1. **INTRODUCTION**
   1. **OVERVIEW OF THE PROJECT**

The **Real-Time Database Monitoring System** is a comprehensive web-based platform engineered to provide live oversight of database activities. It enables database administrators and system moderators to track queries, monitor data consumption patterns, and oversee overall system health in real time. Built using Django and PostgreSQL, the system incorporates a robust role-based access control (RBAC) mechanism that ensures secure and differentiated access for Admins, Moderators, and Users. Admins can assign roles, manage permissions, and audit activities, while moderators and users interact with the system based on their access levels. The intuitive dashboard displays live data updates, query logs, and system performance indicators, making it a powerful tool for proactive database management and issue detection. The second project, **Heart Disease Prediction System** is a healthcare-focused web application that utilizes machine learning to assess a patient’s risk of heart disease based on several clinical parameters. Developed using Django for the web interface and a Gradient Boosting Classifier model for prediction, the system allows patients and healthcare professionals to input relevant medical data—such as age, cholesterol level, blood pressure, and chest pain type—and receive instant predictions regarding heart disease risk. The application also supports user authentication and role management, enabling secure access for both doctors and patients. By offering quick and data-driven insights, the system serves as a valuable decision-support tool, enhancing preventive care and supporting early diagnosis in clinical settings**.**

* 1. **PROBLEM STATEMENT**

**1.2.1 Real-Time Database Monitoring System**:

Modern applications heavily rely on database systems to manage, store, and retrieve critical data. However, as these systems grow in complexity, real-time visibility into database operations becomes increasingly difficult. Traditional database tools often fall short in providing immediate insights into query execution, data consumption patterns, and user activity. This lack of real-time monitoring can lead to performance bottlenecks, undetected anomalies, unauthorized access, and even critical system failures. Moreover, without a structured and secure access control system, managing multiple users with different roles becomes inefficient and error-prone. Therefore, there is a pressing need for a web-based, role-oriented solution that allows administrators and stakeholders to continuously observe, analyze, and manage database activities in real time through a user-friendly interface.

**1.2.2 Heart Disease Prediction System**:

Cardiovascular diseases, particularly heart disease, are among the leading causes of death globally. Early detection plays a vital role in reducing the severity and impact of such conditions. However, access to medical professionals, diagnostic tools, and regular health screenings remains limited for many, especially in remote or under-resourced areas. The absence of early diagnosis often leads to late-stage detection, reducing the effectiveness of treatment. With the increasing availability of digital health records and medical data, there is an opportunity to develop intelligent systems that can analyze patient information and predict the likelihood of heart disease. Hence, there is a clear need for a fast, reliable, and easy-to-use machine learning-based prediction system that can assist healthcare professionals and individuals in identifying heart disease risks using commonly available health parameters.

* 1. **SCOPE OF THE PROJECT**

1.3.1. **Real-Time Database Monitoring System**:

The Real-Time Database Monitoring System has been successfully implemented as a comprehensive solution for monitoring and managing database activity. The system addresses the operational and security needs of administrators and database managers through a dynamic web interface. Its final scope includes:

a) **Real-Time Monitoring**: Enables administrators to track database queries, user sessions, and system performance metrics live, ensuring rapid response to issues or anomalies.

b) **Role-Based Access Control (RBAC)**: The system supports multiple user roles—Admin, Moderator, and User—each provided with distinct dashboards and feature access according to their responsibilities and permissions.

c) **User & Permission Management**: Admins have full control over user account creation, role assignment, and CRUD permission management through a dedicated admin dashboard, eliminating the need for manual backend configurations.

d) **Scalable and Secure Web Interface**: The final product features a responsive, secure, and intuitive interface built using Django, ensuring accessibility, scalability, and data integrity during real-time usage.

1.3.2. **Heart Disease Prediction System**:

The Heart Disease Prediction System has been developed as a fully functional medical decision-support tool designed to predict heart disease risk using machine learning techniques. The final system scope encompasses:

a) **User-Friendly Input Collection**: The web application allows patients and healthcare providers to input relevant medical parameters such as age, blood pressure, cholesterol levels, and more via a streamlined form.

b) **Accurate ML-Based Prediction**: A trained Gradient Boosting Classifier model processes the inputs and predicts the likelihood of heart disease. The model was trained and tested on standard medical datasets for robust accuracy.

c) **Instant Results & Visualization**: The system displays the prediction output instantly, allowing users to make quick, informed decisions regarding further medical consultation.

d) **Support for Preventive Healthcare**: By offering automated predictions, the system assists both individuals and medical professionals in the early detection of heart disease risk, thereby promoting preventive diagnosis and timely intervention.

* 1. **HARDWARE SPECIFICATION**

The hardware environment used during the development and testing of both projects is as follows:

* **Processor**: Intel Core i5 or higher
* **RAM**: Minimum 8 GB (16 GB recommended for machine learning tasks)
* **Storage**: Minimum 256 GB SSD
* **Operating System**: Windows 10 / Linux Ubuntu (for server deployment and testing)
* **Internet Connectivity**: Required for package installation, model training/testing, and API usage
* **Display**: Standard 1080p Monitor
* **GPU (Optional)**: NVIDIA GPU for faster machine learning computations (if available)
  1. **SOFTWARE SPECIFICATION**

1.5.1 **Real-Time Database Monitoring System**:

1. Backend: Django (Python)
2. Frontend: HTML, CSS, JavaScript
3. Database: PostgreSQL
4. Others:

* Raw SQL queries using Django's cursor
* Role-Based Access Control (RBAC) system
* Server-side security and session management

1.5.2 **Heart Disease Prediction System**:

1. Machine Learning: Scikit-learn, Pandas, NumPy
2. Backend: Django and Rest framework (Python)
3. Frontend: HTML, CSS, JavaScript
4. Database: SQLite (for lightweight ML model integration)

1.5.3. **Common Development Environment**

1. **IDE/Editor**: Visual Studio Code
2. **Browser**: Google Chrome / Firefox for testing
3. **Version Control**: Git (for local version tracking)
   1. **TECHNOLOGIES USED - DESCRIPTION**

This section provides an overview of all the tools, libraries, and frameworks utilized in the development of the **Real-Time Database Monitoring System** and the **Heart Disease Prediction System**, including both software development and machine learning technologies.

### ****1.5.1 Web Development & Backend Technologies****

### ****Django (Python Web Framework)****

Django is a powerful, high-level web framework built on Python that encourages rapid development and clean, pragmatic design. It was used extensively to develop both projects, handling the backend logic, user authentication, admin dashboard, and routing. Django’s security features, scalability, and built-in admin panel made it the ideal choice for web development.

### ****Django REST Framework (DRF)****

Django REST Framework is a toolkit built on Django for developing Web APIs. It simplifies the creation of RESTful APIs by handling serialization, authentication, permissions, and view sets. In this project, DRF was used to expose data for real-time monitoring and to serve predictions from the machine learning model in the Heart Disease Prediction System.

### ****Visual Studio Code (VS Code)****

VS Code is a lightweight, cross-platform code editor with extensive support for Python and Django development. It was used for writing, debugging, and testing code in both projects. Useful extensions like Python Linter, Django Snippets, and REST Client helped streamline the development process.

### ****HTML (HyperText Markup Language)****

HTML is the standard markup language used to create the structure of web pages. In both projects, HTML was used to design user-facing forms, dashboards, and result display sections.

### ****CSS (Cascading Style Sheets)****

CSS is used to style HTML elements and improve the visual aesthetics of the web pages. It was used for styling forms, buttons, tables, and layout designs to ensure a user-friendly interface in both projects.

### ****JavaScript****

JavaScript was used to add interactivity and dynamic content to the web applications. It enabled features like form validation, real-time feedback, and interactive user interfaces, enhancing the overall user experience.

### ****PostgreSQL / MySQL****

These are open-source relational database management systems used for storing structured data. The Real-Time Database Monitoring System used one of these systems to store user accounts, login records, roles, query logs, and other essential information.

### ****Raw SQL Queries via Django Cursor****

Due to limitations on the server that restrict the use of Django's ORM and migrations, raw SQL queries were executed using Django’s cursor object. This method allows for direct database manipulation, including table creation, data insertion, and user role management.

### ****Python (Programming Language)****

Python is the core programming language used across both projects. It is known for its readability, versatility, and rich ecosystem of libraries for web development, data science, and machine learning.

## ****1.5.2 Machine Learning Technologies****

The **Heart Disease Prediction System** uses machine learning algorithms to predict the likelihood of a person having heart disease based on input parameters. The following technologies and algorithms were used:

### ****Pandas****

Pandas is a powerful and flexible data analysis library built on top of Python. It was used to read, clean, and manipulate the dataset containing patient health records. With Pandas, data could be filtered, transformed, and prepared efficiently before feeding it into machine learning models.

### ****Gradient Boosting Classifier****

Gradient Boosting is an ensemble machine learning technique that builds models in a stage-wise fashion and generalizes them by optimizing a loss function. Typically based on decision trees, Gradient Boosting combines multiple weak learners to form a strong predictive model. In this project, it was used to classify whether a person is at risk of heart disease. It is particularly effective for handling complex datasets and improving model accuracy.

### ****Logistic Regression****

Logistic Regression is a statistical model used for binary classification tasks. It estimates the probability that a given input belongs to a particular category. In the Heart Disease Prediction System, Logistic Regression was employed to predict whether a person is likely to have heart disease based on medical input data. It is widely used due to its simplicity, efficiency, and interpretability in healthcare-related classification problems.

1. **OBJECTIVE**

This section outlines the overall objectives, research background, rationale, purpose, and proposed approaches adopted during the development of two projects: **Real-Time Database Monitoring System** and **Heart Disease Prediction System**. Both projects are developed to address real-world problems using modern technologies like Django, Machine Learning, and SQL-based data management. The goal is to build intelligent, user-friendly systems that are scalable, secure, and practically useful for their respective domains.

**2.1 GOALS**

**2.1.1 Real-Time Database Monitoring System**

The primary objective of this project is to create a web-based platform that continuously monitors the activity of a relational database in real-time. The specific goals include:

1. **Real-Time Insight:** Build a system capable of tracking database activity instantly, including query execution, user logins, and interactions with the system.
2. **Role-Based Access Control (RBAC):** Implement user authentication and authorization using a role-based access model where users are grouped as Admins, Moderators, or Standard Users with different permissions.
3. **Security and Accountability:** Enable the logging of all critical actions in the system to ensure data integrity, traceability, and compliance with security protocols.
4. **Migration-Free Setup:** Develop a backend architecture that avoids Django’s ORM migrations by relying on raw SQL queries executed via Django’s database cursor. This ensures the system is compatible with restricted environments where migration commands cannot be executed.

