuous experience between different pages of the website until they expire. They are also used by the Authorization Service to decide user roles.

It carriest JWTs expiry, exposes active-session, enforces account suspension at JWT validation which delivers suspended users a 403 error message.

3.3 Persistent Data Layer

PostgreSQL 15 is the system of record for all relational metadata. All tables and their functionality can be seen in Figure 4.

Table	Purpose
users	Stores user accounts, balances, settings, and authentication data
workflows	Stores workflow definitions and reusable execution structures.
pipeline_configs	Stores saved pipeline configuration data linked to workflows.
jobs	Stores submitted job records and their execution metadata.
job_executions	Stores execution attempts, statuses, and runtime error details.
files	Stores uploaded files and generated file metadata.
folders	Stores user storage folder hierarchy and paths.

datasets	Stores workflows published to the community catalog.
pipelines	Stores visual pipelines created by users.
pipeline_votes	Stores user votes on shared pipelines.
community_workflows	Stores workflows published to the community catalog.
saved_workflows	Stores workflows users saved from the community.
forum_threads	Stores forum discussion topics created by users.
forum_answers	Stores answers posted under forum threads.
forum_comments	Stores comments on threads and answers.
forum_thread_votes	Stores user votes on forum threads.
forum_comment_votes	Stores user votes on forum comments.
moderation_reports	Stores user reports for pipelines and forum content.
invitation_codes	Stores registration invite codes and their usage state.

Table 1 - Entity-Relationship Diagram

3.4 Object Storage Layer

AWS S3 via boto3 client handles the object storage. Each user has its own folder in the filesystem. Folders are created on new user registration. File S3 keys follow a namespacing convention reflecting their role (job input, output, intermediate, log, or folder file). Downloads are served via presigned URLs with configurable expiry, keeping storage credentials server-side.

4. Development / Implementation Details

4.1 Technology Stack

Layer	Technology	Version	Purpose
Frontend framework	React + TypeScript	18.2 / 5.2	Component-based SPA
Frontend build	Vite	5.0	Fast development server and production bundler
Visual pipeline editor	ReactFlow	11.10	DAG canvas with custom block types
Client routing	React Router	6.20	Client-side navigation across 18 pages
HTTP client	Axios	1.6	Typed API calls from the browser
Frontend server	nginx	stable	Serves built SPA; proxies /api/ to FastAPI
Backend framework	FastAPI + uvicorn	0.104 / 0.24	Async REST API with OpenAPI schema generation
Backend language	Python	3.9+	All 35+ service modules
Schema validation	Pydantic	2.0	Request and response model validation
Database driver	psycopg2	latest	Raw connection pool
Object storage client	boto3	latest	S3 interface
Container runtime	Docker Engine + Compose	24 / 2.x	Local stack and tool image builds
Container orchestration	Kubernetes (Minikube)	1.28	Job-per-tool execution; namespace isolation

K8s Python client	kubernetes	latest	Programmatic Job submission and lifecycle management
Docker SDK	docker	7	Used to create tool images and containerization of the system components.
Workflow engine	Nextflow DSL2	vendored	Alternative execution backend
Authentication	PyJWT + bcrypt	latest	JWT session management and password hashing
Email	Gmail	latest	Verification codes, 2FA, password reset, job notifications

Table 2 - Tech-Stack Diagram

4.2 Backend Service Modules

The backend is responsible for managing all the other parts of the system. It communicates with the frontend with a REST API in order to interact with system functionalities. The backend is organized into multiple subservices, each responsible for a specific functionality. The user management services handle authentication, account management, and access control. Passwords are stored using cryptographic hashing, while authentication tokens allow secure communication between the client and server. The job management services coordinate the creation, monitoring, and termination of job executions. Each job has a specific configuration and input dataset. Job statuses are made available to the user through the user interface. The data management services handle the upload, storage, and retrieval of datasets. Large datasets are transferred using multipart transfer to ensure reliable data transfer. Pipeline management services process pipelines created in the visual pipeline builder. These services validate pipeline structure and input compatibility, and convert the pipeline into an executable form. Together, these backend modules enable the system to manage complex bioinformatics pipelines while maintaining system stability and security.

4.3 Frontend Implementation

The frontend was implemented as a web application that provides users with an environment for configuring and executing genomic analysis workflows. The interface provides, user authentication and account management, dataset upload and storage management, visual pipeline construction, job configuration, job monitoring and progress visualization, and output visualization and download. The visual pipeline builder allows users to create pipelines by connecting tool blocks with input, result, checkout, and other tool blocks, and connections between blocks define the data flow between workflow steps. The interface also includes monitoring pages that display information about workflow progress. Users can see the status of each workflow step, review execution logs, and access output files.

4.4 Workflow Execution

The execution of a workflow begins when a user submits a configured analysis pipeline for execution. A pipeline can be created either by constructing a pipeline in the visual pipeline builder or by selecting and configuring tools from the tool list. After defining the workflow, the user creates a job by assigning input datasets from their storage space. The system verifies that the selected input files match the required data types expected by the tools included in the workflow. Once the job is started, the workflow manager converts the pipeline configuration into an executable workflow. Then, each tool is executed inside an isolated pod to prevent conflicts between different jobs. The container orchestrator schedules these pods based on the available computational resources. During execution, the system checks the status of the running pods and collects execution logs. These logs are saved and presented to the user. When a tool execution completes successfully, the generated output files are transferred to the user’s storage area. After all tool executions are completed, all output files are compressed into one ZIP file and made available for download. If any workflow step fails, the system marks the job as failed,, and informs the user.

4.5 Data Storage

CASSIE filesystem manages two types of data: user datasets and job outputs. User datasets are uploaded through the storage page and stored in a folder inside for each user in the system bucket. During job execution, intermediate files are temporarily stored in the execution environment. When a tool execution finishes, only the final output files are saved to the user's folder inside the system bucket, and intermediate files are removed from the system. After all tool executions are completed, all output files are compressed into one ZIP file and made available for download.

