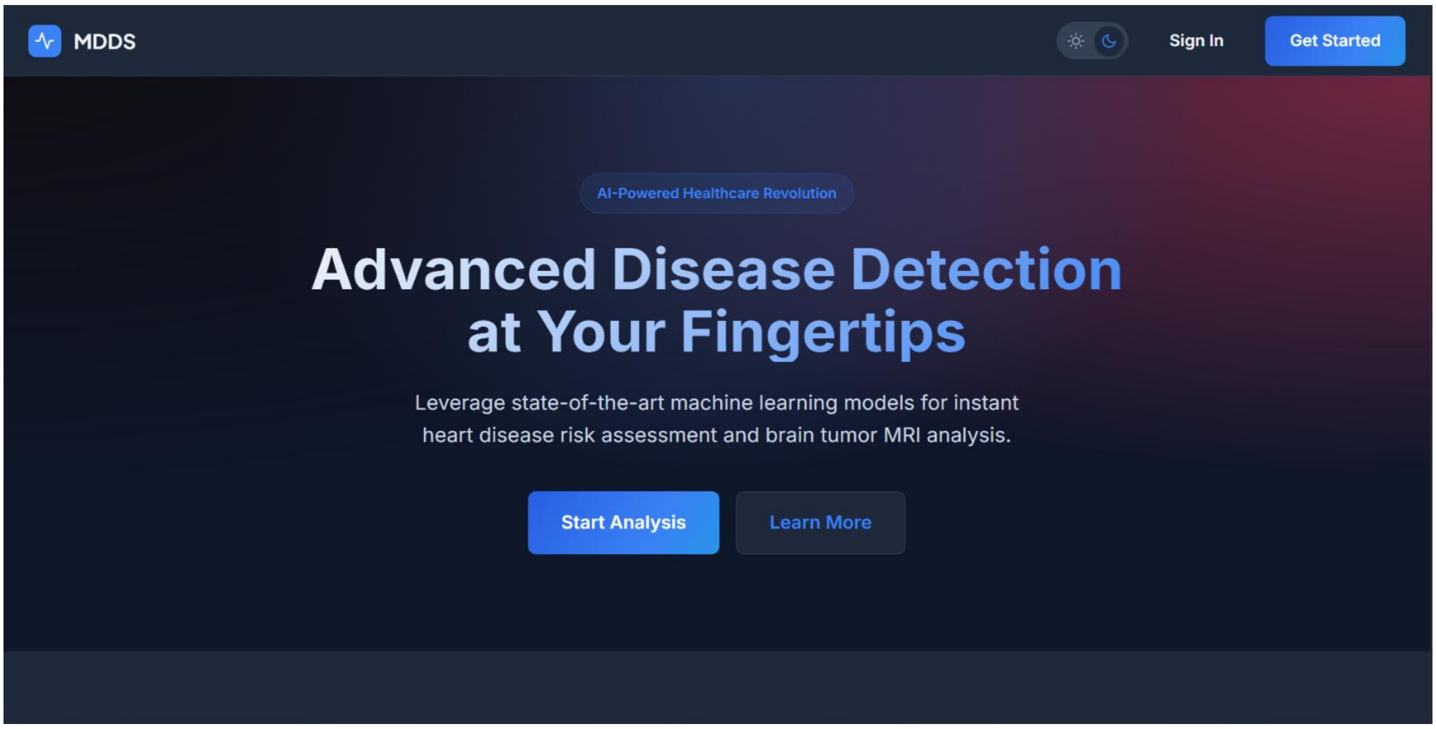
PROJECT REPORT

MULTI DISEASE DETECTION SYSTEM

Project Name	**Multi Disease Detection System**
Project Team	**Mahmoud Ahmed - 192300112** **Zyad Amr - 192300143** **Mohamed Khaled - 192300189** **Seif Ahmed - 192300159** **Farouk Mahmoud - 192300216**
Supervised by	Dr. Cherry Ali / Eng. Hussien Mohammed / Eng. Mohammed Hany



Advanced Disease Detection at Your Fingertips

Leverage state-of-the-art machine learning models for instant heart disease risk assessment and brain tumor MRI analysis.

Project Overview

The **Multi Disease Detection System** is an integrated healthcare platform designed to assist users in identifying potential health risks through advanced **Machine Learning (ML)** and **Deep Learning (DL)** technologies. The system provides automated screening for two critical health conditions: **Heart Disease** and **Brain Tumors**. Additionally, it features an **AI-powered Medical Chatbot** that offers symptom analysis and health guidance based on the user's specific screening history.

The primary goal is to provide an accessible, Medical tool that bridges the gap between raw medical data and patient understanding, emphasizing that while these tools provide high confidence estimates, they serve as aids rather than final clinical diagnoses.

System Architecture

The project follows a modular, service oriented architecture built on the **Flask web framework**. It utilizes a clear separation of concerns to ensure maintainability:

- **Core Managers:** Centralized handling of hardware/resource intensive tasks like database connections (database_manager.py) and ML model loading (model_manager.py).
- **Services Layer:** Contains the logic for predictions, authentication, User Settings, Report and chatbot interactions.
- **Models Layer:** Defines the data structures (User) and the wrappers for ML models (HeartDiseaseModel, BrainTumorModel).
- **UI/Presentation:** A clean frontend using HTML, CSS, and JavaScript, separated into templates and static assets.