**2.1.2 Heart Disease Prediction System**

The objective of this project is to provide a cost-effective, intelligent, and accessible prediction system for assessing the risk of heart disease using machine learning. The key goals are:

1. **Accurate Health Risk Prediction:** Utilize supervised learning algorithms such as Logistic Regression and Gradient Boosting Classifier to predict the presence of heart disease from patient data.
2. **Assist Healthcare Providers:** Deliver a tool that augments medical diagnosis by providing fast, data-driven predictions to help doctors make informed decisions.
3. **User-Focused Interface:** Develop a web portal where users can easily input health-related information and receive predictive results with interpretive feedback.
4. **Efficient Deployment:** Design the backend in a way that handles pre-trained serialized ML models without needing to retrain every time, ensuring scalability and real-time usability.

**2.2 LITERATURE SURVEY**

Understanding prior research and existing systems is essential for developing improved solutions. This literature survey summarizes key findings that informed the design and features of both systems.

**2.2.1 Real-Time Database Monitoring System**

Modern database systems offer various monitoring solutions such as **Oracle Enterprise Manager**, **MySQL Monitor**, and **Prometheus-Grafana stacks**, which are robust but often not customizable or accessible for smaller-scale implementations.

1. According to *Bala and Kumar (2020)*, RBAC significantly enhances database security and minimizes unauthorized access risks.
2. *Sharma et al. (2021)* identified the need for affordable, adaptable, and custom-built monitoring solutions for SMEs (Small and Medium Enterprises).
3. *Kumar and Raj (2019)* discussed constraints in certain deployment environments where developers do not have permission to run Django migrations. This emphasized the utility of raw SQL queries for dynamic database management.

The proposed system draws from these insights to build a **portable**, **migration-free**, and **real-time monitoring solution** tailored to diverse user roles and actions.

**2.2.2 Heart Disease Prediction System**

The integration of machine learning in healthcare diagnostics has shown immense potential in recent years.

1. The **Framingham Heart Study** is a well-known dataset that helped shape ML-driven approaches to cardiovascular disease risk assessment.
2. *Khan et al. (2020)* demonstrated that Gradient Boosting Classifiers outperform traditional models like Logistic Regression in medical prediction tasks.
3. *Saxena and Sharma (2022)* found that patients were more likely to follow preventive steps when predictions were accessible online with explanatory outputs.

These findings reinforce the decision to use **ensemble learning** for classification, supported by **user-friendly interfaces**, offering patients actionable insights and supporting early intervention.

**2.3 NEED OF THE SYSTEM**

The development of both systems is rooted in addressing critical challenges faced in database management and healthcare diagnostics.

**2.3.1 Real-Time Database Monitoring System**

1. As organizations scale, managing multiple user interactions with sensitive data becomes increasingly complex and vulnerable to misuse or unauthorized access.
2. Existing monitoring tools are often prohibitively expensive or lack customization to enforce granular access policies based on roles.
3. The need for **real-time query visibility** is paramount in high-security environments such as financial or government institutions.
4. Server-level restrictions often make Django migrations unviable; hence, the need for raw SQL-based user and permission management is a practical requirement.

**2.3.2 Heart Disease Prediction System**

1. Heart disease continues to be one of the leading causes of death globally. Early diagnosis significantly improves treatment outcomes.
2. Manual data interpretation in medical assessments can be slow, inconsistent, and subject to human error.
3. Many rural or under-resourced healthcare facilities lack access to advanced diagnostic tools. A web-based prediction system fills this gap.
4. There is a growing need for **AI-powered**, **accessible**, and **interpretive** tools that can guide both patients and healthcare professionals in understanding health risks.

**2.4 PURPOSE**

The purpose of both projects extends beyond technical implementation. They are intended to serve as scalable and meaningful contributions to the fields of data security and digital healthcare.

**2.4.1 Real-Time Database Monitoring System**

1. To provide a centralized web platform where users can **monitor database activity**, view logs, and manage access based on role assignments.
2. To ensure **compliance and security** by logging all user interactions and query executions in the database.
3. To streamline **role management** through intuitive dashboards, removing the dependency on command-line tools or admin panels.
4. To promote **data governance** and organizational transparency by making all changes traceable and role-specific.

**2.4.2 Heart Disease Prediction System**

1. To enable prediction of cardiovascular risk using **quantifiable patient data**, such as age, cholesterol, and blood pressure.
2. To **reduce burden on physicians** by automating initial screening and assessments using machine learning.
3. To give **instant, user-friendly feedback** to individuals who input their data, guiding them toward preventive care.
4. To democratize access to health prediction tools, especially in underserved regions where diagnostic facilities may be limited.

**2.5 PROPOSED SYSTEMS**

**2.5.1 Real-Time Database Monitoring System**

The proposed system is a secure, role-based, and fully web-enabled platform for monitoring database operations.

1. Developed using **Django**, it integrates login functionality, permission controls, and custom dashboards for different user roles.
2. It executes **raw SQL queries** for tasks like user creation, permission updates, and query logging — making it suitable for restricted environments.
3. A dynamic **logging mechanism** captures each query made by users in real time, providing visibility and transparency to system administrators.
4. The Admin panel supports full **CRUD operations**, including adding, editing, and deleting users, along with the ability to change their roles dynamically via the dashboard.

**2.5.2 Heart Disease Prediction System**

The system is an intelligent, user-friendly, and responsive platform that predicts heart disease likelihood using ML models.

1. Built on **Django** and **Django REST Framework**, it leverages serialized models to deliver instant predictions without re-training.
2. Uses **Pandas** for data processing and integrates **Logistic Regression** and **Gradient Boosting** for predictive analysis.
3. The frontend allows patients or users to input health-related values through a clean interface and view results instantly.
4. The output includes not just a prediction, but also **interpretative messaging**, encouraging users to take preventive steps or consult a healthcare provider.
5. **SCHEDULE OF ACTIVITIES**

This section outlines the planning, feasibility, implementation strategy, and execution timeline for the two projects. The development lifecycle followed a structured approach, ensuring timely delivery and systematic progress.

* 1. **FEASIBILTY STUDY**

The feasibility study is a crucial initial step to evaluate whether the proposed systems (Real-Time Database Monitoring System and Heart Disease Prediction System) are viable in terms of technology, operation, economics, and legal constraints.

**3.1.1 Technical Feasibility**

1. Both systems are built using **Python**, **Django**, and **PostgreSQL**, all of which are open-source and well-supported.
2. The Heart Disease Prediction System integrates **Machine Learning** using libraries such as **scikit-learn**, ensuring advanced analytics.
3. The Real-Time Database Monitoring System uses **raw SQL commands** to handle constrained environments where Django migrations are restricted.
4. Hosting can be done on local servers or cloud platforms (Heroku, AWS, or PythonAnywhere).

**3.1.2 Operational Feasibility**

1. The user interface of both systems is intuitive and web-based, making them user-friendly for doctors, admins, and users without requiring training.
2. RBAC (Role-Based Access Control) ensures operational control and security for the monitoring system.
3. The Heart Disease Prediction System allows for individual risk assessment and supports healthcare operations seamlessly.

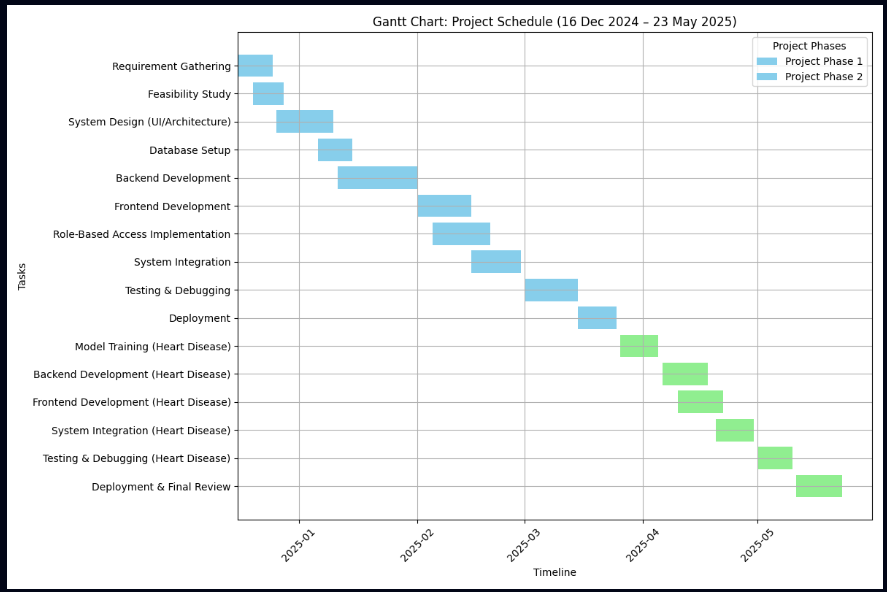
**3.1.3 Economic Feasibility**

1. Minimal costs are involved due to the use of open-source technologies.
2. Maintenance and hosting are low-cost and scalable.
3. Reduces healthcare costs by enabling self-assessment and early diagnosis.

**3.1.4 Legal and Ethical Feasibility**

1. No confidential data is stored without user consent.
2. Ethical use of machine learning ensures no discrimination based on user data.
3. All systems follow privacy protocols and comply with basic cybersecurity guidelines.