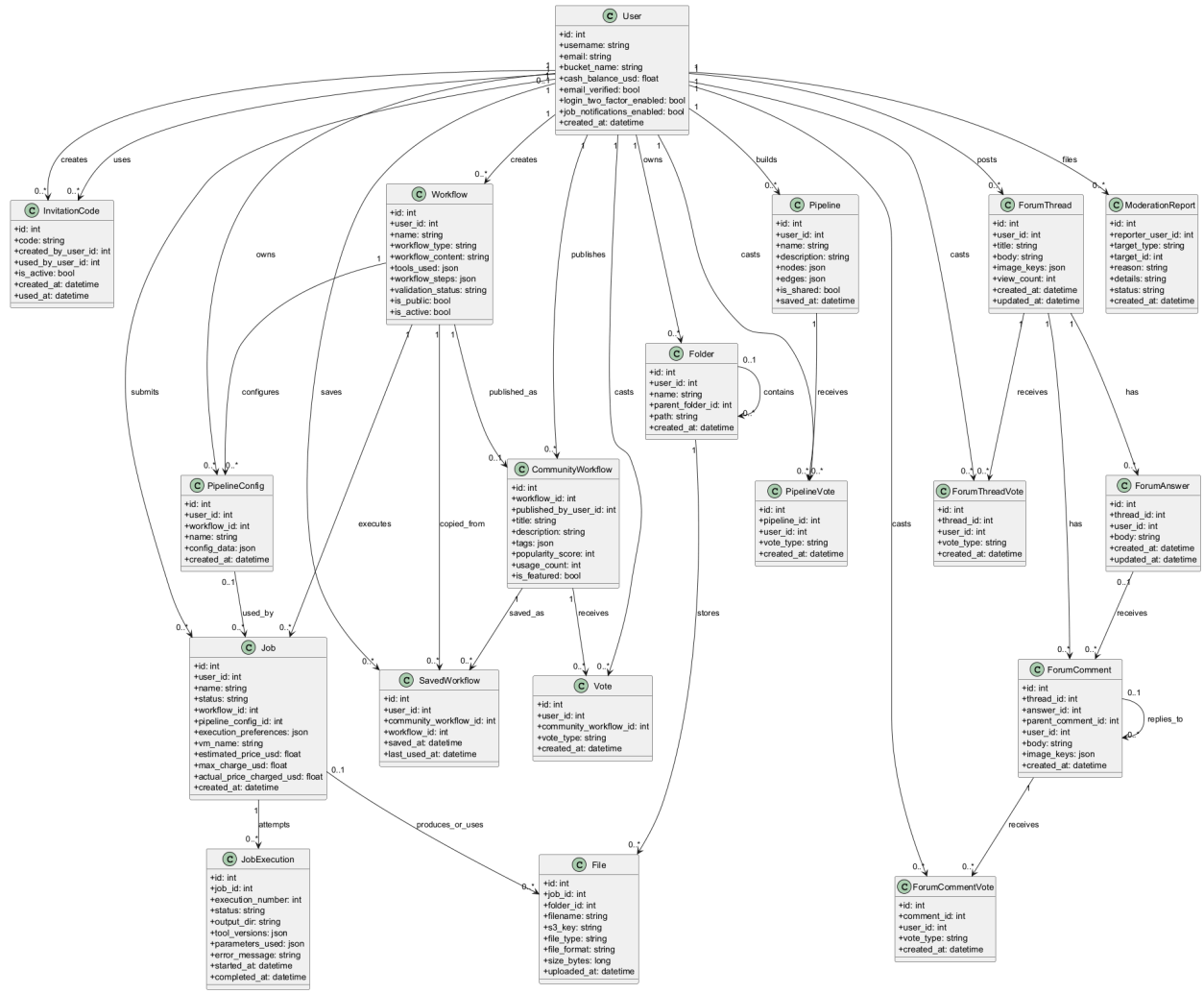

Figure 2 - Class Diagram of the System

4.6 GenomicsTools

Tool	Assembly Stage	Read Types	Key Outputs
FastQC	Quality control	Any (FASTQ, BAM)	HTML report, per-base quality plots, adapter content analysis
GenomeScope2	Genome properties estimation	Short-read or HiFi k-mer histograms	Genome size, heterozygosity, repeat content estimates
SPAdes	Short-read assembly	Illumina (paired-end, mate-pair, single)	FASTA contigs and scaffolds, assembly graph (GFA)
metaSPAdes	Metagenomic assembly	Illumina paired-end metagenomics	FASTA contigs for mixed microbial communities
HiFiasm	Long-read assembly	PacBio HiFi (CCS)	GFA assembly graph; phased haplotype contigs
Verkko	Phased long-read assembly	PacBio HiFi + Oxford Nanopore	Telomere-to-telomere phased haplotype assemblies
QUAST	Assembly quality assessment	Any FASTA (+ optional reference)	HTML report, N50/NG50, misassembly statistics
BUSCO	Gene completeness assessment	Any genome FASTA or protein set	Completeness score, duplication/fragmentation/missing rates
CAT	Contig taxonomic annotation	Any FASTA + database	Per-contig taxonomic classification
Liftoff	Reference-based annotation transfer	Any FASTA + reference GFF/GTF	GFF3 annotation lifted to the target assembly

Meryl	k-mer counting	Short-read or long-read sequencing data	k-mer frequency database used for genome analysis
Merqury	k-mer-based assembly evaluation	PacBio HiFi reads + FASTA assembly	QV score, k-mer completeness, phase block statistics

Table 3 - Genomics Tools Diagram

4.7 Runtime and Cost Estimator

The Runtime and Cost Estimator provides users with estimations of both expected time and cost before execution. Each tool in the platform carries a baseline runtime constant that represents its expected duration according to the reference. This baseline constant is determined by four factors: Firstly, the VM partition multiplier, which allocates more CPU across more concurrent jobs, slowing individual tool executions. Second, a size factor determines the runtime based on the actual input size relative to each tool's referencing input size. Third, a compression penalty is applied when inputs are compressed formats to include additional decompression costs. Lastly, a fixed orchestration overhead that is scaled down for smaller, lighter jobs is added. For the tools that do not have inputs explicitly assigned by the user, the estimator infers input sizes by generating estimated output sizes from upstream tools. Cost is calculated by multiplying the estimated runtime in minutes by the price per minute of the selected VM tier.

5. Test Cases and Results

Test ID	T-01	**Category**	Functional	**Severity**	Major
Objective	Successful User Registration				
Steps	1. Navigate to Sign Up page. 2. Enter valid email address. 3. Enter and confirm a strong password. 4. Click 'Register' button.				
Expected	User account is created; user is redirected to the Home Page.				
Date	25.04.2026	**Result**	User account succesfully created. User is redirected to the Home Page		

Fig. 3: Test Case 1

Test ID	T-02	**Category**	Functional	**Severity**	Critical
Objective	User Login with Valid Credentials				
Steps	1. Navigate to Login page. 2. Enter registered email. 3. Enter correct password. 4. Click 'Login' button.				
Expected	Successful authentication; user is redirected to the Home Page with active session.				
Date	25.04.2026	**Result**	User successfully logged in and redirected to the Home Page with active session		

Fig. 4: Test Case 2

Test ID	T-03	Category	Functional	Severity	Major
Objective	Login Validation - Incorrect Password				
Steps	1. Navigate to Login page. 2. Enter valid email. 3. Enter an incorrect password. 4. Click 'Login'.				
Expected	Authentication fails; clear error message 'Invalid email or password' is displayed.				
Date	25.04.2026	Result	Authentication failed; 'Invalid email or password' was displayed.		

Fig. 5: Test Case 3

Test ID	T-04	Category	Security	Severity	Major
Objective	JWT Session Persistence				
Steps	1. Log into CASSIE. 2. Close the browser tab. 3. Re-open the browser and navigate back to CASSIE. 4. Check if session is active.				
Expected	User remains logged in.				
Date	25.04.2026	Result	User stayed loged in. User retained their token.		

Fig. 6: Test Case 4

Test ID	T-05	Category	Functional	Severity	Minor
Objective	Password Confirmation Mismatch				
Steps	1. Navigate to Sign Up page. 2. Fill email and password. 3. Enter a different password in 'Confirm Password'. 4. Click 'Register'.				
Expected	Error message displayed stating passwords do not match; account not created.				
Date	25.04.2026	Result	The system displayed the 'Password does not match' message.		

