

PROJECT REPORT ON
GENOVATE AI

Carried Out at



CENTRE FOR DEVELOPMENT OF ADVANCED COMPUTING
ELECTRONIC CITY, BANGALORE.

UNDER THE SUPERVISION OF

KARUNA. P
C-DAC Bangalore

Submitted By

Kalyani Patil(202409332200219)
Shivaranjani A (202409332200391)
Nidhi Sahu(202409332200135)
K L Kalpana(202409332200223)
Rohit Rokade(2024332200187)

ADVANCED CERTIFICATE COURSE IN HPC,
C-DAC, BANGALORE

Candidate's Declaration

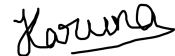
We hereby certify that the work being presented in the report entitled **GenovateAI**, in the partial fulfillment of the requirements for the award of **Kalyani Patil, Shivaranjani A, Nidhi Sahu, K L Kalpana and Rohith Rokade** and submitted in the department of Diploma in Advance Computing of the **C-DAC Bangalore**, is an authentic record of our work carried out during the period 03th September 2024– 28th February 2025 under the supervision of **Karuna** , C-DAC Bangalore.

The matter presented in the report has not been submitted by me for the award of any degree of this or any other Institute/University.

Submitted By :

Kalyani Patil
Shivaranjani A
Nidhi Sahu
K L Kalpana
Rohith Rokade

Submitted To :



Karuna P

ACKNOWLEDGMENT

I take this opportunity to express my gratitude to all those people who have been directly and indirectly with me during the competition of this project

I pay thank to **Karuna.P** who has given guidance and a light to me during this major project. His versatile knowledge about “GenovateAI” has eased me in the critical times during the span of this Final Project.

I acknowledge here out debt to those who contributed significantly to one or more steps. I take full responsibility for any remaining sins of omission and commission.

Submitted By

Kalyani Patil(202409332200219)
Shivaranjani A (202409332200391)
Nidhi Sahu(202409332200135)
K L Kalpana(202409332200223)
Rohit Rokade(2024332200187)

TABLE OF CONTENTS

SL.NO	Title	Page number
1.	Abstraction	i
2.	Introduction	1
3.	Literature Survey	2
4.	Software Requirement Specification	4
5.	System Design	6
6.	Architecture	7
7.	Implementation	13
8.	Conclusion	31
9.	Reference	32

1. ABSTRACTION

GenovateAI is an advanced genomic analysis platform that integrates Artificial Intelligence (AI) and High-Performance Computing (HPC) to enhance genomic research and clinical applications. The platform automates complex genomic data processing, enabling efficient gene expression prediction, variant impact assessment, and disease risk analysis. By leveraging machine learning models, real-time data synchronization, and scalable infrastructure, GenovateAI ensures high accuracy, efficiency, and reliability in genomic analysis.

Designed to support diverse genomic data sources, the platform provides automated workflows, actionable insights, and seamless integration with external tools while maintaining robust security and compliance with global privacy standards. GenovateAI empowers researchers and healthcare professionals to accelerate discoveries in precision medicine, improving clinical decision-making and patient outcomes.

2. INTRODUCTION

Genomics is an interdisciplinary field of biology that studies the structure, function, evolution, mapping, and editing of genomes. A genome is the complete set of DNA instructions in an organism, containing all the genetic information necessary for development and functioning.

To analyse vast amounts of genomic data, researchers rely on High-Performance Computing (HPC) and Artificial Intelligence (AI) encompasses both Machine Learning (ML) and Deep Learning (DL). HPC enables rapid processing of large datasets, while ML & DL helps automate complex tasks such as annotating genomic sequences and interpreting large genomic datasets.

Recent advancements in genomics have led to the development of new approaches combining HPC and Artificial intelligence (AI) to enhance genomic related predictions. These methods aim to improve speed, accuracy, and scalability of genomic related prediction, potentially enabling real-time clinical applications for Researchers and Lab-Technicians.

One such approach involves developing web applications that provide access to genomic data. The application we have created is **GenovateAI**. GenovateAI allows researchers and lab technician to input raw genomic data and user-defined parameters, receiving detailed reports on DNA-related aspects such as Gene Expression Predictions, Variant Impact Assessments, Lung Cancer Predictions, and Functional Annotations.

The integration of HPC and AI in genomics has revolutionized the field, enabled faster analysis and reduced costs. For instance, GPU-based approaches can accelerate analysis from 24 hours plus on CPUs to less than 25 minutes on specialized systems, significantly improving efficiency and reducing computational costs.

Furthermore, advancements in genomics are crucial for precision medicine, enabling personalized treatments based on an individual's genetic profile. Machine learning techniques and Deep Learning techniques are being employed to handle the massive amounts of clinical data collected and analysed in this field, making genomic research more accessible.

In conclusion, the combination of HPC, ML and DL is transforming genomics by enabling faster, more accurate analysis of large genomic datasets, paving the way for improved precision medicine and personalized treatments.

3. LITERATURE SURVEY

The development of Genovate AI is grounded in the latest advancements in genomics, AI, and high-performance computing. Below is a summary of the key areas of research and technologies that have influenced the design and functionality of the platform:

Author	Title	Published In	Year	Summary
Avsec, Ž., Agarwal,	Effective gene expression prediction from sequence by integrating long-range interactions	Nature Methods	2021	Introduces a deep learning model that integrates long-range genomic interactions for accurate gene expression prediction from DNA sequences.
Schatz, M.C.	Computational genomics in the cloud: Methods and applications	Genome Biology	2017	Discusses cloud-based HPC approaches for large-scale genomic analysis, focusing on scalability and cost-effectiveness.
Libbrecht, M.W., & Noble, W.S.	Machine learning applications in genetics and genomics	Nature Reviews Genetics	2015	Explores how ML techniques are applied in genomics, including variant prediction, gene expression modeling, and disease association studies.
Langmead, B., & Nellore, A.	Cloud computing for genomic data analysis and collaboration	Nature Reviews Genetics	2018	Highlights the role of cloud-based HPC solutions in genomic data sharing, analysis, and collaboration.
Hassanzadeh, H.R., & Luo, Y.	Leveraging artificial intelligence for genomic variant interpretation	Briefings in Bioinformatics	2021	Discusses AI-driven approaches for annotating and interpreting genomic variants, enhancing accuracy in clinical genomics.
He, K., Zhang, X., Ren, S., & Sun, J.	Deep learning in medical image analysis and genomics	Proceedings of the IEEE	2016	Examines deep learning models and their applications in genomic data analysis, including disease risk prediction and functional genomics.

Karczewski, K.J., & Snyder, M.P.	Integrative omics for precision medicine	Nature Reviews Genetics	2018	Explores how multi-omics data combined with AI and HPC can drive advancements in precision medicine and personalized treatments.
Lee, S., & Schatz, M.C.	Genomic data analysis using GPU-accelerated computing	Bioinformatics	2020	Details GPU-based acceleration techniques for faster genomic analysis, reducing computation time from days to minutes.
Poplin, R., & Chang, R.	A universal SNP and small-indel variant caller using deep neural networks	Nature Biotechnology	2018	Introduces DeepVariant, a deep learning model for SNP and small indel variant calling with improved accuracy over traditional methods.
Koumakis, L.	Deep learning in precision medicine and biomedical informatics	Computational and Structural Biotechnology Journal	2020	Discusses the use of deep learning models for processing vast biomedical datasets, including genomic and transcriptomic data.