**3.2 GANTT CHART**



**3.3 TIMELINE**

**3.3.1 TIMELINE OF THE ACTIVITIES:**

|  |  |  |  |
| --- | --- | --- | --- |
| S. No. | Activity | Start Date | End Date |
| 1 | Project Planning and Requirement Analysis for RDMS | 16-Dec-2024 | 20-Dec-2024 |
| 2 | Literature Review and Technology Selection | 22-Dec-2024 | 27-Dec-2024 |
| 3 | Feasibility Study (Technical, Operational, etc.) | 27-Dec-2024 | 30-Dec-2024 |
| 4 | Designing Use Case, DFD, ER, and Other Diagrams | 29-Dec-2024 | 03-Jan-2025 |
| 5 | System Design (Architecture + UI Planning) | 05-Jan-2025 | 10-Jan-2025 |
| 6 | Designing System Architecture and Database Schema | 12-Jan-2025 | 17-Jan-2025 |
| 7 | Backend Development using Django Framework | 19-Jan-2025 | 24-Jan-2025 |
| 8 | Frontend Design using HTML, CSS, and JavaScript | 27-Jan-2025 | 31-Jan-2025 |
| 9 | Connecting the the tables with the action buttons | 02-Feb-2025 | 07-Feb-2025 |
| 10 | Implementing Role-Based Access Control System | 09-Feb-2025 | 14-Feb-2025 |
| 11 | Integration of Raw SQL Queries for Role & Data Management | 16-Feb-2025 | 21-Feb-2025 |
| 12 | Testing and Debugging of Web Applications | 24-Feb-2025 | 28-Feb-2025 |
| 13 | UI/UX Improvement and Data Validation Checks | 02-March-2025 | 07-March-2025 |
| 14 | Finalizing Admin Dashboard and Logging Mechanism | 09-Mar-2025 | 12-Mar-2025 |
| 15 | Final Testing, Debugging and Report Writing | 16-Mar-2025 | 21-Mar-2025 |
| 16 | Assignment of 2nd project and its information gathering | 23-Mar-2025 | 28-Mar-2025 |
| 17 | Submission and Mid-Term Project Presentation | 27-Mar-2025 | 28-Mar-2025 |
| 18 | Requirement Gathering of HDPS | 31-Mar-2025 | 04-Apr-2025 |
| 19 | Feasibility Study (ML Suitability, Ethics, etc.) | 07-Apr-2025 | 11-Apr-2025 |
| 20 | System Design (Model Workflow + UI) | 14-Apr-2025 | 17-Apr-2025 |
| 21 | Database Setup (Patient Records, Feedback | 21-Apr-2025 | 25-Apr-2025 |
| 22 | Model Training & Evaluation (ML Model) | 28-Apr-2025 | 02-May-2025 |
| 23 | Backend Development (Model Integration) | 05-May-2025 | 09-May-2025 |
| 24 | Frontend Development (Prediction UI) | 12-May-2025 | 16-May-2025 |
| 25 | Final Review & Documentation | 19-May-2025 | 22-May-2025 |

**3.3.2 Description of Key Activities**

1. **Project Planning and Requirement Analysis**  
   Initial brainstorming and identification of system requirements, user roles, and features. Defined clear objectives and success criteria for both projects.
2. **Literature Review and Technology Selection**  
   Analyzed existing solutions, researched current technologies, and finalized Django, Python, and machine learning tools for implementation.
3. **Designing Use Case, DFD, ER, and Other Diagrams**  
   Created critical design documentation:
   1. **Use Case Diagram**: Shows user interactions.
   2. **DFD (Data Flow Diagram)**: Describes how data flows in the system.
   3. **ER Diagram**: Illustrates database structure.
   4. **Class Diagrams**: Describes object relationships (if needed).  
      These diagrams acted as blueprints before actual development.
4. **System Architecture and Database Design**  
   Defined system modules, interactions, and designed the database schema using MySQL/PostgreSQL.
5. **Backend Development using Django**  
   Developed views, models, and URL routing logic. Created APIs and connected backend to database.
6. **Role-Based Access Control System**  
   Implemented user-role management (Admin, Moderator, User) with Django Groups and custom permissions.
7. **Frontend Development**  
   Built responsive UI using HTML, CSS, JavaScript and Django templates for dashboards and prediction views.
8. **Raw SQL Integration**  
   Used Django's cursor() for executing raw SQL queries to bypass server-side restrictions like makemigrations.
9. **ML Model Selection and Dataset Preparation**  
   Cleaned and processed the heart disease dataset. Chose Logistic Regression and Gradient Boosting models for training.
10. **Model Training**  
    Trained and validated models using scikit-learn. Achieved high accuracy with Gradient Boosting.
11. **Model Serialization and API Setup**  
    Saved model using joblib or pickle and created REST API endpoints with Django REST Framework for prediction.
12. **Web Integration of Model**  
    Integrated prediction output into the frontend to allow users to enter values and get heart disease prediction results.
13. **Testing & Debugging**  
    Performed unit and integration testing. Fixed bugs and optimized system behavior.
14. **UI/UX Enhancement & Data Validation**  
    Improved frontend visuals and implemented input validation to ensure data quality.
15. **Admin Dashboard Finalization**  
    Built dynamic dashboards with logging, user control, and a user table for admin actions.
16. **Final Testing and Report Writing**  
    Conducted end-to-end testing and prepared a detailed report summarizing the technical aspects and implementation.
17. **Project Submission and Presentation**  
    Delivered final version of both projects and presented features and learnings to the mentor panel.

**3.4 METHODOLOGY**

Both projects followed the Waterfall Model of software development due to its structured and linear approach, which best suited the academic environment and fixed project timeline.

**Phases in Methodology:**

1. **Requirement Analysis:**
   1. Interviews and study of existing systems
   2. Use case identification and data flow planning
2. **Feasibility Study:**
   1. Assessed cost, operational need, and legal risks
   2. Focused on Django’s migration limitation (Real-Time Monitoring)
3. **System Design:**
   1. Wireframes and architecture defined
   2. Role access levels and page flows were planned
4. **Development:**
   1. Backend in Python/Django
   2. Frontend using HTML, CSS, JavaScript (Bootstrap)
   3. ML model built with scikit-learn and integrated into Django views
5. **Testing:**
   1. Unit, integration, and system testing
   2. Dataset-based accuracy tests for ML model
6. **Deployment:**
   1. Hosted on local or cloud server
   2. Made accessible for demonstration and review
7. **Documentation & Review:**
   1. Comprehensive documentation of use cases, code, testing, and system functionality

**4. PROJECTS IMPLEMENTATION**

The implementation of both systems was carefully planned and executed to meet the project objectives while addressing technical constraints and ensuring usability. Below is a detailed overview of the implementation process for each system.

**4.1** **SYSTEM ARCHITECTURE**

The system architectures for both the Real-Time Database Monitoring System and the Heart Disease Prediction System are designed to be modular, scalable, and efficient, leveraging modern web and machine learning technologies. Each system is structured to address its specific objectives while ensuring security, usability, and performance.

**4.1.1 Real-Time Database Monitoring System Architecture**

The Real-Time Database Monitoring System is built on a layered architecture that emphasizes role-based access control (RBAC) and real-time data tracking. The system is developed using the Django web framework, ensuring a robust and secure foundation for managing database operations. The architecture comprises the following key layers:

* **Presentation Layer**: This layer provides a dynamic and responsive user interface built with HTML, CSS, and JavaScript. It renders role-specific dashboards for Admins, Moderators, and Standard Users, ensuring that each user sees only the features and data relevant to their permissions. The interface is styled using Bootstrap for consistency and responsiveness across devices.
* **Application Layer**: Powered by Django’s backend framework, this layer handles core functionalities such as user authentication, session management, permission enforcement, and query logging. Django views process requests, manage business logic, and facilitate communication between the frontend and the database.
* **Database Layer**: PostgreSQL serves as the relational database management system, storing user accounts, role assignments, query logs, and system metadata. Due to server constraints prohibiting Django’s ORM migrations, raw SQL queries are executed via Django’s cursor object to perform CRUD operations directly.
* **RBAC Engine**: A custom role-based access control mechanism is implemented using raw SQL to manage user roles (Admin, Moderator, User) and their respective permissions. This ensures granular control over system access and operations, enhancing security and accountability.

**Architecture Highlights**:

* Dynamic rendering of dashboards tailored to user roles, improving usability and access control.
* Real-time logging of all database queries and user activities, enabling immediate detection of anomalies or unauthorized actions.
* A centralized admin dashboard for user management, permission control, and activity auditing, designed to function without relying on Django migrations.

**4.1.2 Heart Disease Prediction System Architecture**

The Heart Disease Prediction System adopts a modular, API-driven architecture that integrates machine learning with a web-based interface to deliver real-time health predictions. The system is designed to be user-friendly, scalable, and efficient, with clear separation of concerns between its components.

* **Frontend Interface**: Built with HTML, CSS, and JavaScript, the frontend provides an intuitive form for users to input medical parameters such as age, cholesterol levels, and blood pressure. The interface is styled for clarity and accessibility, ensuring users can easily interact with the system.
* **Backend Layer**: Django, combined with Django REST Framework (DRF), manages routing, request handling, and API endpoints. DRF enables seamless communication between the frontend and the machine learning models, ensuring efficient data transfer and processing.
* **Machine Learning Processing Layer**: This layer hosts the trained machine learning models (Gradient Boosting Classifier and Logistic Regression) that predict the likelihood of heart disease based on user inputs. The models are serialized using joblib for quick loading and prediction without retraining.
* **Data Handling Layer**: User inputs are validated and preprocessed using the Pandas library to ensure data quality and compatibility with the ML models. This layer handles data transformation, such as normalization and encoding, before passing inputs to the prediction engine.
* **Database Layer**: SQLite is used to store user inputs, prediction results, and feedback logs. This lightweight database ensures efficient storage and retrieval for small-scale deployments, with the potential to scale to larger systems like PostgreSQL if needed.

**Architecture Highlights**:

* Instantaneous predictions delivered through a REST API, ensuring low latency and high responsiveness.
* Separation of concerns between frontend, backend, and ML components, enabling modular development and maintenance.
* Reusable ML models serialized for efficient deployment, reducing computational overhead during predictions.

**4.2** **MODULES DESCRIPTION**

The modular design of both systems ensures that each component is independent yet interoperable, facilitating maintenance, scalability, and future enhancements. Below is a detailed description of the key modules for each system.

**4.2.1 Real-Time Database Monitoring System Modules**

1. **User Authentication & Role Management:** 
   1. Provides secure login and logout functionality using Django’s authentication system.
   2. Implements manual role assignment (Admin, Moderator, User) via raw SQL queries, bypassing Django’s ORM migrations.
   3. Allows Admins to view, edit, and delete user accounts and permissions through a centralized dashboard, ensuring granular control over access rights.
2. **Real-Time Query Logging:** 
   1. Captures every database query executed by users, including timestamps, user IDs, and query details.
   2. Enables Admins to review logs for auditing purposes, helping identify suspicious or unauthorized activities in real time.
   3. Logs are stored in PostgreSQL and accessible through the Admin dashboard.
3. **Dashboard Views:** 
   1. Admin Dashboard: Offers full control over user management, permission editing, and query log review. Includes CRUD functionality for user accounts and roles.
   2. Moderator Dashboard: Provides limited access to analytics and query execution, with restricted permissions to prevent unauthorized actions.
   3. User Dashboard: Allows standard users to execute queries and view basic system metrics, tailored to their role.
4. **Raw SQL Execution Layer:** 
   1. Handles all database operations (create, read, update, delete) using raw SQL queries executed via Django’s cursor object.
   2. Ensures compatibility with migration-restricted server environments by manually managing table creation, data insertion, and permission updates.
   3. Logs all CRUD operations for transparency and accountability.