Fig. 7: Test Case 5

Test ID	T-06	Category	Functional	Severity	Critical
Objective	Upload Valid Genomic Files (FASTQ)				
Steps	1. Navigate to Workflow Configuration. 2. Click 'Upload Data'. 3. Select a valid .fastq file. 4. Wait for upload to finish.				
Expected	File is uploaded successfully and appears in the user's data list.				
Date	25.04.2026	Result	User was able to upload files until it reached their limit.		

Fig. 8: Test Case 6

Test ID	T-07	Category	Integration	Severity	Major
Objective	Direct Cloud-to-S3 Data Transfer				
Steps	1. Select 'Import from Cloud Storage'. 2. Provide External Cloud credentials/link. 3. Select target genomic files. 4. Click 'Transfer'.				
Expected	Files are transferred directly to CASSIE's S3 isolated directory without local download.				
Date	25.04.2026	Result	Files from Google Drive are uploaded to the filesystem without local download.		

Fig. 9: Test Case 7

Test ID	T-08	Category	Security	Severity	Critical
Objective	Data Isolation - Bucket Access Control				
Steps	1. Log in as User A. 2. Upload a file. 3. Log out and log in as User B. 4. Navigate to Data list.				
Expected	User B cannot see or access User A's uploaded files.				
Date	25.04.2026	Result	User was not able to reach the files of other users.		

Fig. 10: Test Case 8

Test ID	T-09	**Category**	Functional	**Severity**	Major
Objective	Delete Uploaded Dataset				
Steps	1. Navigate to Profile/Manage Datasets. 2. Click 'Delete' next to a file. 3. Confirm deletion in the pop-up. 4. Refresh page.				
Expected	File is removed from the UI and from the underlying S3 storage.				
Date	25.04.2026	**Result**	Files successfully removed from S3 bucket after a file delete.		

Fig. 11: Test Case 9

Test ID	T-10	**Category**	Functional	**Severity**	Major
Objective	Manual Workflow Construction				
Steps	1. Click 'Build Pipeline'. 2. Drag 'QC' component to canvas. 3. Drag 'Assembly' component and connect. 4. Click 'Save Pipeline'.				
Expected	Workflow structure is saved and visible in 'Saved Pipelines' on Profile Page.				
Date	25.04.2026	**Result**	Pipeline is saved after dragging and connecting additionalinput and output blocks.		

Fig. 12: Test Case 10

Test ID	T-11	Category	Functional	Severity	Major
Objective	Automated Tool Selection via Goals				
Steps	1. Select 'Configure' with automation. 2. Provide analysis goals. 3. Answer high-level questions. 4. Review proposed tools.				
Expected	System automatically selects appropriate tools based on user answers.				
Date	25.04.2026	Result	System showed possible tools according to the answers.		

Fig. 13: Test Case 11

Test ID	T-12	Category	Functional	Severity	Major
Objective	Dynamic Parameter Display				
Steps	1. Go to Workflow Configuration. 2. Select 'Tool A' from the list. 3. Select 'Tool B'. 4. Check parameter fields.				
Expected	UI only shows parameters specific to Tool A and Tool B; unrelated parameters are hidden.				
Date	25.04.2026	Result	UI did not show any parameters when job is created without using pipeline builder.		

Fig. 14: Test Case 12

Test ID	T-13	Category	Functional	Severity	Major
Objective	Validation of Missing Parameters				
Steps	1. Configure a workflow. 2. Leave a required parameter empty. 3. Click 'Validate/Execute'. 4. Check error markers.				
Expected	Validation fails; UI highlights missing field with a 'required' warning.				
Date	25.04.2026	Result	Validation failed. UI did not allow the user to move forward in job creation steps.		

Fig. 15: Test Case 13

Test ID	T-14	Category	Functional	Severity	Major
Objective	Resource Estimation Accuracy				
Steps	1. Finalize workflow config. 2. View Resource Estimation section. 3. Change the dataset size. 4. Observe the estimated time/price.				
Expected	Estimates cost and time based on the ML model's prediction.				
Date	25.04.2026	Result	Time and related cost estimated relatively correctly.		

Fig. 16: Test Case 14

Test ID	T-15	**Category**	Functional	**Severity**	Major
Objective	Manual Machine Selection				
Steps	1. View recommended Machine instance. 2. Open dropdown to select a different class. 3. Click 'Confirm Selection'.				
Expected	Workflow is updated to run on the user-selected instance class.				
Date	25.04.2026	**Result**	Job ran on the selected class.		

Fig. 17: Test Case 15

Test ID	T-16	**Category**	Integration	**Severity**	Critical
Objective	Workflow Submission to Nextflow				
Steps	1. Complete configuration. 2. Click 'Execute Pipeline'. 3. Navigate to 'Active Jobs'. 4. Verify job status is 'Queued/Running'.				
Expected	Backend sends parameters to Nextflow; Nextflow initiates the pipeline on Kubernetes.				
Date	25.04.2026	**Result**	Kubernetes started the first job pod successfully.		

Fig. 18: Test Case 16

Test ID	T-17	Category	Security	Severity	Critical
Objective	Kubernetes Namespace Isolation				
Steps	1. Start Job One. 2. Start Job Two as a different user. 3. Check Kubernetes API for namespaces. 4. Verify separate namespaces.				
Expected	Each job runs in a dedicated, isolated Kubernetes namespace.				
Date	25.04.2026	Result	Each tool had its own pod, achieving job isolation.		

Fig. 19: Test Case 17

Test ID	T-18	Category	Functional	Severity	Major
Objective	Job Termination/Cancellation				
Steps	1. Start a running job. 2. Navigate to Jobs Page. 3. Click 'Cancel Workflow'. 4. Confirm action.				
Expected	Workflow stops; Kubernetes pods for that job are terminated; status changes to 'Cancelled'.				
Date	25.04.2026	Result	Job successfully cancelled. Resources are freed. Status changed to 'Cancelled'.		

Fig. 20: Test Case 18

Test ID	T-19	Category	Reliability	Severity	Minor
Objective	Intermediate File Cleanup				
Steps	1. Run a workflow to completion. 2. Access the Kubernetes namespace/storage. 3. Verify presence of final outputs vs intermediate files.				
Expected	Temporary/intermediate files are removed; only final tool outputs remain.				
Date	25.04.2026	Result	After the tool was successfully executed, only outputs remained; all other files were removed.		

Fig. 21: Test Case 19

Test ID	T-20	Category	Functional	Severity	Major
Objective	Real-time Execution Status Updates				
Steps	1. Start a job. 2. Stay on the Jobs Page. 3. Watch the progress bar/status. 4. Wait for step change.				
Expected	Status updates from 'Working' to 'Completed' for each step in near-real-time via UI.				
Date	25.04.2026	Result	After a tool finished, it got labelled as 'Completed'. Tools that are waiting are labelled as 'Waiting For Resources' or 'Waiting for Dependency'.		

Fig. 22: Test Case 20

Test ID	T-21	Category	Functional	Severity	Major
Objective	Download Results as Compressed Archive				
Steps	1. Navigate to Completed Jobs. 2. Click 'Download All Results'. 3. Open the downloaded .zip file. 4. Check contents.				
Expected	Download is successful; archive contains all relevant output files.				
Date	25.04.2026	Result	All output files are downloaded as a single zip file.		