4. SOFTWARE REQUIREMENT SPECIFICATION

Technical Stack:

Frontend	Reactjs + vite	for responsive user interfaces.
Backend	Node.js and Django	for scalable network applications and data handling.
Database	MongoDB Atlas	for secure and flexible NoSQL data storage.
AI/ML Libraries	TensorFlow and PyTorch	for developing and deploying machine learning models.
HPC Integration	GPU	resources for accelerated data processing.

Non-Functional Requirements:

Performance:

Handle large genomic datasets efficiently.
Ensure real-time updates within 2-5 seconds.

Scalability:

Support growing user numbers and increasing data volumes.

Reliability:

Ensure stable operation with minimal downtime.

Maintainability:

Facilitate easy updates and fixes for system improvements.

Interoperability:

Integrate seamlessly with external genomic tools and databases.

Real-Time Data Synchronization:

Automatically sync genomic data and analysis results in real-time.

User Interface:

Provide an intuitive, responsive, and customizable dashboard for users.

Security:

Implement two-factor authentication (2FA), data encryption, and compliance with GDPR and HIPAA.

Functional Requirements:**Gene Expression Analysis:**

Predict gene expression levels using AI models.

Study regulatory mechanisms and cellular behaviors.

Variant Impact Assessment:

Evaluate genetic variations to determine their functional consequences.

Provide insights into disease mechanisms and therapeutic targets.

Lung Cancer Prediction:

Use predictive analytics to assess the likelihood of lung cancer based on genomic patterns.

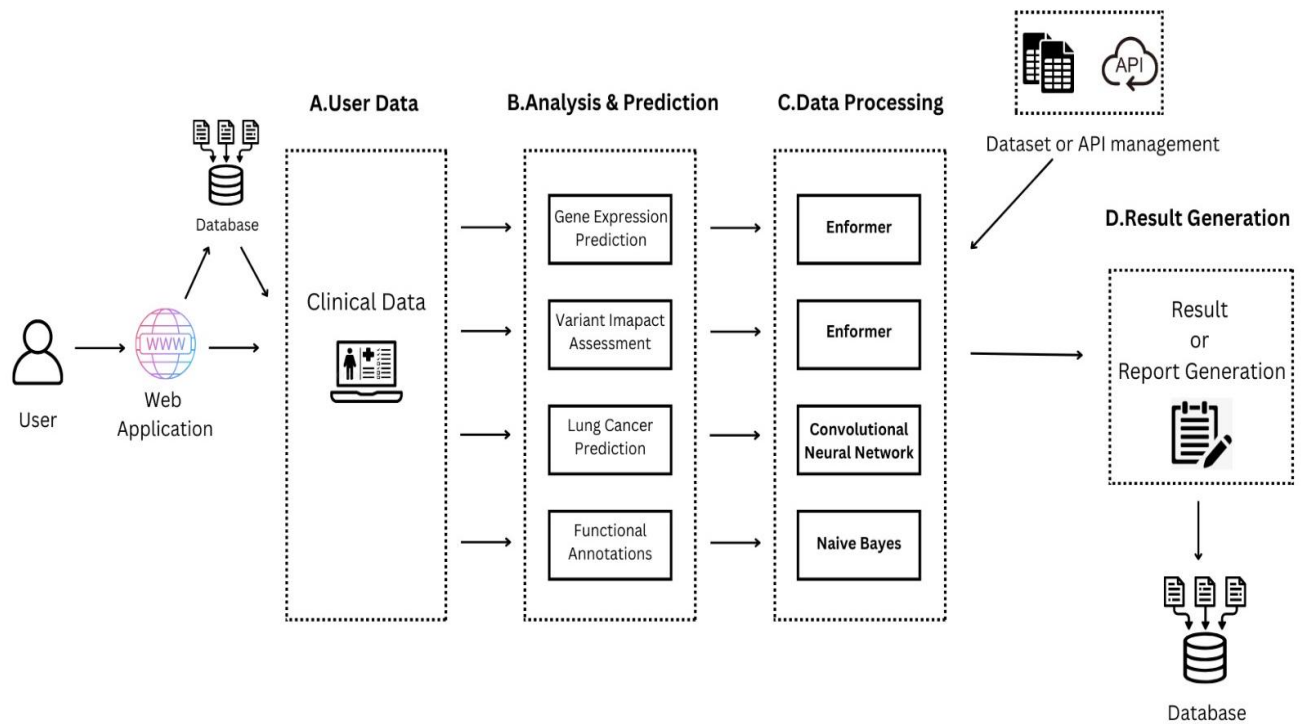
Enhance early detection and intervention strategies.

Functional Annotations:

Assign biological significance to genetic variants.

Combine variant databases and predictive modeling for informed clinical decisions.

5. SYSTEM DESIGN



1. User Data: The user (lab technician or researchers) gives the input by uploading raw genomic data file (FASTQ files) in the web applications.

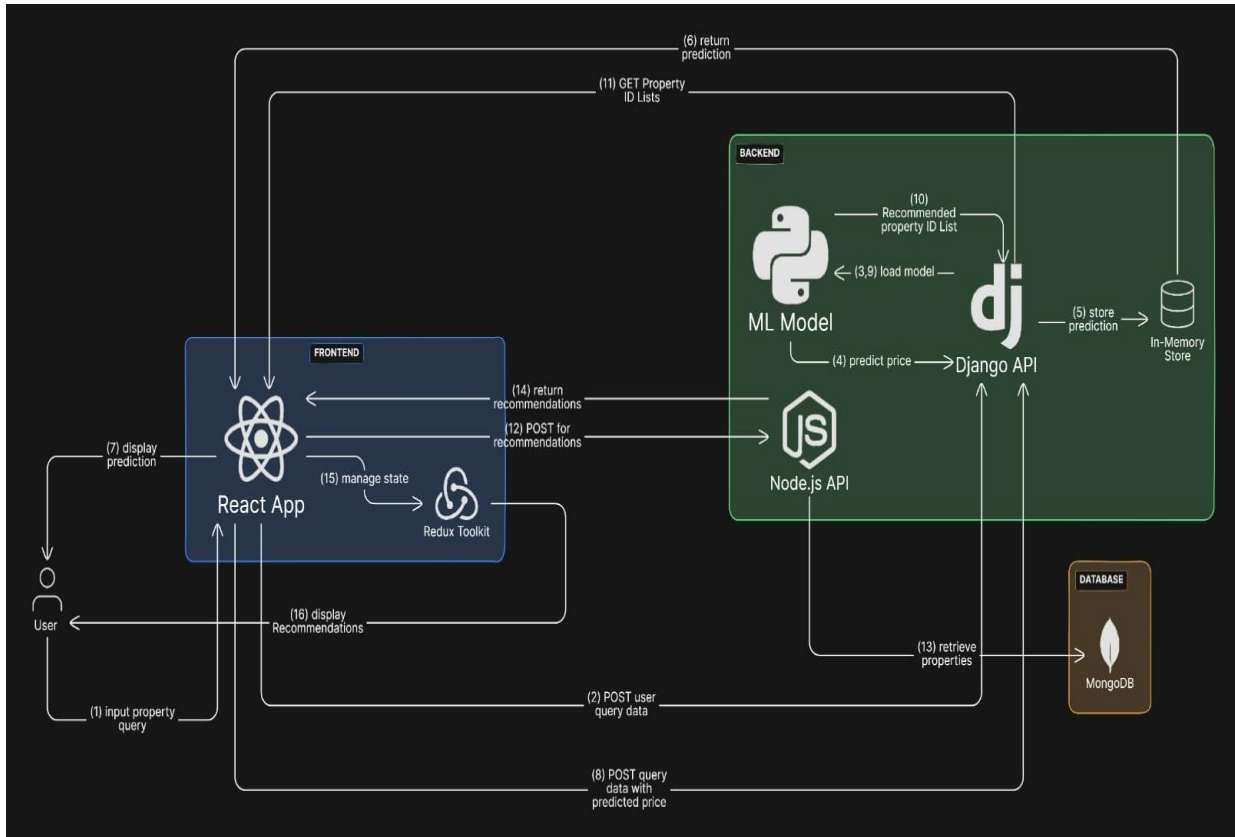
2. Analysis& Prediction: The user selects from the following analysis options:

- Gene Expression Prediction
- Variant Impact Assessment
- Lung Cancer Prediction
- Functional Annotations

3. Data Processing: Based on the selected analysis, the appropriate algorithms will be executed. The results will be compared against a predefined dataset in the backend.