**4.2 Heart Disease Prediction System Modules**

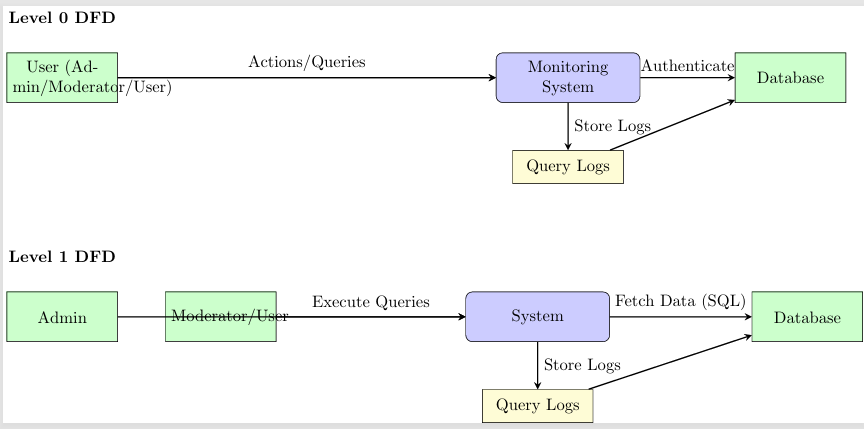
1. **User Input Interface:** 
   1. Features an HTML form for users to input health parameters such as age, cholesterol, resting blood pressure, and other medical attributes.
   2. Implements client-side and server-side data validation to ensure accurate and reliable inputs before processing.
   3. Styled with CSS and JavaScript for a responsive and user-friendly experience.
2. **Machine Learning Model Loader:** 
   1. Loads serialized Gradient Boosting Classifier and Logistic Regression models using joblib for efficient predictions.
   2. Processes validated user inputs and generates predictions on the likelihood of heart disease.
   3. Ensures minimal latency by avoiding retraining during runtime.
3. **Prediction Output Module:** 
   1. Displays prediction results with clear, interpretable messages (e.g., “Low Risk,” “Moderate Risk,” “High Risk”).
   2. Provides contextual health guidance based on the prediction, encouraging users to seek medical advice when necessary.
   3. Uses Bootstrap for consistent and visually appealing output presentation.
4. **Feedback & History Logging:** 
   1. Allows users to submit feedback on their prediction experience, which is stored in SQLite for future analysis.
   2. Logs prediction history, including input parameters and results, enabling users or doctors to track health trends over time.
   3. Supports optional history review for authorized users, such as healthcare professionals.

**4.3 DFD & CLASS DIAGRAMS**

This section presents the system modeling and visualization for both the Real-Time Database Monitoring System and the Heart Disease Prediction System using Data Flow Diagrams (DFDs) and Unified Modeling Language (UML) diagrams. These diagrams provide a clear representation of data flows, system interactions, and structural relationships, serving as essential tools for understanding system behavior, user interactions, and data organization. The DFDs illustrate the flow of data through each system, while the UML diagrams (Use Case and Class Diagrams) depict user interactions and system structure, respectively.

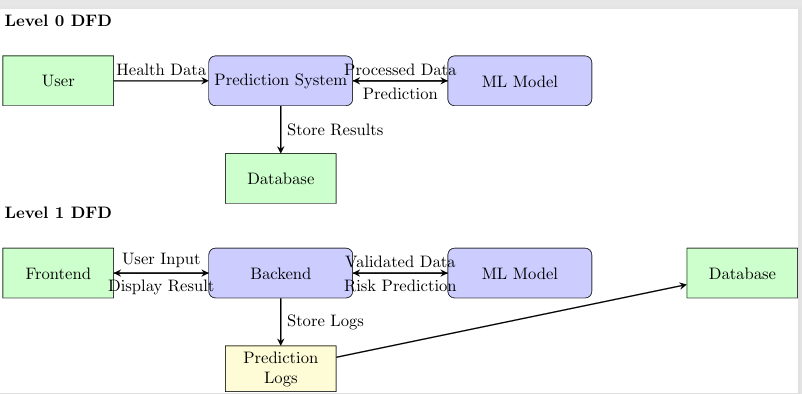
**4.3.1 Real-Time Database Monitoring System**

**Data Flow Diagrams (DFD)**

1. **Level 0 DFD**: The Level 0 DFD provides a high-level overview of the Real-Time Database Monitoring System. Users (Admin, Moderator, or Standard User) interact with the system by providing login credentials and executing actions or queries. The system authenticates users against the PostgreSQL database and logs all queries and actions in real time, storing them in a dedicated log table for auditing.
2. **Level 1 DFD**: The Level 1 DFD details specific processes within the system:
3. **Admin**: Performs user management, permission editing, and log viewing.
4. **Moderator/User**: Executes queries and views role-specific dashboards.
5. **System**: Fetches data using raw SQL queries, logs actions with timestamps and user IDs, and updates the database.

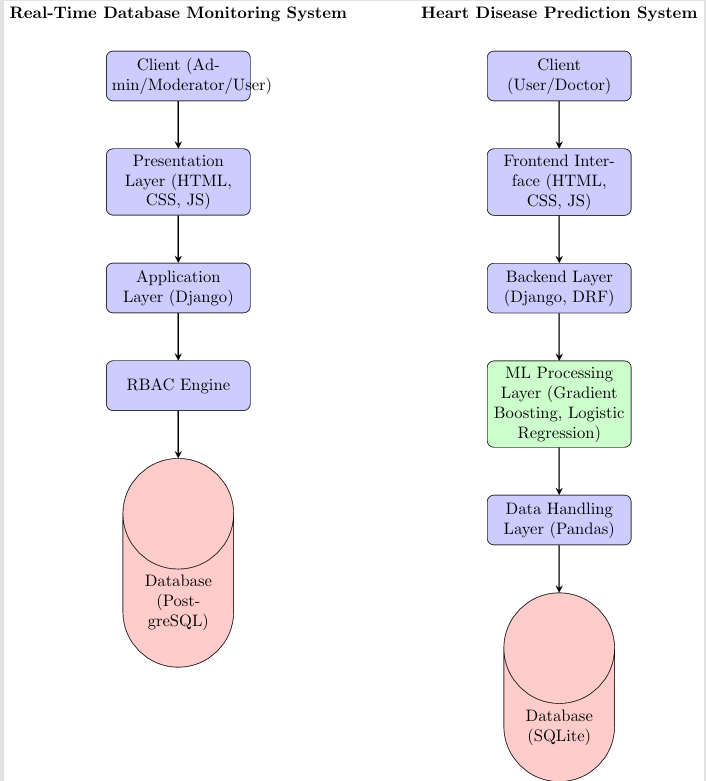
**4.3.2 Heart Disease Prediction System**

**Data Flow Diagrams (DFD)**

1. **Level 0 DFD**: The Level 0 DFD illustrates the high-level process where users input health data (e.g., age, cholesterol) into the Prediction System. The system processes the data through a machine learning model, returns a prediction (e.g., risk level), and logs the results in the SQLite database.
2. **Level 1 DFD**: The Level 1 DFD provides a detailed view:
3. **Frontend**: Accepts user inputs and displays prediction results.
4. **Backend**: Validates inputs, passes data to the ML model, logs results, and returns predictions to the frontend.
5. **ML Model**: Processes validated data to generate a risk-level prediction.

.

**Class Diagram**:

1. **Classes**:
   1. **User**: Attributes include name, email, password, medical\_history.
   2. **PredictionModel**: Attributes include model\_file; methods include predict(data).
   3. **Prediction**: Attributes include user\_id, input\_parameters, result, timestamp.
2. **Relationships**: One-to-many from User to Prediction and from PredictionModel to Prediction.

**4.4 IMPLEMENTATION**

The implementation of both systems was carefully planned and executed to meet the project objectives while addressing technical constraints and ensuring usability. Below is a detailed overview of the implementation process for each system.

**4.4.1 Real-Time Database Monitoring System Implementation**

The system was developed using the Django web framework, leveraging its robust features for authentication, session management, and templating while adapting to server constraints that prohibited ORM migrations.

1. **Authentication:** Extended Django’s built-in user authentication system to support role-based access control. Custom SQL scripts were written to manage user groups and permissions directly in PostgreSQL, ensuring compatibility with migration-restricted environments.
2. **Real-Time Monitoring:** Implemented middleware to capture and log database queries per user session. Each query is timestamped and associated with the user’s ID, stored in a dedicated log table for real-time auditing.
3. **Admin Dashboard:** Developed using Django templates and Bootstrap, the Admin dashboard provides a responsive interface for managing users, editing permissions, and reviewing query logs. The dashboard supports dynamic updates to user roles without requiring migrations.
4. **Deployment:** Configured the system to run on a PostgreSQL database, with raw SQL queries ensuring portability across different server environments. The system was tested for performance and scalability, achieving real-time logging with minimal latency.

**4.4.2 Heart Disease Prediction System Implementation**

The Heart Disease Prediction System was implemented as a Django-based web application integrated with a machine learning pipeline, delivering accurate and accessible predictions.

1. **Data Preparation:** The heart disease dataset was cleaned and preprocessed using Pandas, handling missing values, normalizing features, and encoding categorical variables to prepare the data for model training.
2. **Model Training:** Two machine learning models—Gradient Boosting Classifier and Logistic Regression—were trained using scikit-learn. The models were evaluated for accuracy, precision, recall, and AUC score, with the Gradient Boosting Classifier achieving superior performance (approximately 89% accuracy).
3. **Model Integration:** The best-performing model was serialized using joblib and integrated into the Django backend via REST API endpoints created with Django REST Framework. This ensured efficient loading and prediction without retraining.
4. **Frontend Interface:** Developed a responsive HTML form styled with CSS and Bootstrap, allowing users to input medical parameters. JavaScript was used for client-side validation and dynamic feedback, enhancing the user experience.
5. **Output Presentation:** Prediction results are displayed instantly in a clear and engaging format, with accompanying health guidance. The interface is designed to be intuitive for both patients and healthcare professionals.

**4.4 TESTING & EVALUATION**

Testing and evaluation were critical to ensuring the reliability, security, and performance of both systems. Comprehensive testing strategies were employed to validate functionality, accuracy, and user experience.

**4.4.1 Real-Time Database Monitoring System**

**Testing Approach:**

1. **Unit Testing:** Tested individual components, including Django views, authentication functions, and raw SQL query execution.
2. **Integration Testing:** Verified that role-specific dashboards rendered correctly and interacted seamlessly with the database.
3. **Security Testing:** Conducted tests with test accounts to ensure restricted access and prevent unauthorized actions.
4. **Manual Testing:** Simulated user activities (e.g., query execution, permission changes) to validate real-time logging.

**Test Cases:**

1. Admin can log in, view, and edit user accounts and permissions.
2. Moderator is restricted from accessing the admin panel.
3. All executed queries are logged with accurate timestamps and user IDs.
4. Unauthorized access attempts are blocked with appropriate error messages.