Fig. 23: Test Case 21

Test ID	T-22	Category	Functional	Severity	Major
Objective	View Workflow Execution Logs				
Steps	1. Go to a completed job. 2. Click 'View Logs'. 3. Scroll through tool output. 4. Verify readability.				
Expected	Tool logs are accessible for debugging and transparency.				
Date	25.04.2026	Result	Each tool showed their Kubernetes logs successfully.		

Fig. 24: Test Case 22

Test ID	T-23	Category	Functional	Severity	Major
Objective	Quality Control Visualization				
Steps	1. Click 'Quality Control Results' for a job. 2. Observe displayed diagrams. 3. Verify data axes/labels.				
Expected	UI displays visual diagrams as shown in mock-ups.				
Date	25.04.2026	Result	UI displayed visual diagrams as shown in mock-ups.		

Fig. 25: Test Case 23

Test ID	T-24	Category	Functional	Severity	Major
Objective	Resubmit Failed Job with Adjusted Params				
Steps	1. Identify a 'Failed' job. 2. Click 'Adjust & Resubmit'. 3. Change a parameter. 4. Execute again.				
Expected	System re-runs workflow without requiring data re-upload; new job ID created.				
Date	25.04.2026	Result	System allowed user to restart a failed job by bringing the configuration of the job and allow changes on that configuration.		

Fig. 26: Test Case 24

Test ID	T-25	Category	Functional	Severity	Major
Objective	Publish Workflow to Community				
Steps	1. Open a saved workflow. 2. Click 'Publish to Community'. 3. Enter metadata (tags, description). 4. Submit.				
Expected	Workflow appears in the public Community catalog; no private datasets are linked.				
Date	25.04.2026	Result	Workflow appeared in the public Community catalog; no private datasets were linked to it.		

Fig. 27: Test Case 25

Test ID	T-26	Category	Functional	Severity	Minor
Objective	Search/Filter Community Workflows				
Steps	1. Navigate to Community Page. 2. Type 'Assembly' in search. 3. Apply a 'Highly Voted' filter. 4. Review results.				
Expected	List updates to show only workflows matching the keyword and filter criteria.				
Date	25.04.2026	Result	Workflows that have 'Assembly' keyword in them were filtered and ordered according to their popularity.		

Fig. 28: Test Case 26

Test ID	T-27	Category	Functional	Severity	Major
Objective	Import Community Workflow				
Steps	1. Select a workflow from Community. 2. Click 'Import Pipeline'. 3. Navigate to 'My Workflows'. 4. Check for the copy.				
Expected	A private, editable copy of the workflow is added to the user's workspace.				
Date	25.04.2026	**Result**	A private, editable copy of the workflow were added to the user's workspace.		

Fig. 29: Test Case 27

Test ID	T-28	Category	Functional	Severity	Minor
Objective	Voting on Community Workflows				
Steps	1. Open a community workflow detail. 2. Click 'Upvote'. 3. Refresh page. 4. Check vote count.				
Expected	Vote count increases by 1; system maintains attribution to original author.				
Date	25.04.2026	**Result**	Vote count increased by 1; system maintained attribution to original author.		

Fig. 30: Test Case 28

Test ID	T-29	**Category**	Usability	**Severity**	Minor
Objective	Contextual Help Tooltips				
Steps	1. Navigate to Workflow Config. 2. Hover over a '?' icon next to a tool parameter. 3. Read the popup.				
Expected	Help text explains the parameter's purpose and expected values.				
Date	25.04.2026	**Result**	In the Visual Pipeline Builder user clicked the '...' and Edit buttons consecutively on a tool block. UI showed each possible parameter and their explanation.		

Fig. 31: Test Case 29

Test ID	T-30	**Category**	Performance	**Severity**	Major
Objective	Concurrent Job Handling				
Steps	1. Open multiple tabs. 2. Initiate 5 different workflows simultaneously. 3. Monitor 'Active Jobs'.				
Expected	System handles all requests without crashing; Kubernetes schedules pods per resource availability.				
Date	25.04.2026	**Result**	Due to resource/storage limitations, the user was able to start only 3 jobs consecutively.		

Fig. 32: Test Case 30

Test ID	T-31	**Category**	Reliability	**Severity**	Major
Objective	Data Durability - Storage Outage				
Steps	1. During an upload, simulate a brief S3 connection loss. 2. Check if system retries. 3. Verify file integrity after reconnect.				
Expected	Retry mechanism prevents data loss; partial uploads are handled gracefully.				
Date	25.04.2026	**Result**	Partial uploads are not supported.		

Fig. 33: Test Case 31

Test ID	T-32	**Category**	Security	**Severity**	Critical
Objective	Unauthorized Endpoint Access				
Steps	1. Copy a Job ID from User A. 2. Log in as User B. 3. Manually enter the URL for Job A's result. 4. Observe response.				
Expected	System redirects to home; unauthorized access is blocked.				
Date	25.04.2026	**Result**	User could not acces most of the features without logging in.		

Fig. 34: Test Case 32

Test ID	T-33	Category	Functional	Severity	Major
Objective	Reproducibility with Identical Parameters				
Steps	1. Run a workflow. 2. Record results. 3. Re-run identical workflow with same dataset and parameters. 4. Compare outputs.				
Expected	Outputs are scientifically identical, ensuring reproducibility.				
Date	25.04.2026	Result	Outputs were identical.		

Fig. 35: Test Case 33

Test ID	T-34	Category	Integration	Severity	Major
Objective	Docker Container Version Locking				
Steps	1. Inspect a 'Workflow Step'. 2. Check the Docker image tag. 3. Ensure it's not using 'latest'. 4. Run job.				
Expected	System uses specific version tags to prevent changes in results over time.				
Date	26.04.2026	Result	System does not use specific version tags.		

Fig. 36: Test Case 34

Test ID	T-35	Category	Functional	Severity	Minor
Objective	Empty Dataset Selection Validation				
Steps	1. Create a workflow. 2. Click 'Execute' without selecting any data. 3. Observe UI behavior.				
Expected	Execution is blocked; user is prompted to 'Select at least one dataset'.				
Date	26.04.2026	Result	UI alerted the user for uploading data before job start.		

Fig. 37: Test Case 35

Test ID	T-36	Category	Functional	Severity	Major
Objective	Large Data Volume Support				
Steps	1. Upload a 50GB genomic dataset. 2. Configure a workflow. 3. Initiate execution. 4. Monitor storage and pod status.				
Expected	System handles high-volume data and allocates appropriate disk space in Kubernetes.				
Date	26.04.2026	Result	The system allowed high-volume data.		

Fig. 38: Test Case 36

Test ID	T-37	Category	Usability	Severity	Major
Objective	Readable Error Messages for CLI Failures				
Steps	1. Provide invalid input that triggers a tool-level error (not UI). 2. View the job error report. 3. Read the message.				
Expected	Error is translated into a user-friendly explanation with guidance.				
Date	26.04.2026	Result	UI explained what went wrong in a human-readable way.		

Fig. 39: Test Case 37

Test ID	T-38	Category	Functional	Severity	Minor
Objective	Save/Edit User Profile Information				
Steps	1. Go to Profile Page. 2. Edit 'Name' field. 3. Click 'Update Profile'. 4. Refresh page.				
Expected	Database updates; new name is displayed correctly in the UI.				
Date	26.04.2026	Result	Visible name of the user successfully edited and shown on the UI.		