4. Results Generation: The results will be on the webpage the users can also download reports in the desired formats.

6. Architecture Diagram



A **Microservice Architecture** is a software design approach where an application is divided into small, independent services that communicate through APIs. Each service focuses on a specific function, allowing for greater scalability, flexibility, and ease of maintenance. In **GenovateAI**, the **microservice architecture** enables seamless integration of different components:

- **MERN stack (React.js with Vite, Express.js, Node.js, MongoDB Atlas)** handles user authentication and data management.
- **Django + HPC cluster** processes genomic data using AI models.
- The **React.js frontend** displays real-time AI-driven insights.

This architecture enhances performance, supports independent scaling of services, and ensures a smooth user experience.

Layered Architecture Components

1. Presentation Layer (Front-end)

1. **Users:** Lab technician and Researchers.
2. **Technologies:**
 1. Web: React.js.

3. Features:

1. User-friendly UI
2. Course browsing & enrollment
3. Interactive user interface
4. Getting output results in the form of reports
5. Real-time analysis

2. Application Layer (Back-end & Business Logic)

1. **Technologies:** Node.js, Django,

- **Key Components:**

2.1.1 Component 1: Data Input

- **Purpose:** Load and parse input files (VCF and FASTA).

- **Key Responsibilities:**

1. Read the ClinVar VCF file (contains variant details).
2. Load the genome FASTA file (provides the reference genome).

Steps:

1. Parse VCF file for chromosome, position, reference allele, and alternate allele.
2. Load genome FASTA file to extract sequences around variants.

i. Component2: Data Preprocessing

- **Purpose:** Prepare input data for the model.

- **Key Responsibilities:**

1. Extract sequences around the variants.
2. One-hot encode the extracted sequences for model compatibility.

Steps:

3. Extract DNA sequences around each variant using the genome FASTA file.
4. One-hot encode the DNA sequences (A, C, G, T as binary matrices).
5. Generate a dataset of encoded sequences for input to the model.

ii. Component3: Model

- **Purpose:** Analyze the prepared sequences and make regulatory activity predictions.

- **Key Responsibilities:**

1. Load the Enformer model from TensorFlow Hub.
2. Input one-hot encoded sequences into the model.
3. Predict regulatory activity across various cell types and tissues.

Steps:

1. Load the pre-trained Enformer model.
2. Pass one-hot encoded sequences as input.
3. Predict regulatory activity changes for each sequence.
4. Send predictions to the postprocessing step.

iii. Component 4: Data Post Preprocessing

- **Purpose:** Process model predictions and categorize variants.

- **Key Responsibilities:**

1. Normalize the predictions using PCA and scaling.
2. Categorize variants into "Benign," "Pathogenic," etc., based on scores.

Steps:

1. Receive predictions from the model.
2. Normalize predictions using PCA/scaling.
3. Define bins for categorization (e.g., "Benign," "Pathogenic").
4. Categorize each variant based on its scores.
5. Save the results for reporting.

iv. Component 5: Output

- **Purpose:** Store and present results.

- **Key Responsibilities:**

1. Save categorized variants in structured formats (e.g., CSV files).
2. Provide interfaces to view or query results.

- **Steps:**

3. Store results (e.g., variant scores and categories).
4. Provide outputs for visualization or further analysis.

6. Data Layer (Database & Storage)

1. **Technologies:** MongoDB Atlas

• **Components:**

1. **User Management** (Profiles, login credentials dashboard)
2. **Course Content Storage**

(Reports, Image and Fasta file)

3. **Process Tracking** (User engagement data(History))

4. Additional Components

1. API Gateway & Microservices

1. **Purpose:** Manages multiple service requests efficiently.
2. **Technologies:** Kong, Nginx

- **Use Cases:**

1. Route user authentication requests
2. Fetch data dynamically

- Handle input file

2. Security Measures

1. **Data Encryption:** SSL/TLS for secure connections
2. **Authentication:** OAuth2.0, JWT tokens
3. **Role-Based Access Control (RBAC)**

The **system design** of an e-learning platform outlines its architecture, components, and interactions to ensure scalability, efficiency, and user-friendliness. Below is a **detailed system design**, covering **high-level design (HLD)** and **low-level design (LLD)**.

1. High-Level Design (HLD)

1.1 System Architecture

The e-learning platform follows a **three-tier architecture**:

1. **Presentation Layer (Front-End)**
 - User Interface (Website)
 - Technologies: React.js
2. **Application Layer (Back-End)**
 - Handles business logic, authentication, and API requests
 - Technologies: Node.js, Django, HPC
3. **Data Layer (Database & Storage)**

- Stores user data, reports and history
- Technologies: MongoDB

1.2 System Components

Component	Description	Technology
User Authentication	Handles login, registration, and rolebased access	Mongoddb atlas
Data input	Load and parse input files (VCF and FASTA).	Node.js, Django
Data Preprocessing	Prepare input data for the model.	Django, HPC
Model	Analyze the prepared sequences and make regulatory activity predictions.	Python, Machine Learning
Data Post Preprocessing	Process model predictions and categorize variants.	TensorFlow, CNN, Enformer model , Naïve Bayes algorithm
Output	Store results and show output	Reactjs

1.3 Use Case Diagram

The **Use Case Diagram** defines the interactions between users and the system.

1. **User Authentication** – Login for lab technicians and researchers.
2. **Upload Medical Image** – Lab technicians upload lung X-ray/CT images (PNG/JPEG), Fasta file ,csv file.
3. **Run Lung Cancer Prediction** – System processes the uploaded image through the lung cancer prediction model on the HPC cluster.
4. **View Prediction Result** – Users receive output.
5. **Track Process Status** – Real-time progress tracking while the model runs.
6. **View Input History** – Users can view previously uploaded images and results.
7. **Generate Report** – Researchers can generate a report based on model outputs.

2. Low-Level Design (LLD)

2.1 Database Design (ER Diagram)

The **Entity-Relationship (ER) Diagram** defines the database schema.

Entities:

1. **Users** (Lab_ID or Res_ID, Name, Email, Designation)
2. **Reports** (Report id, prediction id)

Diagram Relationships:

- **One-to-Many**: One lab technician can upload multiple patient data entries.
 - **One-to-One**: Each patient record has one corresponding result.
 - **One-to-Many**: Each prediction generates multiple reports.
 - **One-to-Many**: Each patient data entry has a tracking record for process status
-

2.2 API Design

The system follows a **RESTful API design** for communication between the front-end and back-end.