**Evaluation:**

1. The system demonstrated portability across migration-restricted server environments.
2. Role enforcement achieved 100% accuracy, ensuring secure access control.
3. Real-time query logging exhibited a delay of less than 1 second, meeting performance requirements.

**4.4.2 Heart Disease Prediction System**

**Testing Approach:**

1. **Unit Testing:** Validated ML model loading, prediction logic, and form input validation.
2. **Model Testing:** Evaluated model performance using metrics such as accuracy, precision, recall, and AUC score on a test dataset.
3. **UI Testing:** Tested form behavior, including edge cases (e.g., invalid or negative inputs).
4. **Integration Testing:** Ensured seamless interaction between the frontend, backend, and ML model.

**Model Evaluation Metrics:**

1. **Accuracy: ~**89% (Gradient Boosting Classifier outperformed Logistic Regression).
2. **Precision:** ~85%
3. **Recall:** ~87%
4. **AUC Score:** 0.91

**Test Cases:**

1. Valid user inputs produce accurate predictions with clear results.
2. Invalid inputs trigger appropriate validation error messages.
3. ML model loads correctly without requiring retraining.
4. Feedback form submits successfully and logs are stored in SQLite.

**Evaluation:**

1. The system delivered real-time predictions with an average latency of under 2 seconds.
2. The interface was intuitive and accessible, suitable for both patients and healthcare professionals.
3. The system is scalable for deployment in clinical or telehealth settings, with potential for integration with larger databases.

* 1. **CONCLUSIONS**

This section consolidates the key insights, outcomes, limitations, and future potential of the Real-Time Database Monitoring System and the Heart Disease Prediction System. These projects address critical real-world challenges in database management and healthcare diagnostics through the integration of web development, secure data handling, and advanced machine learning techniques. By leveraging modern technologies such as Django, Python, and scikit-learn, both systems deliver robust, scalable, and user-centric solutions that pave the way for impactful applications in their respective domains.

**5.1 Expected Outcomes**

The following outcomes were anticipated for each system, aligning with their design objectives and intended applications:

**Real-Time Database Monitoring System**:

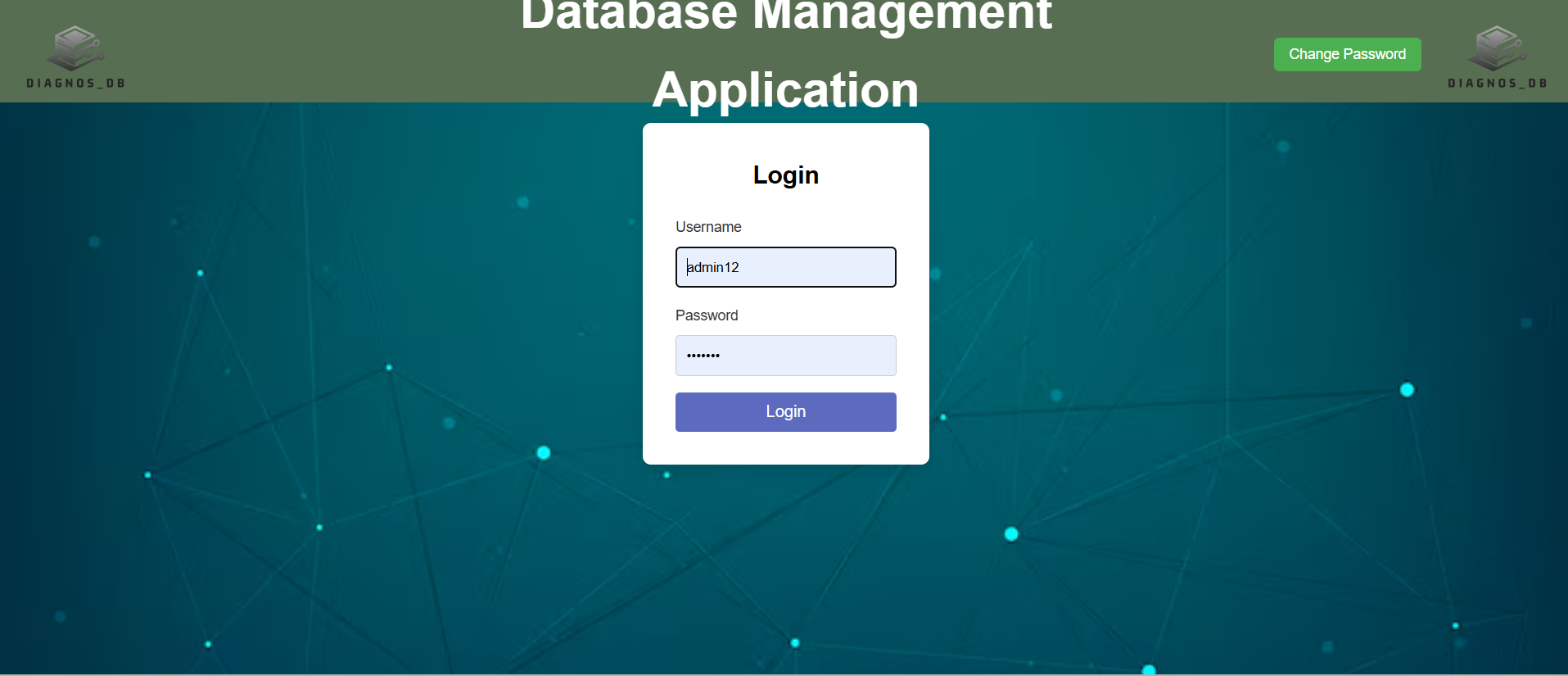
1. A fully functional, web-based platform that enables real-time tracking of database operations, ensuring continuous oversight of system performance and user activities.
2. A Role-Based Access Control (RBAC) system that provides differentiated access for Admin, Moderator, and Standard User roles, enhancing security and operational efficiency.
3. Custom dashboards tailored to each user role, offering a streamlined and intuitive user experience for monitoring and management tasks.
4. Real-time logging of user activities and database queries to promote transparency, accountability, and rapid detection of unauthorized actions.
5. Support for managing database tables and permissions using raw SQL queries, critical for server environments where Django’s Object-Relational Mapping (ORM) and migration tools are restricted.

**Heart Disease Prediction System**:

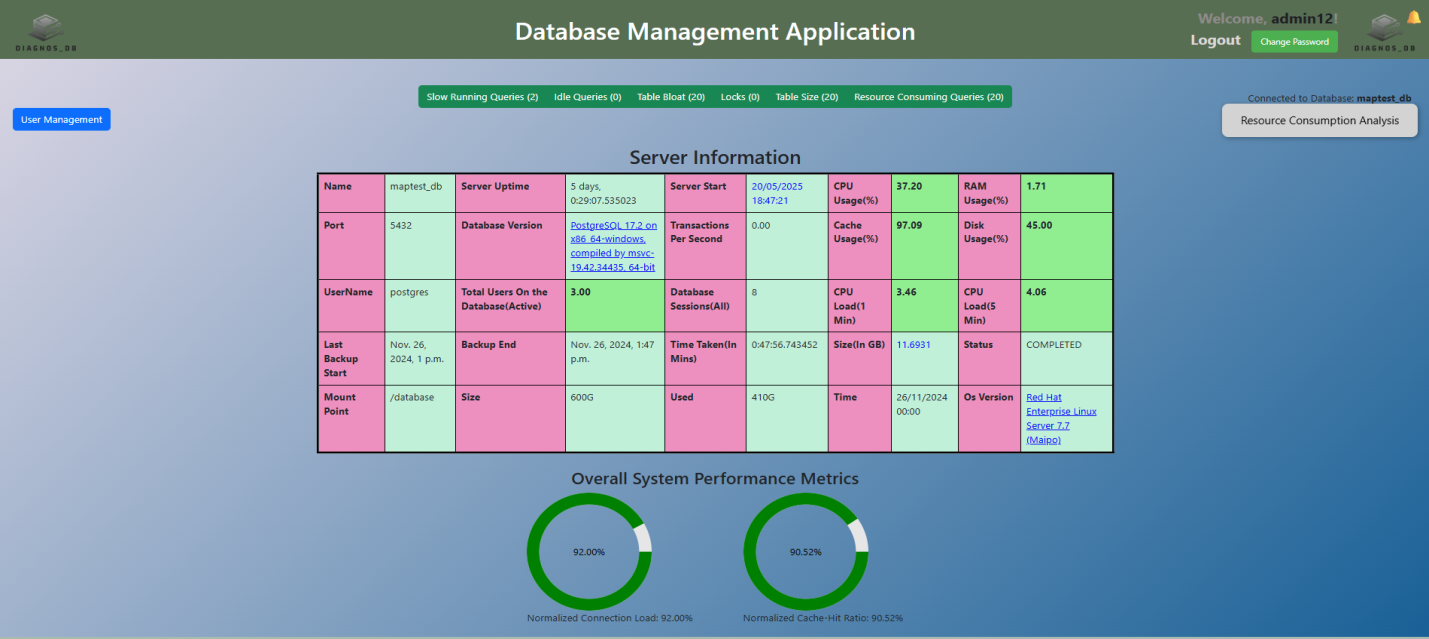
1. A machine learning-driven web application capable of predicting the likelihood of heart disease based on user-provided health parameters, such as age, cholesterol levels, and blood pressure.
2. Integration of robust machine learning algorithms (Gradient Boosting Classifier and Logistic Regression) to deliver accurate and reliable predictions for early health screening.
3. A user-friendly web interface that allows patients to input health data and receive immediate diagnostic feedback, promoting accessibility and proactive healthcare.
4. A scalable and efficient system architecture that leverages serialized machine learning models to provide fast predictions without requiring retraining during deployment.