Fig. 40: Test Case 38

Test ID	T-39	Category	Functional	Severity	Major
Objective	Component Connection Validation				
Steps	1. In Pipeline Builder, try to connect 'Annotation' output to 'QC' input. 2. Observe the canvas behavior.				
Expected	Invalid logic connections are blocked.				
Date	26.04.2026	Result	The system automatically converts the wrong input blocks to their correct counterparts.		

Fig. 41: Test Case 39

Test ID	T-40	Category	Performance	Severity	Minor
Objective	Database Query Performance (Jobs List)				
Steps	1. Populate a test account with multiple previous jobs. 2. Navigate to Jobs Page. 3. Measure load time.				
Expected	Jobs list loads in < 2 seconds through efficient SQL pagination.				
Date	26.04.2026	Result	The jobs list loaded fast with efficient pagination.		

Fig. 42: Test Case 40

Test ID	T-41	**Category**	Functional	**Severity**	Minor
Objective	Handling of Concurrent Same-File Uploads				
Steps	1. Start uploading a FASTQ file. 2. Immediately start another upload of the same file. 3. Check for duplicates.				
Expected	System handles the conflict.				
Date	26.04.2026	**Result**	Duplicates did not create conflict.		

Fig. 43: Test Case 41

Test ID	T-42	**Category**	Compatibility	**Severity**	Major
Objective	Browser Compatibility - Safari/Firefox/Chrome				
Steps	1. Open Pipeline Builder in 3 different modern browsers. 2. Test drag-and-drop functionality in each.				
Expected	Core functionalities work identically across all supported browsers.				
Date	26.04.2026	**Result**	All main features successfully worked on 3 different browers.		

Fig. 44: Test Case 42

Test ID	T-43	Category	Functional	Severity	Major
Objective	Reset Password via Email				
Steps	1. Click 'Forgot Password' on Login page. 2. Provide valid email. 3. Follow the link in the simulated email. 4. Change password.				
Expected	Password is updated successfully; user can log in with the new password.				
Date	26.04.2026	Result	Password was successfully changed.		

Fig. 45: Test Case 43

Test ID	T-44	Category	Documentation	Severity	Minor
Objective	Verify README/Instructions Link				
Steps	1. Locate 'Help/Docs' link in the sidebar. 2. Click and navigate to documentation page. 3. Check for 'Getting Started' guide.				
Expected	Documentation is accessible and provides relevant setup instructions for new researchers.				
Date	26.04.2026	Result	Documentation is available.		

Fig. 46: Test Case 44

Test ID	T-45	Category	Security	Severity	Major
Objective	Multi-Factor Authentication (MFA) Setup				
Steps	1. Navigate to 'User Profile'. 2. Select 'Security Settings'. 3. Click 'Enable 2FA'. 4. Enter the security code sent via email.				
Expected	MFA is successfully linked; system prompts for code on next login.				
Date	26.04.2026	Result	MFA was successfully linked; system prompted for a code on the next login.		

Fig. 47: Test Case 45

Test ID	T-46	Category	Security	Severity	Major
Objective	Account Session Revocation				
Steps	1. Log into CASSIE on two different browsers. 2. In Browser A, go to 'Active Sessions'. 3. Click 'Log out of all other sessions'. 4. Attempt to navigate in Browser B.				
Expected	Browser B is automatically redirected to the login page.				
Date	26.04.2026	Result	This did not happen as it should not.		

Fig. 48: Test Case 46

Test ID	T-47	Category	Reliability	Severity	Critical
Objective	DB Failover Recovery				
Steps	1. Start a workflow job. 2. Simulate a database primary node failover (admin action). 3. Wait for the failover to complete. 4. Refresh the job status page.				
Expected	The job status remains accurate and the UI recovers connection to the DB.				
Date	26.04.2026	Result	Job status could not recovered gracefully.		

Fig. 49: Test Case 47

Test ID	T-48	Category	Functional	Severity	Major
Objective	Custom Tool Container Import				
Steps	1. Go to 'Admin Tool Settings'. 2. Click 'Add Custom Docker Image'. 3. Enter the Docker Hub URI. 4. Define the command-line interface (CLI) mapping.				
Expected	Tool is successfully indexed and appears in the workflow builder.				
Date	26.04.2026	Result	Tool changes on the system made when EC2 instance were stopped momentarily.		

Fig. 50: Test Case 48

Test ID	T-49	Category	Functional	Severity	Major
Objective	Automated Storage Cleanup Policy				
Steps	1. Go to 'Settings'. 2. Set 'Auto-delete intermediate files' to '7 days'. 3. Run a workflow. 4. Check file availability after the specified period.				
Expected	Intermediate files are automatically purged from S3 after 7 days.				
Date	26.04.2026	**Result**	Intermediate files are stored in the storage now.		

Fig. 51: Test Case 49

Test ID	T-50	Category	Functional	Severity	Major
Objective	User Account Deletion				
Steps	1. Go to 'Profile Settings'. 2. Click 'Delete My Account'. 3. Confirm identity via password. 4. Click 'Permanently Delete'.				
Expected	Account is deactivated; personal data and files are queued for purging.				
Date	26.04.2026	**Result**	After the user entered the deletion code that was delivered to their email, their account was deleted.		

Fig. 52: Test Case 50

Test ID	T-51	Category	Performance	Severity	Minor
Objective	Export Large Result Logs				
Steps	1. Locate a job with a 100MB+ log file. 2. Click 'Download Full Log'. 3. Monitor the download speed. 4. Verify the integrity of the downloaded file.				
Expected	Log file downloads without truncation or connection timeout.				
Date	26.04.2026	Result	Finding a 100MB+ log file was not possible; however, the user was able to download log files for each tool.		

Fig. 53: Test Case 51

Test ID	T-52	Category	Functional	Severity	Minor
Objective	Notification Preferences				
Steps	1. Go to 'Notifications Settings'. 2. Toggle 'Email on Job Completion' to OFF. 3. Run a workflow. 4. Verify that no email is received.				
Expected	System respects the user preference and only sends requested alerts.				
Date	26.04.2026	Result	System sent emails to the user when user selected related checkboxes and did not send email when those checkboxes were not selected.		

Fig. 54: Test Case 52

Test ID	T-53	Category	Security	Severity	Major
Objective	Password Complexity Enforcement				
Steps	1. Go to 'Change Password'. 2. Attempt to set password to '123456'. 3. Attempt with no special characters. 4. Enter a valid complex password.				
Expected	Weak attempts are rejected with clear requirements; valid one is accepted.				
Date	26.04.2026	**Result**	Weak password attempts were rejected.		

Fig. 55: Test Case 53

Test ID	T-54	Category	Performance	Severity	Major
Objective	UI Responsiveness during Heavy Upload				
Steps	1. Start a multi-file batch upload (20+ files). 2. While uploading, navigate to the 'Jobs' page. 3. Edit a workflow. 4. Return to the 'Data' page.				
Expected	The UI remains responsive and does not lag during background uploads.				
Date	26.04.2026	**Result**	The UI remained responsive and did not lag during background uploads.		