Example API Endpoints:

Functionality	Endpoint	Method
User Login	/api/login	POST, GET
Register	/api/register	POST
History	/api/history/{userid}	GET
Dashboard	/api/dashboard	GET, PUT, POST

2.3 Workflow of Key Features

1. User Registration & Login

1. User enters credentials.
2. Backend verifies user via **authentication**.
3. Role-based access is assigned (**Lab-technician, Researchers**).

2. Analysing Input

1. Users can give input by selecting one of component among 4 components which is Gene expression, Variant Risk, Functional Annotation and lung cancer prediction.
2. According to the component selected User needs to enter file type.

3. Data Preprocessing

1. Extract DNA sequences around each variant using the genome FASTA file.
 2. One-hot encode the DNA sequences (A, C, G, T as binary matrices).
 3. Generate a dataset of encoded sequences for input to the model.
-

3. Security Considerations

1. **Data Encryption:** SSL/TLS for secure communication.
 2. **Role-Based Access Control (RBAC):** Ensures only authorized users access data.
 3. **Token-Based Authentication:** OAuth2.0, JWT for session management.
 4. **DDoS Protection:** Cloudflare for preventing cyberattacks.
-

7. IMPLEMENTATION

Here's a complete **implementation guide for a GenovateAI using MERN stack and DJANGO using HPC.**

1. Technology Stack

Component	Technology
Frontend (UI)	Reactjs + vite,
Backend	Nodejs,Django
Database	MongoDB Atlas
State Management	React Context API
Deployment	Reactjs (Frontend)

2. Setting Up the React.js Frontend

Step 1.1: Create a React App

`npm create vite@latest my-react-app`

Step 1.2 : Select a framework: select the **React** framework here using the downward arrow key.

Vanilla

Vue

React

Preact

Lit

Svelte

Solid

Qwik

Others

Step 1.3: Select Variant: choose any variant of your choice using the downward arrow key,i.e: choose JavaScript

TypeScript

TypeScript + SWC

JavaScript

JavaScript + SWC

Step 1.4:

`cd my-react-app`

Step 2: Install Required Dependencies

```

npm install
"dependencies": {
  "axios": "^1.7.9",
  "bootstrap": "^5.3.3",
  "chart.js": "^4.4.7",
  "gasp": "^0.0.2",
  "gsap": "^3.12.7",
  "react": "^18.3.1",
  "react-bootstrap": "^2.10.9",
  "react-chartjs-2": "^5.3.0",
  "react-dom": "^18.3.1",
  "react-icons": "^5.4.0",
  "react-router-dom": "^7.1.4"
}

```

3. Implementing Authentication (MongoDB Atlas)

Step 1: Configure MongoDB Atlas

1. Register to the MongoDB atlas to store that data in a MongoDB Atlas
2. Create a Cluster and click "Create Deployment" button
3. Create a database User -A username and password can be autogenerated, but you can change those and provide a good username and a password, avoid any special characters (!@#\$%^_&) in your password otherwise you can not connect it with the database. Once you are done, click on the 'Create Database User' button.
4. Once you have done all the steps, select the connection method as the first option ("Drivers"). Then copy the connection string
5. Create a .env file. After that create a .env file in your project root directory to hide the sensitive details and put your connection string in there.

```

MONGODB_URI=mongodb+srv://<username>:<password>@cluster0.mong
odb.net/<dbname>?retryWrites=true&w=majority

```

Step 2: Initialize MongoDB Atlas in React

Config.db

```

import mongoose from "mongoose";
export const connectDB = async () => {
  try {
    const conn = await mongoose.connect(process.env.MONGO_URI);
    console.log(`MongoDB Connection: ${conn.connection.host}`);
  } catch (error) {
    console.error(`Error: ${error.message}`);
    process.exit(1); // Exit with failure
  }
};

```

Step 3. Nodejs + MongoDB connection(index.js)

```

import express from "express";
import bodyParser from "body-parser";
import cors from "cors";
import dotenv from "dotenv";

```

```

import userRoutes from './Routes/UserRoute.js';
import { connectDB } from './config/db.js';
dotenv.config();
console.log("MONGO_URI:", process.env.MONGO_URI);
const app = express();
app.use(cors());
app.use(bodyParser.json());
app.use("/api",userRoutes);
// Basic Route
app.get("/", (req, res) => {
  res.send("Backend is running");
});
// MongoDB connection
app.listen(process.env.PORT || 5000, () => {
  connectDB();
  console.log(`Server started at http://localhost:${process.env.PORT || 5000}`);
});

```

4. Implementing Course Management

Step 1: Create Backend API (Node.js + Express)

```

mkdir elearning-backend && cd elearning-backend
npm init -y
npm install express mongoose cors dotenv body-parser

```

create index.js

```

import express from "express";
import bodyParser from "body-parser";
import cors from "cors";
import dotenv from "dotenv";
import userRoutes from './Routes/UserRoute.js';
import { connectDB } from './config/db.js';
dotenv.config();
console.log("MONGO_URI:", process.env.MONGO_URI);
const app = express();
app.use(cors());
app.use(bodyParser.json());
app.use("/api",userRoutes);
// Basic Route
app.get("/", (req, res) => {
  res.send("Backend is running");
});
// MongoDB connection
app.listen(process.env.PORT || 5000, () => {

```

```

connectDB();
console.log(`Server started at http://localhost:${process.env.PORT || 5000}`);
});
// Graceful shutdown
process.on("SIGINT", async () => {
  console.log("Shutting down server...");
  await mongoose.connection.close();
  console.log("MongoDB connection closed.");
  process.exit(0);
});
process.on("SIGTERM", async () => {
  console.log("Shutting down server...");
  await mongoose.connection.close();
  console.log("MongoDB connection closed.");
  process.exit(0);
});
Step 2. Login component
import React, { useState } from "react";
import { Link, useNavigate, useLocation } from "react-router-dom";
import "./Login.css";
import axios from "axios";
import NavigationBar from "../Home/Navbar"; // Import the Navbar component

const Login = () => {
  const [password, setPassword] = useState("");
  const [resId, setResId] = useState("");
  const [labId, setLabId] = useState("");
  const [category, setCategory] = useState("");
  const [errorField, setError] = useState("");
  const [showPassword, setShowPassword] = useState(false);

  const togglePasswordVisibility = () => {
    setShowPassword(!showPassword);
  };

  const navigate = useNavigate();
  const location = useLocation();

  const handleCategoryChange = (e) => {
    const value = e.target.value;
    setCategory(value);
    setError("");
  };

```

```

const handleResIdChange = (e) => {
  const value = e.target.value;
  setResId(value);
  setError("");
};

const handleLabIdChange = (e) => {
  const value = e.target.value;
  setLabId(value);
  setError("");
};

const handlePasswordChange = (e) => {
  const value = e.target.value;
  setPassword(value);
  setError("");
};

const handleLogin = (e) => {
  e.preventDefault();
  if (!category) {
    setError("Please select a category");
    return;
  }
  if (category === "Researcher" && (!resId || !password)) {
    setError("Res ID and password cannot be empty");
    return;
  }
  if (category === "Lab Technician" && (!labId || !password)) {
    setError("Lab ID and password cannot be empty");
    return;
  }
  Login(category, resId, labId, password);
};

const Login = (category, resId, labId, password) => {
  setError("");
  const loginData = { category, resId, labId, password };
  console.log("Login Payload:", loginData); // Debugging
  axios
    .post("http://localhost:5000/api/login", loginData)
    .then((response) => {
      if (response.data.message === "Login successful") {
        console.log("Login successful");
      }
    })