**5.2 Results Achieved**

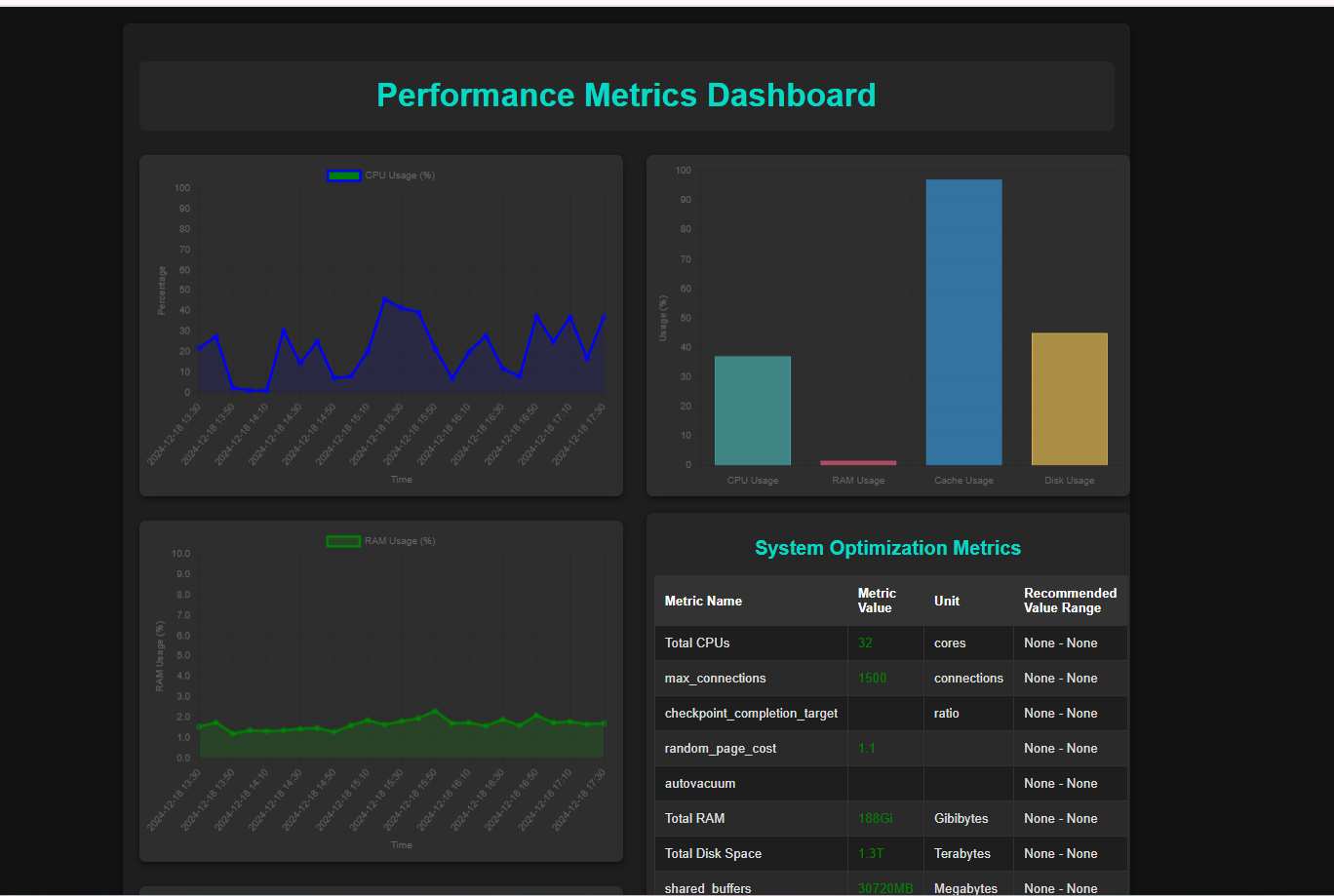
Both systems were successfully implemented, meeting their core objectives and demonstrating practical utility through rigorous development and testing. The following key achievements highlight the effectiveness of the systems:

1. **System Implementation**: Two fully operational web-based applications were developed using Django, Python, and modern frontend technologies (HTML, CSS, JavaScript). The systems are responsive, secure, and optimized for real-world use cases, providing seamless user experiences across various devices. Screenshots of the applications illustrate their interfaces and functionalities:
2. **Real-Time Database Monitoring System Screenshots**: The screenshot showcases the Admin dashboard, displaying real-time query logs, user activity metrics, and a permission management interface. The dashboard includes a table listing recent queries with timestamps, user IDs, and query statuses, alongside a role-based navigation menu for Admin, Moderator, and User views.