Fig. 56: Test Case 54

Test ID	T-55	Category	Functional	Severity	Minor
Objective	Interactive Tool Modal Validation				
Steps	1. Open a tool configuration modal. 2. Enter a string into a numeric-only field. 3. Attempt to click 'Apply'. 4. Correct the value to a number.				
Expected	Apply' is disabled and an error message appears until input is valid.				
Date	26.04.2026	Result	An error message is displayed.		

Fig. 57: Test Case 55

Test ID	T-56	Category	Functional	Severity	Minor
Objective	Search by File Extension				
Steps	1. Go to the Data Manager. 2. Type ".vcf" in the search bar. 3. Verify the results list. 4. Clear search and type ".fastq".				
Expected	Only files matching the specific extension are displayed in each case.				
Date	26.04.2026	Result	Only files matching the extension and has a name consisting the extension (dot included) displayed.		

Fig. 58: Test Case 56

Test ID	T-57	Category	Reliability	Severity	Critical
Objective	Kubernetes Node Scaling				
Steps	1. Submit 10 heavy genomic pipelines simultaneously. 2. Monitor the cluster via Admin dashboard. 3. Observe new nodes spinning up. 4. Verify jobs complete.				
Expected	The cluster scales out to meet demand and scales back in after completion.				
Date	26.04.2026	Result	This was not possible due to resource contraints.		

Fig. 59: Test Case 57

6. Maintenance Plan and Details

The CASSIE system runs on an AWS EC2 instance and deal with large data resulting in a need for continuous maintenance. The highest priority maintenance targets are EC2 instance workload, database and filesystem resource loads and genomics tool updates. Maintenance will happen with prior notices and warnings in order to prevent disruptions as much as possible.

6.1 EC2 Workload

The EC2 instance hosting the platform will be monitored monthly to maintain stable system performance. Maintenance activities include tracking CPU usage, memory consumption, and active computational jobs running on the instance. When workload demand increases and jobs start to fail, administrators will change instance configurations to achieve more stable performance. Additionally, the operating system will be updated.

6.2 Resource Loads

The CASSIE system processes large genomic datasets and therefore requires monitoring of both database activity and filesystem storage. Continuous monitoring ensures that the platform remains responsive and prevents interruptions caused by storage or database overloads.

6.2.1 Database Loads

Maintaining tasks focused on monitoring query response time and storage growth. Database structures are reviewed monthly to maintain efficient query performance as the dataset grows. If the number of users or job executions increases significantly, database resources will be scaled to handle higher loads while preserving system responsiveness.

6.2.2 Filesystem Loads

Genome sequencing files and intermediate workflow outputs can be extremely large, which makes filesystem management a critical maintenance task. Filesystem health checks are conducted monthly to detect corrupted files or storage failures that could affect workflow execution. In case of a resource shortage, S3 bucket size will be increased while not affectingthe contents in the bucket to preserve system responsiveness.

6.3 Genomics Tools

CASSIE integrates multiple genomics tools for genome assembly, quality control, and annotation. Because bioinformatics tools evolve rapidly, catching up to those evolutions is important for ensuring the highest accuracy. Maintenance includes monthly evaluation of new tool releases, updating software versions when improvements are available, and validating that updated tools remain compatible with existing workflows. Updates will be tested in a local test environment before deployment to avoid disruptions in production pipelines.

7. Other Project Elements

7.1 Consideration of Various Factors in Engineering Design

7.1 Constraints

During the development of CASSIE, various subjects affecting the public are considered, and each one is given a priority level between 1 (lowest) and 10 (highest), indicating the effects of CASSIE on that particular consideration.

Public Health Considerations - Priority 3

CASSIE does not directly affect public health outcomes as it is proposed solely as a tool to support researchers in analyzing their data effectively and efficiently. Any scientific decision-making is left up to the researchers.

Public Safety Considerations - Priority 6

CASSIE's public safety considerations are addressed through design choices covering the secure execution of computational workflows and prevention of malicious or unintended system behaviour. Since CASSIE is a system that enables users to execute complex pipelines, safety measures regarding resources, unauthorized access, or execution of unsafe code must be implemented. The platform provides isolation between workflow executions, controlled access to computational resources, and strict permission boundaries to reduce the risk of system misuse.

Public Welfare Considerations - Priority 5

Public welfare is addressed through equitable and responsible access to computational resources. CASSIE is designed to support scientific research by simplifying individual software infrastructure work performed by researchers. The system imposes constraints on resource usage and execution scheduling to prevent a small number of users from establishing monopolies over the platform and disrupting the shared infrastructure. Thereby, CASSIE ensures fair access and maintains service availability for all users.

Global Considerations - Priority 7

Global considerations are limited, as CASSIE is designed to be used for educational or research purposes and does not impose the use of unfamiliar or poorly established tools in the analysis process.

Cultural Considerations - Priority 2

Cultural considerations are minimal for CASSIE, as the platform provides technical functionality. There exists no visible risk of any kind of cultural violations, as the system does not include any culturally sensitive content.

Social Considerations - Priority 4

Social interactions are only possible via the community page, and constraints and monitoring are required to prevent any plagiarism, misleading workflow information, manipulation in the popularity metrics, and the language used in the workflow details. CASSIE has a dedicated reporting system and communication channels to prevent any kind of misconduct.

Environmental Considerations - Priority 5

Environmental considerations are addressed through efficient use of computational resources. Introducing time and cost estimation helps prevent unnecessary use of computation units and helps reduce the energy consumption during executions.

Economic Considerations - Priority 7

Economic constraints are addressed by time and cost estimations. The proposed estimation functionalities help users to estimate the cost of execution beforehand. Also, different pricing based on different instance classes encourages users to select more cost-efficient options, which reduces redundant computation of the system. Therefore, lower operational costs for both the user and the system can be achieved.

7.1.2 Standards

Software and Engineering Standards:

IEEE 830 — Software Requirements Specification: The SRS and DDR requirements sections follow the structure and intent of IEEE 830-1998, stating requirements in terms of system behaviour rather than implementation details [2].

UML 2.5.1 — Modelling Standards: Applied to all design diagrams in the DDR, including the class diagram (DDR Fig. 1) and subsystem diagram (DDR Fig. 2) [3].

OCI — Open Container Initiative: All Docker images in dockerized_tools/ follow OCI Image Format Specification v1.0.0 and OCI Runtime Specification v1.0.0, ensuring portability across OCI-compliant runtimes [4, 5].

Kubernetes API Standards: Pod, Namespace, and ResourceQuota resources use standard Kubernetes batch/v1 and v1 API conventions [6].

Nextflow DSL2: The Nextflow execution backend uses DSL2 syntax for modular, reusable, independently testable process definitions [7, 8].

RFC 7519 — JSON Web Tokens: Session management uses JWTs with standard claims (sub, exp, iat) and custom claims for user role and identifier.

Bioinformatics Data Format Standards:

FASTQ: Stores raw sequencing reads with per-base quality scores; primary input format for QC and assembly [9].

FASTA: Stores raw sequence data without quality scores; primary output format for assembled contigs; input to annotation tools [10].

GFA (Graphical Fragment Assembly): Graph format for sequence overlaps; output by HiFiasm and SPAdes; consumed by graph-aware tools [11].

GFF3 / GTF: Standard annotation data formats; output by Liftoff and CAT; input to downstream analysis [12, 13].