```

```

const queryString = location.search;
let strReturnUrl = new URLSearchParams(queryString).get("returnUrl");
if (strReturnUrl === null) {
  strReturnUrl = "/dashboard";
}
let token = "ASJDFJF87ADF8745LK4598SAD7FAJSDF45JSDFLKAS";
sessionStorage.setItem("user-token", token);
sessionStorage.setItem("category", category);
sessionStorage.setItem("userId", category === "Researcher" ? resId : labId); // Store userId
navigate(strReturnUrl);
setError(response.data.message);
} else {
  setError("Invalid credentials");
  console.error("Login failed:", response.data.error);
}
})
.catch((error) => {
  setError(error?.response?.data?.message);
  console.error("Error during login:", error);
});
};

return (
  <>
  <NavigationBar />
  <div className="login-outer-container login-background">
    <div className="login-container">
      <div className="login-overlay-container">
        <div className="login-overlay">
          <div className="login-overlay-panel login-overlay-left">
            <h1 className="login-heading">Welcome to GenovateAI</h1>
            <p className="login-section">
              GenovateAI is a cutting-edge platform for researchers and lab technicians to collaborate
and innovate in the field of genomics.
            </p>
          </div>
        </div>
      </div>
    </div>
    <div className="login-form-container login-log-in-container">
      <form className="login-input-form">
        <h1 className="login-headings">Login</h1>
        <div className="login-formgroup">
          <select
            className="login-input"

```

```

        value={category}
        onChange={handleCategoryChange}
    >
        <option value="">Select Category</option>
        <option value="Researcher">Researcher</option>
        <option value="Lab Technician">Lab Technician</option>
    </select>
</div>
{category === "Researcher" && (
    <div className="login-formgroup">
        <input
            className="login-input"
            type="text"
            placeholder="Enter your Res ID"
            value={resId}
            onChange={handleResIdChange}
        />
    </div>
)}
{category === "Lab Technician" && (
    <div className="login-formgroup">
        <input
            className="login-input"
            type="text"
            placeholder="Enter your Lab ID"
            value={labId}
            onChange={handleLabIdChange}
        />
    </div>
)}
<div className="login-formgroup">
    <input
        className="login-input"
        type={showPassword ? "text" : "password"}
        placeholder="Enter your password"
        value={password}
        onChange={handlePasswordChange}
    />
    <p className="login-showPassword1" onClick={togglePasswordVisibility}>
        {showPassword ? (
            <i className="bi bi-eye-slash"></i>
        ) : (
            <i className="bi bi-eye"></i>
        )}
    </p>

```



```

    </p>
  </div>
  <button type="button" className="login-submit" onClick={handleLogin}>
    Login
  </button>
  <Link
    to="/forgotpassword"
    className="mb-3 mt-3 link-success link-offset-2 link-underline-opacity-25 link-underline-
opacity-100-hover text-decoration-none"
    onClick={(e) => {
      e.preventDefault();
      navigate("/forgotpassword");
    }}
  >
    Forgot Password
  </Link>
  <p className="d-inline">
    Not a Member?
    <Link
      to="/registration"
      className="mb-3 mt-3 link-success link-offset-2 link-underline-opacity-25 link-
underline-opacity-100-hover text-decoration-none"
    >
      SignUp
    </Link>
  </p>
  {errorField && (
    <div className="login-error-message text-danger text-center">
      {errorField}
    </div>
  )}
</form>
</div>
</div>
</div>
</>
);
};

```

```
export default Login;
```

5. Implementing Analysis

```

import React, { useState, useEffect } from "react";
import "bootstrap/dist/css/bootstrap.min.css";
import "./Dashboard.css";

```

```

import { useNavigate } from "react-router-dom";
import { FaBars, FaHome, FaChartLine, FaHistory, FaSignOutAlt } from "react-icons/fa";
import axios from "axios";

const Dashboard = () => {

  const [user, setUser] = useState({
    name: "",
    phone: "",
    email: "",
  });
  const [editMode, setEditMode] = useState(false);
  const [showAnalysis, setShowAnalysis] = useState(false);
  const [showHistory, setShowHistory] = useState(false); // New state for History panel
  const [history, setHistory] = useState([]);
  const [analysisType, setAnalysisType] = useState("Gene Expression");
  const [file, setFile] = useState(null);
  const [imageUrl, setImageUrl] = useState("");
  const [isCollapsed, setIsCollapsed] = useState(false);

  const category = sessionStorage.getItem("category");
  const userId = sessionStorage.getItem("userId");
  const navigate = useNavigate();

  // Logout Functionality
  const handleLogout = async () => {
    try {
      // Update logout time in the backend
      await axios.put(`http://localhost:5000/api/history/logout/${userId}`);
    } catch (error) {
      console.error("Error updating logout time:", error);
    }
  }

  // Clear session storage
  sessionStorage.removeItem("user-token");
  sessionStorage.removeItem("category");
  sessionStorage.removeItem("userId");

  // Redirect to login page
  navigate("/login");
};

// PROFILE VIEW AND UPDATE
useEffect(() => {

```

```

const fetchUserDetails = async () => {
  try {
    const response = await axios.get(`http://localhost:5000/api/user/${userId}`);
    setUser(response.data);
  } catch (error) {
    console.error("Error fetching user details:", error);
  }
};

fetchUserDetails();
}, [userId]);

const handleProcess = async () => {
  if (!file) {
    alert("Please upload a FASTA file");
    return;
  }

  try {
    // Create FormData to send the file as multipart/form-data
    const formData = new FormData();
    formData.append("file", file);

    const response = await fetch("http://127.0.0.1:8000/mlapp/process/", {
      method: "POST",
      body: formData, // Send the file inside formData
    });

    if (!response.ok) {
      throw new Error("Error processing ML model");
    }

    const data = await response.json();
    setImageUrl(data.image_url);

    // Store process time and analysis preference in the history
    const processTime = new Date().toISOString();
    await axios.post("http://localhost:5000/api/history", {
      userId,
      name: user.name,
      phone: user.phone,
      analysisPreference,
      processTime, // Send process time to the backend
    });
  }
};

```

```

// Fetch history again to update the history panel
const historyResponse = await axios.get(`http://localhost:5000/api/history/${userId}`);
setHistory(historyResponse.data);

} catch (error) {
  console.error("Error processing ML model:", error);
}
};

const handleChange = (e) => setUser({ ...user, [e.target.name]: e.target.value });

const handleSubmit = async (e) => {
  e.preventDefault();
  try {
    await axios.put(`http://localhost:5000/api/user/${userId}`, user);
    alert("User details updated!");
    setEditMode(false);
  } catch (error) {
    console.error("Error updating user details:", error);
  }
};

const handleDelete = async () => {
  try {
    await axios.delete(`http://localhost:5000/api/user/${userId}`);
    alert("User details deleted!");
    setUser({ name: "", phone: "", email: "" });
  } catch (error) {
    console.error("Error deleting user details:", error);
  }
};

// HISTORY VIEW
useEffect(() => {
  const fetchHistory = async () => {
    try {
      const response = await axios.get(`http://localhost:5000/api/history/${userId}`);
      setHistory(response.data);
    } catch (error) {
      console.error("Error fetching history:", error);
    }
  };
  fetchHistory();
}, [userId]);