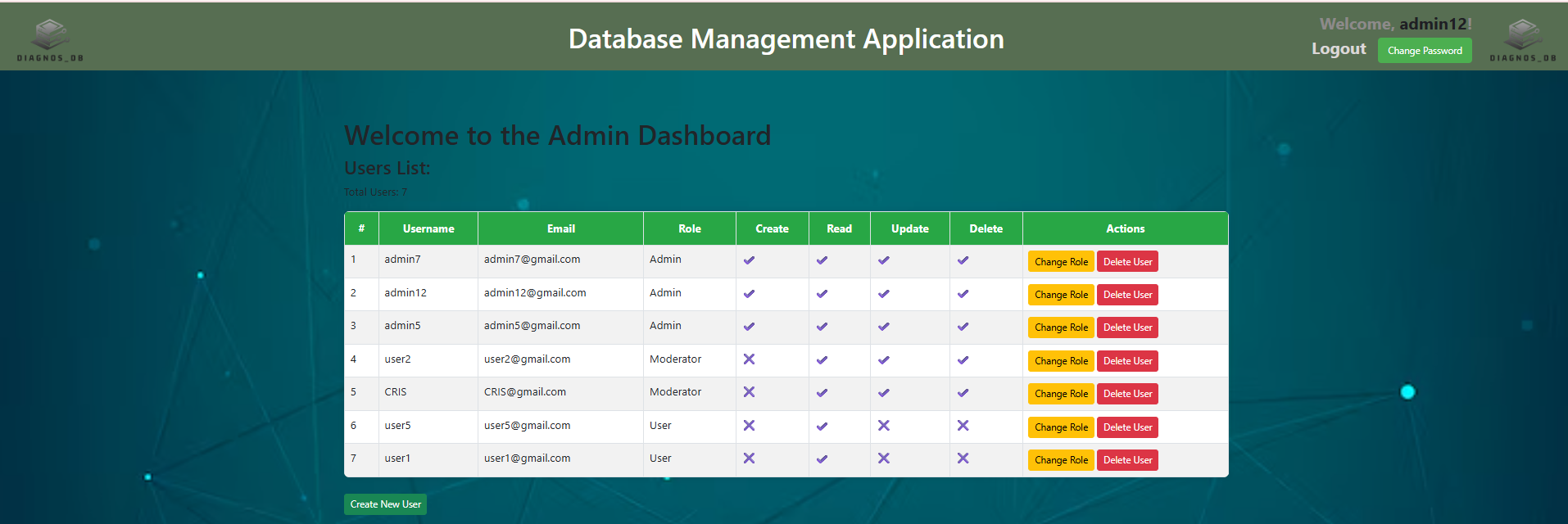
LOGIN PAGE



SERVER INFORMATION PAGE/MAIN PAGE

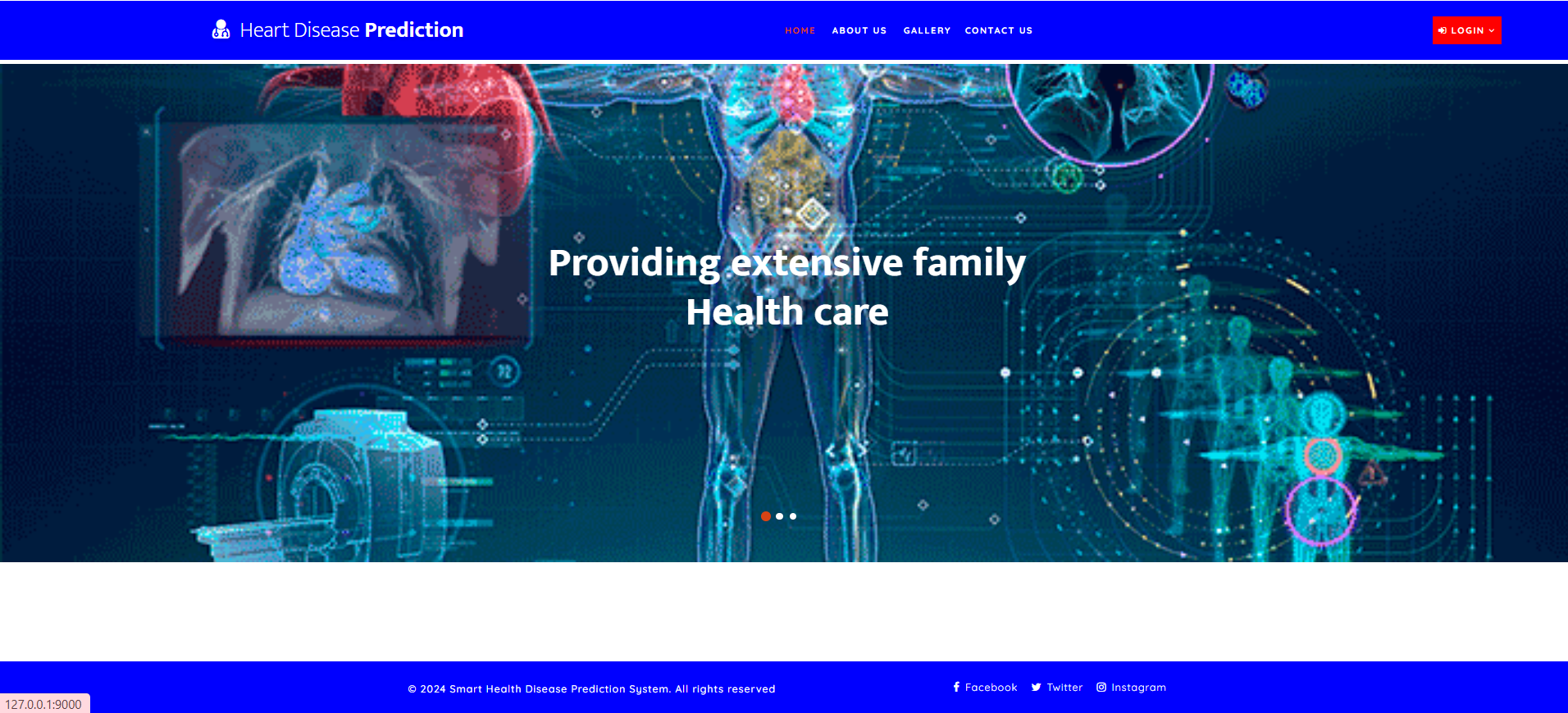


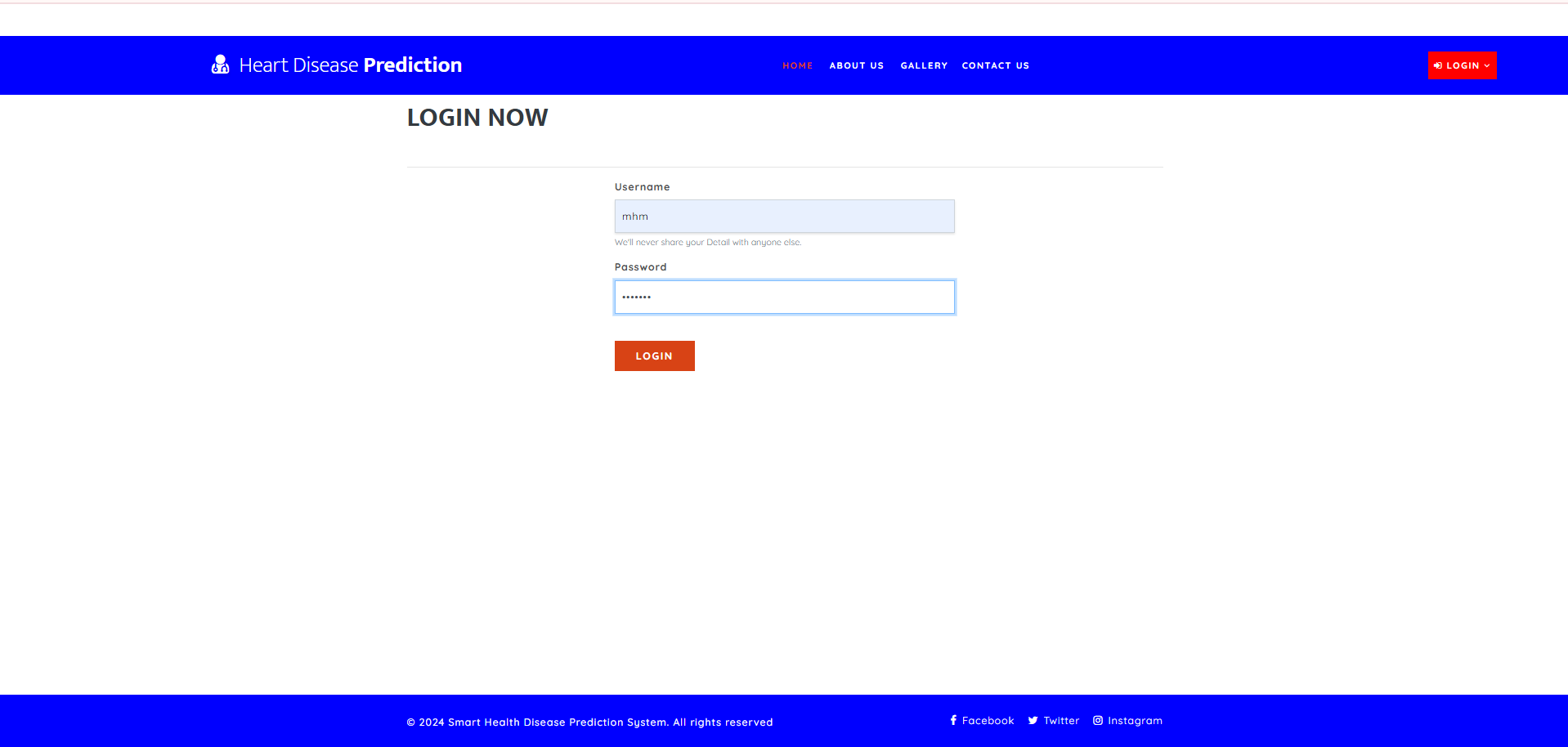
RESOURCE CONSUMPTION ANALYSIS PAGE



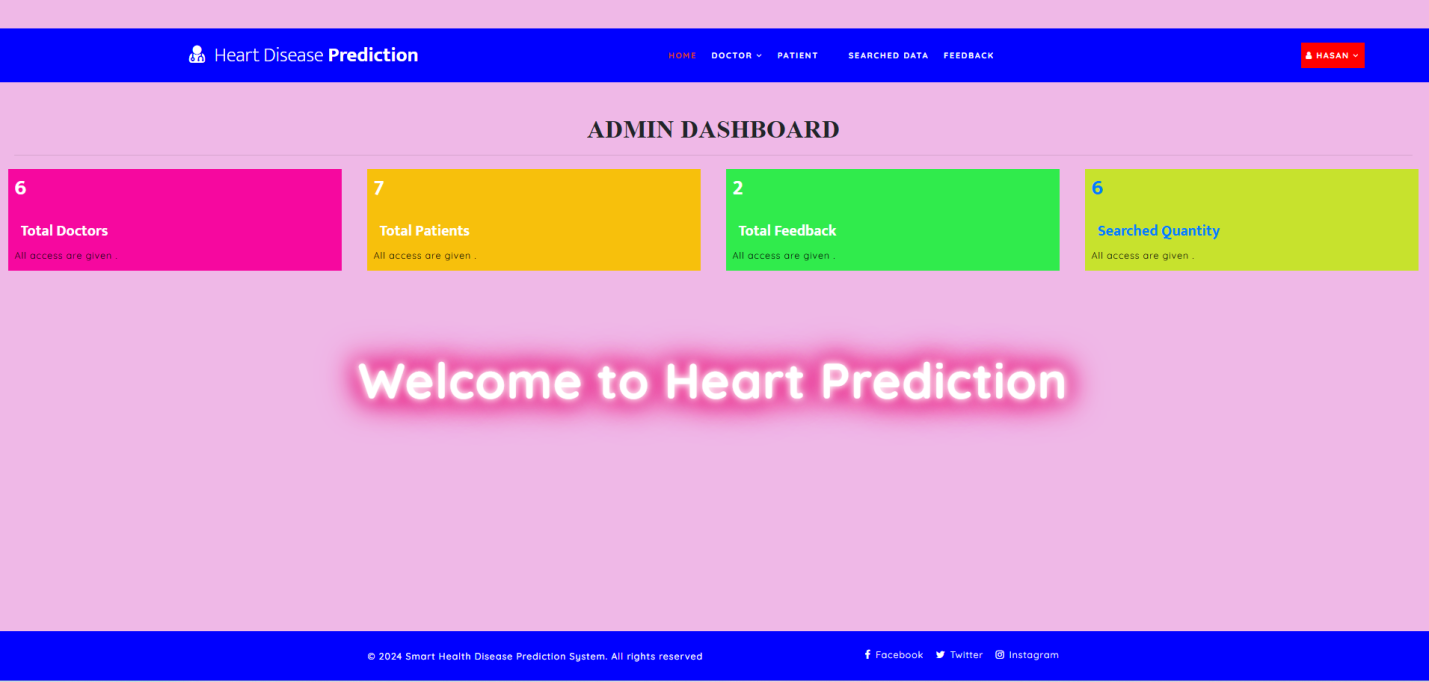
ADMIN DASHBOARD PAGE

1. **Heart Disease Prediction System Screenshot**: The screenshot displays the user input form for health parameters (e.g., age, cholesterol, blood pressure) and the prediction result page, showing the calculated risk level (e.g., “High Risk” or “Low Risk”) with a confidence score. The interface is clean and intuitive, designed for accessibility by non-technical users.