BAM / CRAM / SAM: Alignment formats; BAM and CRAM are compressed binary versions of SAM; supported as inputs to several tools [14, 15].

BED: Simple tabular interval format for feature coordinates; used by annotation and coverage analysis tools [16].

Compliance with these standards ensures that data produced by CASSIE can be consumed by any tool in the broader bioinformatics ecosystem without format conversion.

7.2 Ethics and Professional Responsibilities

Developing a platform for genomic analysis brings specific ethical and professional responsibilities, particularly regarding data privacy, scientific integrity, and system reliability. Since what is being developed is a platform for genomic analysis, this brings specific ethical and professional responsibilities, especially regarding data privacy and scientific integrity.

Data Privacy and Security

Genomic data is one of the most highly sensitive pieces of information. If it belongs to a person, it is the most personal information about their health. Even if it does not belong to a human being, it is still the property of the researcher. As the architects and engineers of this platform, we have obligations to protect this data. CASSIE handles this by:

Isolation: CASSIE enforces namespace isolation with Kubernetes and folder-level isolation in S3. These precautions are taken to ensure that no user can access another researcher's data.

Access Control: Industry-standard measurements are taken for user authentication, user data integrity, and general security of the CASSIE platform. We implement multi-layer access control covering both the web application and the Kubernetes environment underlying it. JSON Web Tokens (JWT) are utilized in the application layer for stateless user authentication. Authorization in the execution environment is done using Kubernetes Role-Based Access Control (RBAC). RBAC defined using Kubernetes declarative configuration files ensures access to computational resources is done with the Principle of Least Privilege in mind.

Scientific Integrity and Reproducibility

A platform that makes its debut with a claim that it will solve a certain problem regarding scientific tools, of course, must ensure the scientific integrity, meaning results produced are valid, trustworthy, and produced with rigorous standards. Its results should also be reproducible. CASSIE uses containerized tools that are locked to specific versions. This guarantees that results are reproducible. These containerized tools are chosen from peer-reviewed bioinformatics tools. This makes sure all output is derived from proven and established methods.

7.3 Teamwork Details

Collaboration and Management Tools:

GitHub: Used for source code management. Every feature was developed on a separate branch and merged via pull request after peer review, ensuring all code changes are traceable and reviewed before integration.

Jira: Used to break work into manageable tasks and visualise the project board, providing transparency into individual contributions and enabling early detection of workload imbalances.

7.3.1 Contributing and Functioning Effectively on the Team

Ege Şirvan contributed mainly to backend development and system integration tasks. He focused on the communication between different subsystems and designed the overall system architecture. He also selected and configured the genomic tools.

Eren Aslan contributed to the implementation of the cloud-side of the project and container orchestration. He worked on integrating Docker-based tools and Kubernetes orchestration into the platform. He worked on the workflow orchestration logic and integration between the workflow manager and the cloud infrastructure.

Arda Kırcı contributed to the backend development and system architecture design of the project. He participated in system design decisions and helped improve workflow execution management. He also contributed to documentation and design discussions.

Emre Can Yoloğlu mainly worked on the user interface of the project. He focused on building an interface that simplifies complex bioinformatics operations for the average user. In addition to UI implementation, he also contributed to debugging, testing, and integration tasks to ensure communication between the frontend and backend services.

Osman Baktır contributed to documentation tasks. He also contributed to writing the reports and ensuring that the project documentation accurately reflects the implemented system.

7.3.2 Helping Creating a Collaborative and Inclusive Environment

To maintain a collaborative and inclusive environment, Ege Şirvan is actively involved in communication channels, including the team's messaging group and regular meetings. He ensured everyone was informed about the project's progress and scheduled meetings as needed.

Eren Aslan maintained active communication with the team through meetings and online messaging platforms. He regularly shared updates about the development progress and regularly discussed the workflow and cluster structure so that every team member could contribute to them.

Arda Kırcı supported the collaborative environment by actively participating in both online and in-person meetings. He also provided his ideas on various technical problems.

Emre Can Yoloğlu contributed to maintaining a positive and inclusive team atmosphere by openly sharing ideas and participating in brainstorming sessions. He also helped other team members when technical challenges arose during tool integration or workflow configuration.

Osman Baktır contributed to team collaboration by supporting task coordination. He participated in discussions regarding task assignments and helped maintain a cooperative and respectful working environment.

7.3.3 Taking Lead Role and Sharing Leadership on the Team

Throughout the project, leadership responsibilities were shared among the team members depending on the tasks. Different members took the lead in areas where they had more experience and were responsible for resulting in a balanced workload for each member.

Ege Şirvan took the lead role in designing the overall software architecture and selection/implementation of genomic tools.

Eren Aslan assumed the leading role in the cloud-side infrastructure of the project. He led the implementation of container orchestration mechanisms and coordinated the integration of Docker-based genomic tools with Kubernetes-based workflow execution.

Arda Kırcı took a leading role in backend development and workflow execution management. He also contributed to improving workflow execution logic.

Emre Can Yoloğlu took the leading role in the development of the user interface. He coordinated the design and implementation of frontend components and ensured that the interface allowed users to easily configure workflows, upload data, and monitor execution results.

Osman Baktır assumed a leading role in the documentation of the project. He coordinated documentation tasks and ensured that the content of the project reports reflected the real system architecture, design decisions, and development progress.

7.3.4 Meeting Objectives

We had bi-weekly meetings and debugging sessions throughout the development of CASSIE. The bi-weekly meetings led to bi-weekly objectives, which were then presented to the rest of the team by the member who implemented them. The planning structure we used during both semesters created the environment for us to come up with reachable objectives. We used Jira for management purposes. Using tickets and being able to receive feedback from the team when a task was being worked on proved useful, and made it easier for us to meet the objectives.

7.4 New Knowledge Acquired and Applied

- **Kubernetes Jobs and namespace isolation:** Job lifecycle (pending→running→succeeded|failed), namespace-scoped resource quotas, shared volume mounts between init and main containers, and programmatic orphan recovery via the Python kubernetes client.
- **Nextflow DSL2:** Writing modular workflow definitions with named channels, process composition, and sub-workflow reuse; invoking the Nextflow binary from Python; parsing Nextflow structured log output for status reporting.
- **boto3 against AWS S3:** Implementing a unified S3-compatible client switchable by environment variable; generating presigned URLs; handling multipart uploads for large genomic files.
- **ReactFlow for visual DAG editing:** Custom block types for pipeline inputs, tools, and outputs with typed connection handles; DAG acyclicity and type-compatibility validation in the browser; persisting canvas state as JSONB for server-side analysis.
- **FastAPI at production scale:** Structuring a 35+ service module backend with dependency injection, lifespan-based startup/shutdown, background tasks for asynchronous job execution, and custom exception handlers.

- **The genomics tool ecosystem:** Understanding the input/output contracts, resource requirements, and runtime characteristics of all eleven tools; containerising each with its dependencies; writing consistent wrapper scripts for kubernetes_manager.
- **Critical-path estimation on a DAG:** Implementing the longest-path algorithm on the ReactFlow DAG in runtime_estimator_service.py to produce accurate pipeline duration estimates accounting for parallel branches.
- **AWS production deployment:** Configuring EC2 + Minikube + Docker Compose; provisioning an ALB with health checks; configuring S3 bucket policies and IAM roles; writing a complete operator runbook.