```

```

const handleFileChange = (e) => {
  const selectedFile = e.target.files[0];
  if (!selectedFile) return;

  const fileTypeMap = {
    "Gene Expression": [".fasta", ".fa"],
    "Lung Cancer Prediction": [".jpeg", ".png"],
    "Functional Annotation": [".csv"],
    "Variant Risk": [".fasta", ".cvf"],
  };

  const validExtensions = fileTypeMap[analysisType];
  const fileExtension = selectedFile.name.split(".").pop().toLowerCase();

  if (!validExtensions.includes(`${fileExtension}`)) {
    alert(`Please upload a valid file type: ${validExtensions.join(", ")}`);
    setFile(null);
    return;
  }
  setFile(selectedFile);
};

return (
  <div className={`dashboard-container ${isCollapsed ? "collapsed" : ""}`}>
    /* Sidebar */
    <div className={`sidebar ${isCollapsed ? "collapsed" : ""}`}>
      <button className="collapse-btn" onClick={() => setIsCollapsed(!isCollapsed)}>
        <FaBars />
      </button>
      <h1 className="logo">
        {!isCollapsed && (
          <>
            
            <span className="title">GenovateAI</span>
          </>
        )}
        {isCollapsed ? "G" : ""}
      </h1>

      <ul>
        <li>
          <button className="sidebar-btn" onClick={() => { setShowAnalysis(false);
setShowHistory(false); }}>

```

```

        <FaHome className="icon" /> {!isCollapsed && "Dashboard"}
    </button>
</li>
<li>
    <button className="sidebar-btn" onClick={() => { setShowAnalysis(true);
setShowHistory(false); }}>
        <FaChartLine className="icon" /> {!isCollapsed && "Analysis"}
    </button>
</li>
<li>
    <button className="sidebar-btn" onClick={() => { setShowAnalysis(false);
setShowHistory(true); }}>
        <FaHistory className="icon" /> {!isCollapsed && "History"}
    </button>
</li>
</ul>

<button className="logout-btn" onClick={handleLogout}>
    <FaSignOutAlt className="icon" /> {!isCollapsed && "Logout"}
</button>
</div>

{/* Main Content */}
<div className="main-content">
    {!showAnalysis && !showHistory ? (
        <div className="user-dashboard">
            <div className="labid">{category === "Researcher" ? "ResID" : "LabID"} : {userId}</div>
            <br />
            {editMode ? (
                <form onSubmit={handleSubmit}>
                    <label>Name</label>
                    <input
                        type="text"
                        name="name"
                        value={user.name}
                        onChange={handleChange}
                        required
                    />
                    <label>Phone</label>
                    <input
                        type="text"
                        name="phone"
                        value={user.phoneNumber}
                        onChange={handleChange}

```

```

        disabled // Disable the phone number field
    />
    <label>Email</label>
    <input
        type="email"
        name="email"
        value={user.email}
        onChange={handleChange}
        required
    />
    <button type="submit" className="process-btn">
        Save
    </button>
</form>
) : (
    <>
        <p><strong>Name:</strong> {user.name}</p>
        <p><strong>Phone:</strong> {user.phoneNumber}</p>
        <p><strong>Email:</strong> {user.email}</p>
        <button onClick={() => setEditMode(true)} className="process-btn">
            Edit
        </button>
        <button onClick={handleDelete} className="delete-btn">
            Delete
        </button>
    </>
    </div>
) : showAnalysis ? (
    <div className="analysis-panel">
        <div className="labid">{category === "Researcher" ? "ResID" : "LabID"} : {userId}</div>
        <br />
        <label>Analysis Preference:</label>
        <select value={analysisType} onChange={(e) => setAnalysisType(e.target.value)}>
            <option>Gene Expression</option>
            <option>Lung Cancer Prediction</option>
            <option>Functional Annotation</option>
            <option>Variant Risk</option>
        </select>

        <p className="file-note">**Required file type: {analysisType === "Gene Expression" ?
"FASTA,FA" : analysisType === "Lung Cancer Prediction" ? "JPEG, PNG" : analysisType ===
"Functional Annotation" ? "CSV" : "FASTA, CVF"}</p>
        <br />

```

```

<label>Upload File:</label>
<input type="file" onChange={handleFileChange} />
<button className="process-btn" onClick={handleProcess}>Process</button>
{imageUrl && (
  <table>
    <thead>
      <tr>
        <th>Generated Image</th>
      </tr>
    </thead>
    <tbody>
      <tr>
        <td>
          <a href={imageUrl} target="_blank" rel="noopener noreferrer">
            <img src={imageUrl} alt="ML Output" width="200" />
          </a>
        </td>
      </tr>
    </tbody>
  </table>
)}
</div>
):(
<div className="history-panel">
<h2>History</h2>
<table>
<thead>
<tr>
<th>Analysis Preference</th>
<th>Login Time</th>
<th>Logout Time</th>
<th>Process Time</th>
</tr>
</thead>
<tbody>
{history.map((entry, index) => (
  <tr key={index}>
    <td>{entry.analysisPreference}</td>
    <td>{new Date(entry.loginTime).toLocaleString()}</td>
    <td>{entry.logoutTime ? new Date(entry.logoutTime).toLocaleString() : "N/A"}</td>
    <td>{entry.processTime ? new Date(entry.processTime).toLocaleString() : "N/A"}</td>
  </tr>
))}
</tbody>

```



```

        </table>
    </div>
    })
</div>
</div>
);
};
export default Dashboard;

```

6. Deployment

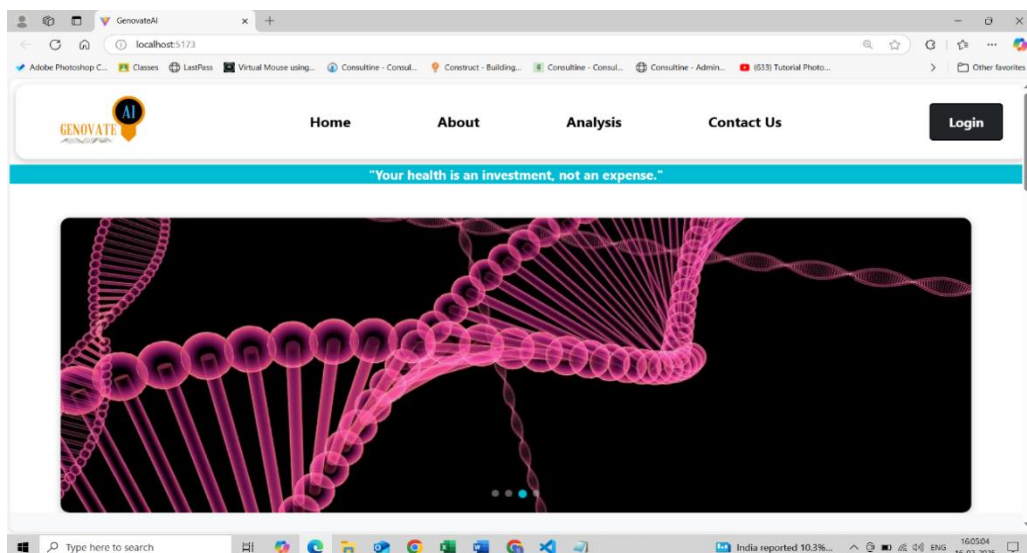
Frontend

npm run dev (for bith frontend and backend)