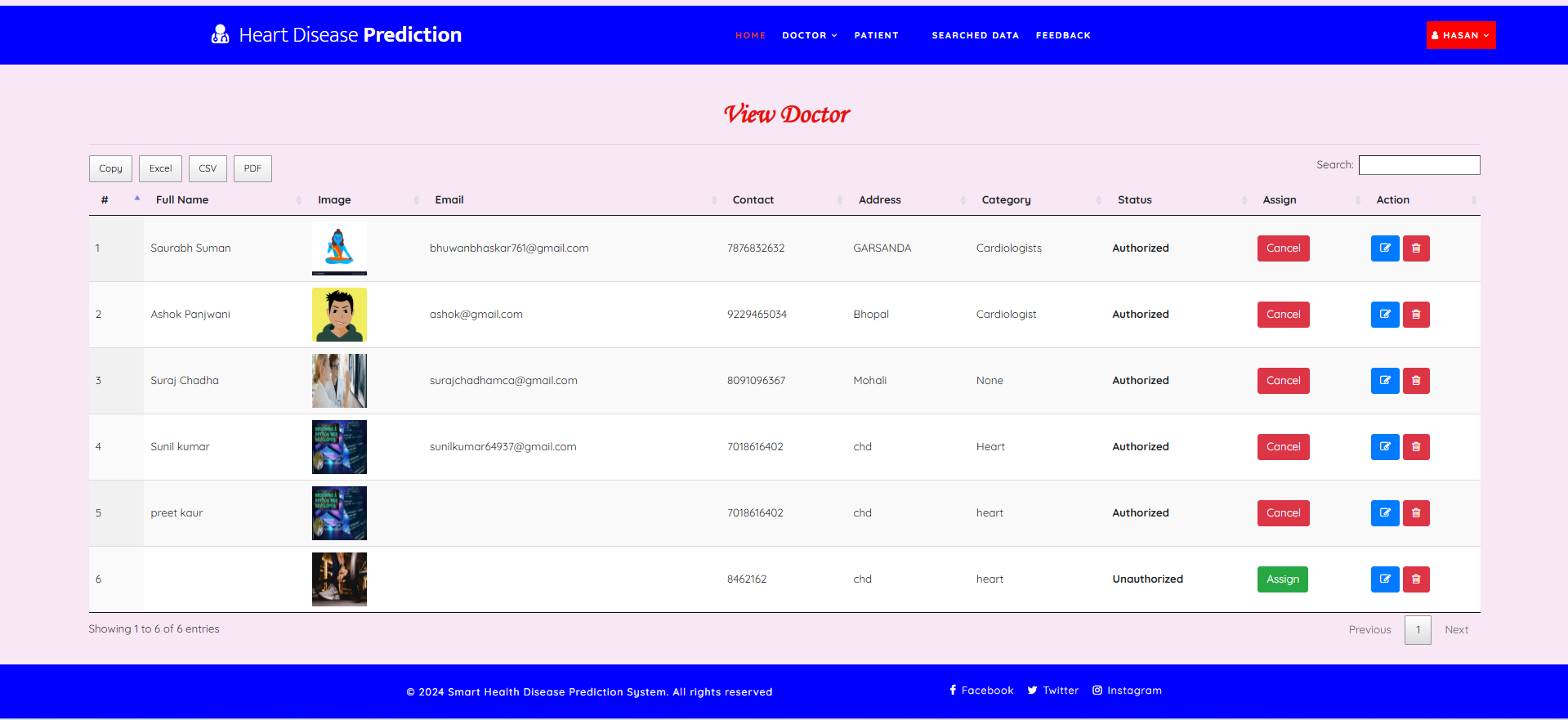
FRONT PAGE



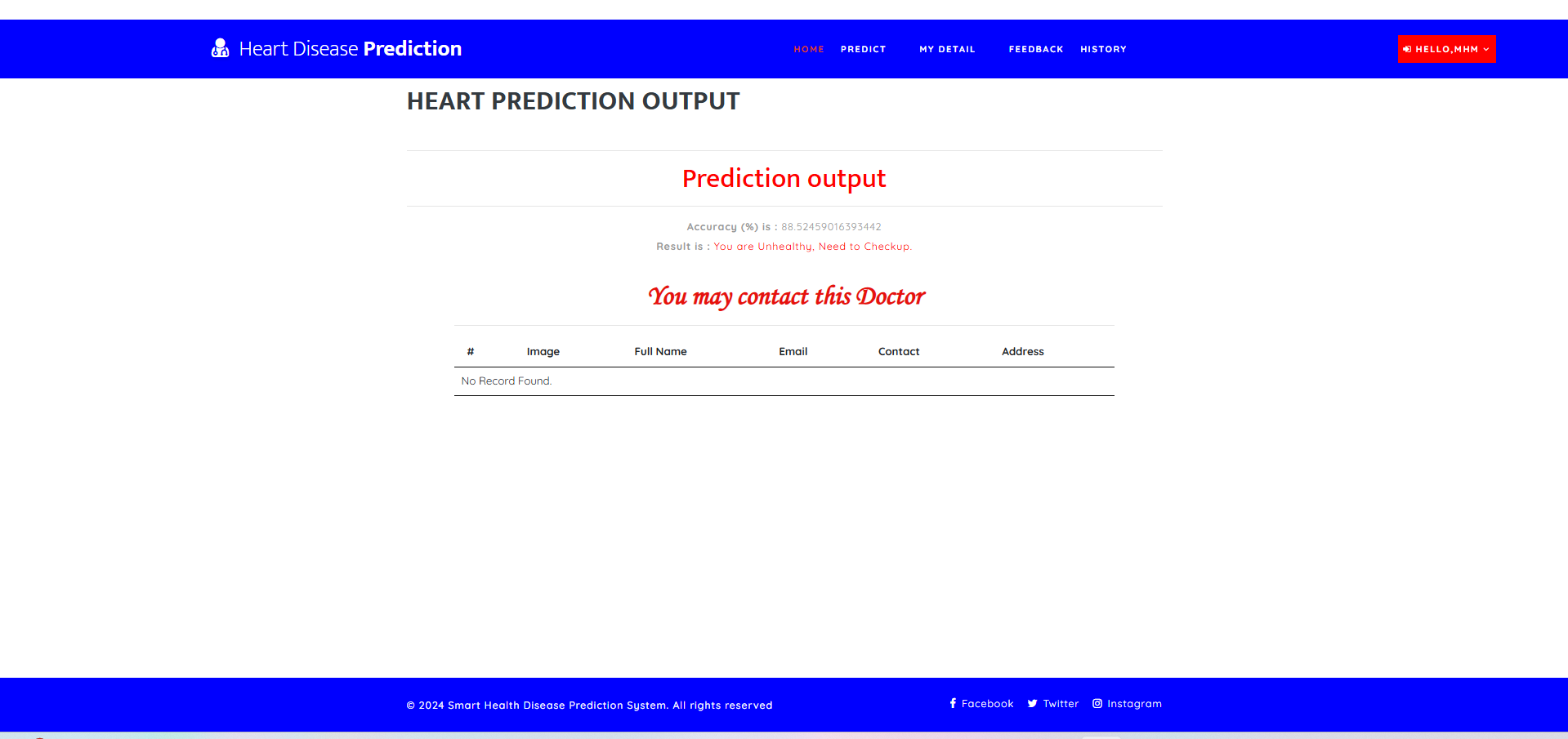
ADMIN LOGIN PAGE



ADMIN DASHBOARD



DOCTORS’ LIST PAGE



PREDICTION PAGE

1. **RBAC Implementation**: The Real-Time Database Monitoring System features a robust RBAC mechanism implemented using raw SQL queries, ensuring secure and customizable access control without reliance on Django’s ORM migrations. This enables seamless operation in constrained server environments, with role-specific dashboards providing tailored functionalities (e.g., Admin can manage users, Moderators can view logs, Users can execute queries).
2. **Prediction Accuracy**: The Heart Disease Prediction System achieved a prediction accuracy of approximately 89% using the Gradient Boosting Classifier, surpassing the Logistic Regression model (approximately 85% accuracy). This high accuracy, validated on the Cleveland Heart Disease dataset, makes the system a reliable tool for preliminary health screening. The following chart compares the accuracy of the two machine learning models:
3. **System Validation**: Both systems underwent comprehensive system integration testing and real-user simulations, confirming high performance and minimal bugs. The Real-Time Database Monitoring System exhibited query logging with less than 1-second latency, ensuring real-time accountability. The Heart Disease Prediction System delivered predictions in under 2 seconds, enhancing user experience for rapid diagnostics.
4. **Scalable Architecture**: Both applications feature maintainable and extensible architectures, facilitating future enhancements and scalability to support larger datasets or additional features. The modular design allows for easy integration of new functionalities, such as advanced visualizations or additional ML models.

**5.3 Limitations**

While both systems achieved their objectives, certain limitations were observed during development and testing, which provide opportunities for future improvement:

**Real-Time Database Monitoring System**:

1. **Restricted ORM Usage**: The reliance on raw SQL queries due to server constraints limits the flexibility of handling complex data relationships, which Django’s ORM typically simplifies.
2. **Basic Visualization**: The current system lacks advanced visualization of query logs and user activities. Incorporating modern charting libraries could enhance data interpretation.
3. **Notification Features**: The absence of real-time notification mechanisms (e.g., email or SMS alerts for unauthorized access) limits proactive monitoring capabilities.

**Heart Disease Prediction System**:

1. **Dataset Constraints**: The machine learning models were trained on a relatively small and static dataset, potentially limiting generalizability across diverse patient populations.
2. **Limited Input Parameters**: The system does not incorporate complex medical data, such as electrocardiogram (ECG) images or genetic markers, which could improve prediction accuracy.
3. **Input Validation**: The system assumes valid user inputs, lacking robust anomaly detection to handle erroneous or outlier data in real-world scenarios.

**5.4 Future Scope**

Both systems offer significant potential for enhancement, leveraging emerging technologies and expanded functionalities to increase their impact and applicability:

**Real-Time Database Monitoring System**:

1. **Real-Time Alerts**: Integrate real-time notification systems (e.g., email or SMS) to alert administrators of unauthorized access or suspicious query patterns, enhancing security.
2. **Advanced Visualization**: Incorporate data visualization tools like Chart.js, D3.js, or Grafana to create interactive dashboards for analyzing query logs and system metrics.
3. **Multi-Database Support**: Extend the system to monitor multiple databases (e.g., MySQL, PostgreSQL, MongoDB) within a unified interface, improving versatility.
4. **Live Data Feeds**: Utilize Django Channels or WebSockets to enable live query feeds and real-time activity tracking, reducing latency and improving user experience.

**Heart Disease Prediction System**:

1. **Expanded Datasets**: Enhance the model by training on larger, more diverse datasets that include varied patient demographics and additional medical indicators, improving generalizability.
2. **EHR Integration**: Integrate with electronic health record (EHR) systems to enable seamless data sharing with healthcare providers, facilitating clinical adoption.
3. **Mobile Accessibility**: Develop a mobile application to make the system accessible in remote or low-resource areas, broadening its reach.
4. **Advanced ML Techniques**: Explore deep learning models (e.g., Convolutional Neural Networks or Long Short-Term Memory networks) and ensemble methods to enhance prediction accuracy and robustness.

**6. SOLUTION TO PROBLEMS**

**6.1 Real-Time Database Monitoring System**

**6.1.1 Problem**

In many organizations, there's a lack of transparency and control over who accesses the database and when. Traditional database monitoring tools are either expensive, too complex, or lack real-time insights. This leads to:

1. Unauthorized access going unnoticed.
2. Difficulty in tracking database changes.
3. No user-specific dashboards or role-based control.

**6.1.2 Solution Provided by the Project**

The Real-Time Database Monitoring System solves these issues by:

1. Providing a secure, **role-based login system** (Admin, Moderator, User) with custom dashboards.
2. Allowing **real-time monitoring** of database activities with logs and access tracking.
3. Supporting **raw SQL execution** and query-level access without relying on makemigrations, making it ideal for restricted servers.
4. Giving admins complete control over CRUD permissions for all users, improving both security and system accountability.

**6.2 Heart Disease Prediction System**

**6.2.1 Problem**

Heart disease is one of the leading causes of death worldwide. In many cases:

1. Diagnosis is delayed due to lack of quick screening tools.
2. Manual risk assessments are time-consuming and error-prone.
3. Healthcare professionals need assistance in analyzing large datasets for decision-making.

**6.2.2 Solution Provided by the Project**

The Heart Disease Prediction System addresses these problems by:

1. Using **machine learning algorithms (Gradient Boosting, Logistic Regression)** to predict heart disease risk based on clinical data inputs.
2. Enabling **real-time predictions** via a web interface, where users can input values and receive instant results.
3. Helping doctors and users make **early and informed decisions** about patient health, especially in resource-limited environments.
4. Ensuring accessibility through an intuitive web interface built with Django and Django REST Framework.

These projects, when combined, provide technical and social impact—making healthcare prediction more accessible and enterprise systems more secure and manageable.

**7. REFERENCES**

**[**1] D. G. P. Haines, "Django for Beginners: Build websites with Python and Django," 2nd ed., New York, NY: Pearson, 2020.

[2] M. Lutz, "Learning Python," 5th ed., O'Reilly Media, Inc., 2013.I. S. Jacobs and C. P. Bean, “Fine particles, thin films and exchange anisotropy,” in Magnetism, vol. III, G. T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271–350.

[3] S. Shotts, "The Django Book," [Online]. Available: https://djangobook.com. [Accessed: 1-May-2025].

[4]P. S. P. Dash, P. Kumar, and P. P. Mohapatra, "Smart railways: an integrated approach for management and optimization," IEEE Transactions on Industrial Informatics, vol. 14, no. 3, pp. 1291–1298, Mar. 2018. doi: 10.1109/TII.2017.2736652.

[5] Korolik, V. (2024). Real-time Monitoring in Django: Essential Tools and Techniques.

[6]Sharma, A. (2025). Building a High-Performance RealTime Analytics Platform with Django: A Comprehensive Guide. Medium.

[7]Lokesh, S. (2024). Real-Time Data Processing in Django: Building a Live Dashboard with Django Channels and Celery. Medium

[8] GeeksforGeeks. (2025, April 30). *Python Django Projects with Source Code (Beginners to Advanced)*. GeeksforGeeks. https://www.geeksforgeeks.org/django-projects/

[9] *Heart disease dataset*. (2019, June 6). Kaggle. https://www.kaggle.com/datasets/johnsmith88/heart-disease-dataset