7.5 Hardships

Cloud architecture was something none of us had any experience with. We needed to learn it from the start. We struggled a lot with even the simplest task regarding cloud software initially, such as getting an instance to run successfully, configuring networking rules and handling cloud storage buckets. After some time we became more comfortable designing and deploying the cloud infrastructure and the scaling of it. As of its final state, our cloud architecture is capable of handling large genomic sequencing datasets and enabling users the run the tools without any local setups.

Kubernetes orchestration was another topic we struggled with as we did not have any prior experience there as well. We didn't know anything about pods, deployments, services, etc. As of our software’s final state, the orchestration layer manages the tool jobs reliably, ensuring coordination between the various components of CASSIE

We experienced many hardships in containerization and resource allocation, even though we had some experience regarding this. We struggled with how to properly containerize bioinformatics tools since we had to handle their complex dependency chains and configure resource limits and requests carefully. In early development some jobs would overconsume resources while others starved, destabilizing the cluster. Now the software is capable of handling multiple concurrent jobs from multiple users, with tools working reliably and no jobs starving.

8. Conclusion and Future Work

8.1 Conclusion

CASSIE delivers a complete implementation of the five subsystems: the Dynamic User Interface, Container Orchestrator, Workflow Manager, Cloud Manager, and Authentication Manager. Most of the test cases were executed successfully. The system supports an AWS EC2/S3 deployment.

The final system consists of Python-based backend services, a React/TypeScript frontend, containerized bioinformatics tools covering the genome assembly and annotation pipeline, and deployment configurations for cloud environment.

CASSIE enables researchers to upload genomic data, construct analysis pipelines, review estimated runtime and cost, execute workflows, and download results entirely through a web interface without requiring knowledge of underlying infrastructure technologies.

8.2 Future Work

Credential manager: Integrating AWS Secrets Manager, HashiCorp Vault, or Kubernetes Secrets would remove the reliance on plain environment variables for sensitive credentials.

Estimator refinement: Once sufficient real-world job data has accumulated, replacing the constants with values fitted from measurements using the template.

Automated backup: Adding a scheduled backup service for the Postgres volume (AWS EBS snapshots) and S3 bucket to reduce the risk of data loss.

Batch job operations and job cloning: Users currently cannot clone a job or perform bulk operations. Adding these would improve usability for power users running parameter sweeps.

9. Glossary

CASSIE: Cloud-based Assembly Streamlined Service Integration Engine

Assembly (genome assembly): The computational process of reconstructing a complete or partial genome sequence from shorter sequencing reads produced by a sequencing instrument.

Annotation (genome annotation): The process of identifying and labelling functional features on an assembled genome, including protein-coding genes, regulatory regions, and repetitive elements.

Pipeline: A user-defined graph of bioinformatics tools persisted as ReactFlow JSONB in the pipelines table

Job: One user submission of a pipeline against specific input files, recorded in the jobs table.

Job execution: One execution attempt for a job.

Tool: One containerised bioinformatics program with its own Dockerfile and wrapper script.

Bucket: An S3 object storage bucket.

SPA: Single-Page Application: React frontend architecture where all navigation is client-side via React Router without full page reloads.

JWT: JSON Web Token (RFC 7519): token format for session management. Issued at login, validated on every protected API request.

Minikube: A tool that runs a single-node Kubernetes cluster on the host.

Nextflow DSL2: The second-generation Nextflow workflow language syntax supporting modular process definitions, named channels, and sub-workflow composition.

ReactFlow: An open-source React library for interactive block-based editors and graph visualisations; used for the CASSIE pipeline builder.

QV score: Quality Value score reported by Merqury: base-level accuracy of an assembled genome sequence, analogous to the Phred quality score for sequencing reads.

10. References

[1] The Galaxy Community, "The Galaxy platform for accessible, reproducible, and collaborative data analyses: 2024 update," Nucleic Acids Research, vol. 52, pp. W83–W94, 2024. https://doi.org/10.1093/nar/gkae410 [Accessed March 6, 2026].

[2] IEEE Computer Society, IEEE Recommended Practice for Software Requirements Specifications (IEEE 830-1998), IEEE Standards Association, 1998. https://doi.org/10.1109/IEEESTD.1998.88286 [Accessed March 6, 2026].

[3] Object Management Group, Unified Modeling Language (UML) Version 2.5.1 Specification, OMG, 2017. https://www.omg.org/spec/UML/2.5.1 [Accessed March 6, 2026].

[4] Open Container Initiative, OCI Image Format Specification v1.0.0, The Linux Foundation, 2017. https://github.com/opencontainers/image-spec [Accessed March 7, 2026].

[5] Open Container Initiative, OCI Runtime Specification v1.0.0, The Linux Foundation, 2017. https://github.com/opencontainers/runtime-spec [Accessed March 7, 2026].

[6] The Linux Foundation and Cloud Native Computing Foundation, Kubernetes API Reference Documentation, 2024. https://kubernetes.io/docs/reference/kubernetes-api [Accessed March 7, 2026].

[7] P. Di Tommaso et al., "Nextflow enables reproducible computational workflows," Nature Biotechnology, vol. 35, pp. 316–319, 2017. https://doi.org/10.1038/nbt.3820 [Accessed March 7, 2026].

[8] Seqera Labs, Nextflow Documentation, 2026. https://www.nextflow.io/docs/latest/ [Accessed March 5, 2026].

[9] P. J. Cock et al., "The Sanger FASTQ file format for sequences with quality scores," Nucleic Acids Research, vol. 38, no. 6, pp. 1767–1771, 2010. https://doi.org/10.1093/nar/gkp1137 [Accessed March 6, 2026].

[10] W. R. Pearson and D. J. Lipman, "Improved tools for biological sequence comparison," PNAS, vol. 85, no. 8, pp. 2444–2448, 1988. https://doi.org/10.1073/pnas.85.8.2444 [Accessed March 8, 2026].

[11] The GFA Specification Team, "Graphical Fragment Assembly (GFA) Format Specification v1.0," 2016. https://github.com/GFA-spec/GFA-spec [Accessed March 8, 2026].

[12] L. Stein, "GFF3 Specification: Generic Feature Format Version 3," Sequence Ontology Project, 2013. https://github.com/The-Sequence-Ontology/Specifications [Accessed March 8, 2026].

[13] The GENCODE Consortium, "GTF Format Specification," EMBL-EBI/GENCODE, 2024. https://www.gencodegenes.org/pages/data_format.html [Accessed March 8, 2026].

[14] H. Li et al., "The Sequence Alignment/Map format and SAMtools," Bioinformatics, vol. 25, no. 16, pp. 2078–2079, 2009. https://doi.org/10.1093/bioinformatics/btp352 [Accessed March 8, 2026].

[15] J. Bonfield et al., "CRAM 3.1: advances in reference-based compression of high throughput sequencing data," Bioinformatics, vol. 38, no. 6, pp. 1497–1503, 2022. https://doi.org/10.1093/bioinformatics/btac010 [Accessed March 9, 2026].

[16] J. Kent et al., “The Human Genome Browser at UCSC,” Genome Research, vol. 12, no. 6, pp. 996–1006, 2002. https://doi.org/10.1101/gr.229102 [Accessed March 9, 2026].