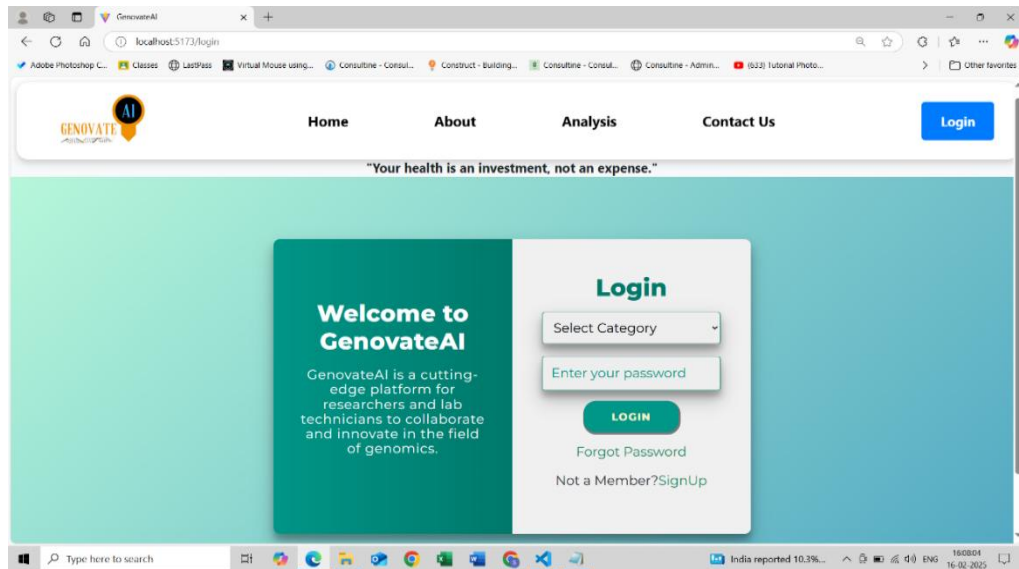
Python manage.py runserver

OUTPUTS

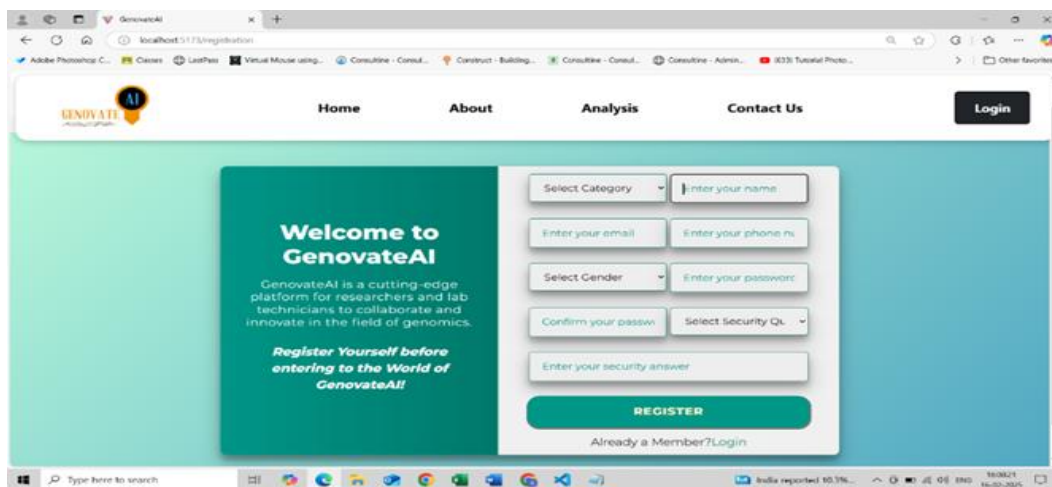
1.Home Page



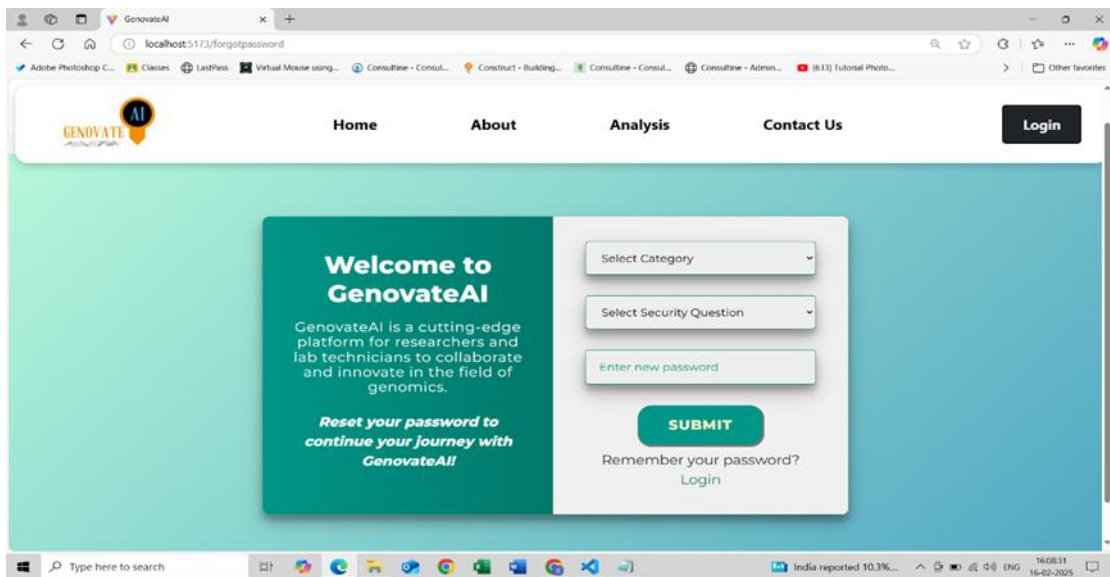
1. Login Page-1:



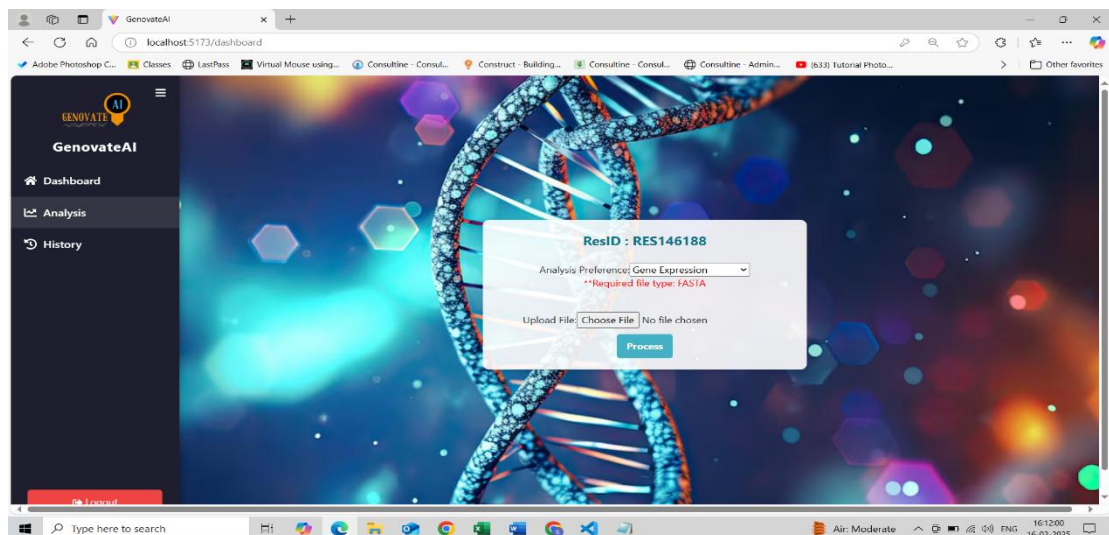
2. Login page 2



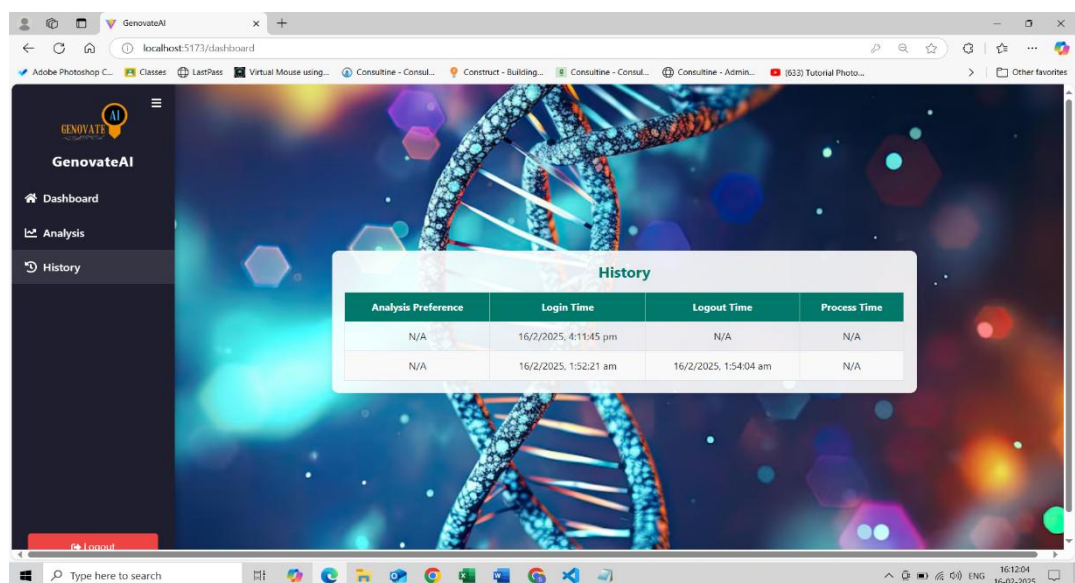
3. Login page 3



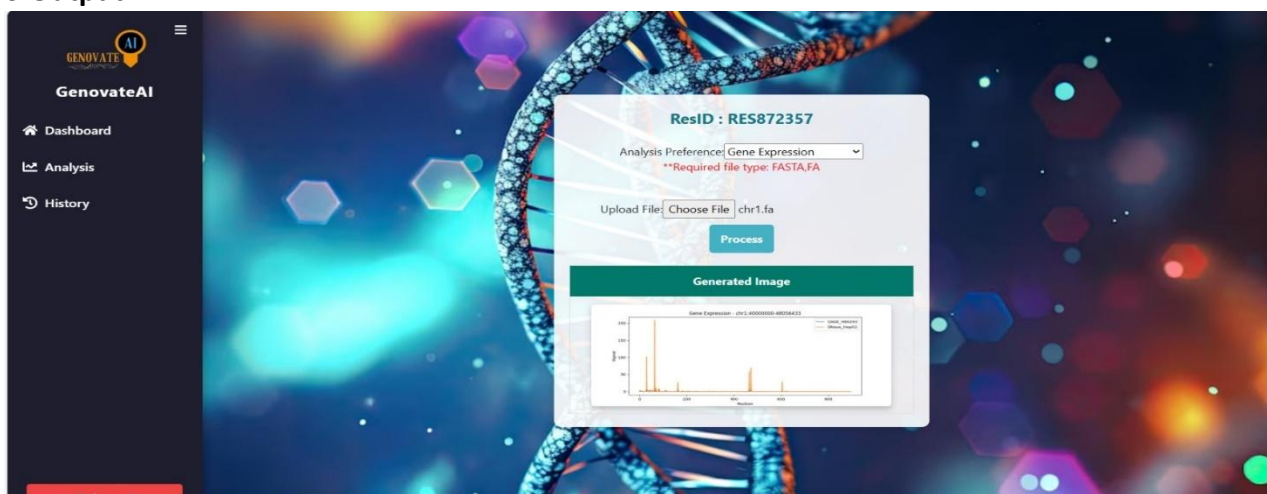
4. Dashboard - Analysis



5. Dashboard-History



6. Output



8. Conclusion

In conclusion, the project successfully integrates four essential components—Data Input, Data Preprocessing, Model, and Postprocessing—to provide a comprehensive solution for analyzing genetic data and making regulatory activity predictions. These components work seamlessly to enhance the accuracy and efficiency of the system, demonstrating the power of a structured workflow in genomic analysis.

Key takeaways:

1. **Data Input:** The ability to handle various file formats, such as ClinVar VCF and genome FASTA files, ensures flexibility in processing diverse data types. This foundational step enables the system to work with relevant genomic information, setting the stage for accurate predictions.
2. **Data Preprocessing:** Extracting sequences around variants and encoding them for machine learning compatibility is crucial for ensuring high-quality input for the model. The preprocessing component streamlines raw genetic data into a format that enhances model performance and reliability.
3. **Model:** By leveraging the Enformer model from TensorFlow Hub, the system efficiently analyzes one-hot encoded sequences to predict regulatory activity. This component brings together cutting-edge machine learning techniques to handle complex genomic data and provide actionable insights.
4. **Postprocessing:** The postprocessing step categorizes the predictions into clinically relevant categories such as "Benign" or "Pathogenic," offering users meaningful output for further analysis. Normalization and categorization ensure the predictions are actionable and ready for clinical interpretation.

Areas for further development:

1. Digital Divide: Enhancing accessibility for users with limited technological resources is critical to ensure equitable use of the system across different regions.
2. Feedback Mechanisms: Developing more sophisticated feedback systems that provide deeper insights into the model's decision-making process can improve user understanding and trust in the system.
3. Collaborative Features: Enabling better peer-to-peer interaction and collaborative learning among researchers and clinicians can enhance the overall effectiveness of the system in a clinical or research setting.
4. Scalability: Expanding the system to handle additional genomic datasets and further improving the model's prediction accuracy are potential areas for future growth.

This project has the potential to revolutionize how genomic data is processed and analyzed, offering a powerful tool for researchers, clinicians, and geneticists. With continued improvements and future developments, this system can contribute significantly to the field of genomics and personalized medicine.

9. REFERENCES

1. Online websites and Resources: Genomics Education Programme: Online resources for genomic research and education (<https://www.genomicseducation.hee.nhs.uk>)
2. NCBI Genomic Data: National Center for Biotechnology Information (<https://www.ncbi.nlm.nih.gov>)
3. Genomic Data Processing and Variant Analysis.
4. Pang, M. et al. (2020). "Computational analysis of genomic variation in a diverse human population." *Nature Communications*. [Link](#)
5. Abyzov, A. et al. (2011). "Mapping and functional analysis of structural variation in the human genome." *Nature*. [Link](#)
6. Angermueller, C. et al. (2017). "Deep learning for computational biology." *Molecular Systems Biology*.
7. Zhou, J. & Troyanskaya, O.G. (2015). "Predicting effects of noncoding variants with deep learning-based sequence model." *Nature Methods*. [Link](#)
8. Enformer Model and Regulatory Activity Prediction