Technical Stack

Backend & Web Framework:

- **Python:** The primary programming language.
- **Flask:** Used for routing, session management, and serving the web application.
- **SQLite:** A lightweight relational database for storing user profiles and prediction history.

Machine Learning & Deep Learning:

- **TensorFlow / Keras**: Utilized for building and training the **Convolutional Neural Network** (CNN) for brain tumor detection.
- **Scikit-learn:** Used to develop the **Random Forest** Classifier for heart disease prediction.

AI & LLM:

- **Groq Cloud API:** Powers the chatbot using the **Llama-3.1-8b-instant** model.

Implementation Details

1. Heart Disease Prediction

- **Dataset:** Trained on the “70k Cardiovascular Dataset“.
- **Methodology:** A Random Forest Classifier was chosen for its robustness and ability to handle tabular data.
- **Preprocessing:**
  - Filters out outliers

- **Feature Engineering:** Calculates BMI from height and weight to improve model accuracy.
- Converts age from years to days to match the original dataset format.

Output: Provides a risk level (Low, Medium, High) and a confidence probability.

2. Brain Tumor Detection

- **Technology:** A Convolutional Neural Network (CNN).
- **Architecture:**
  - Four Convolutional layers for feature extraction.
  - MaxPooling layers for spatial reduction.
  - Dropout layers (0.5) to prevent overfitting.
  - Softmax activation for multi-class classification.
- **Classes:** Glioma, Meningioma, Pituitary tumor, and No Tumor.
- **Preprocessing:** Resizes MRI images to 128x128 pixels and normalizes pixel values.

3. AI Medical Chatbot & Symptom Analyzer

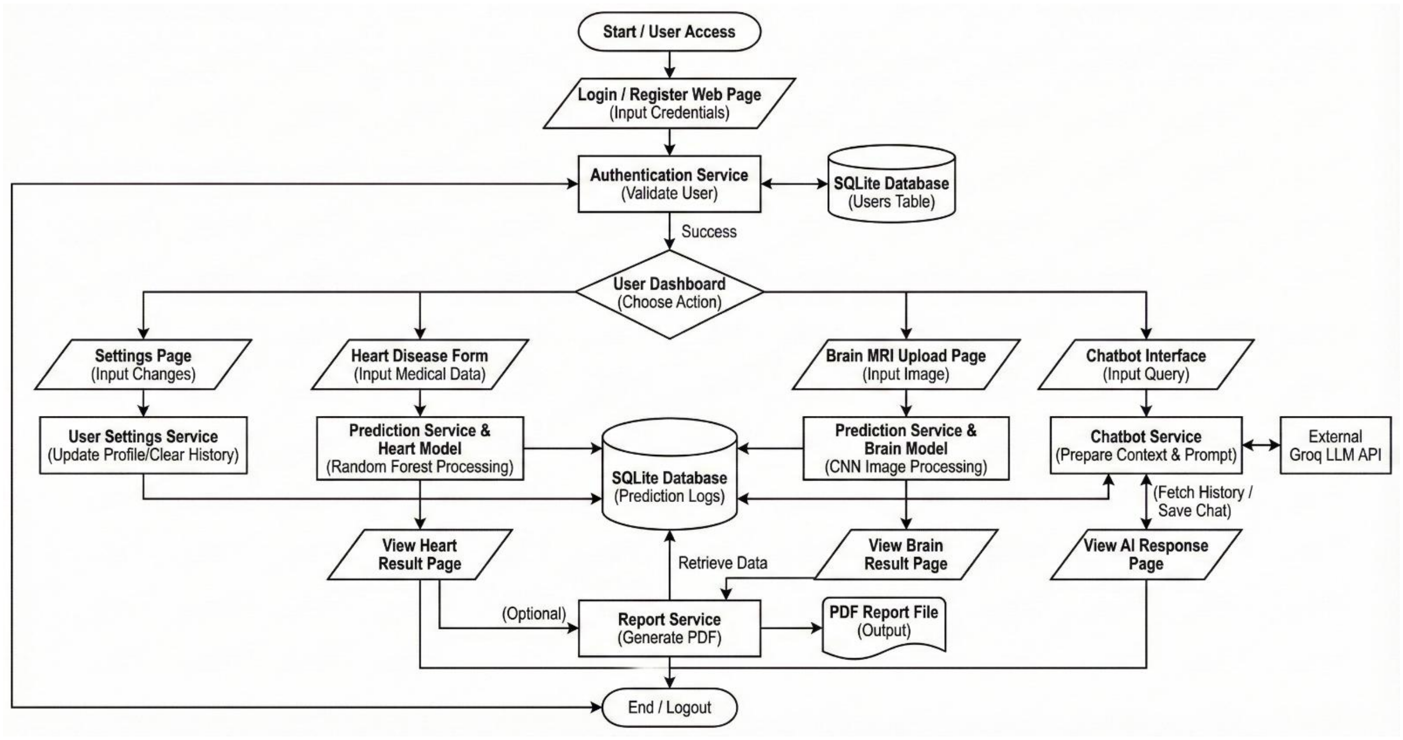
- **Context Awareness:** The chatbot is "memory-capable." It fetches the user's latest heart and brain screening results from the database to provide personalized advice.
- **Safety Filtering:** Hardcoded keywords ensure the bot only answers health-related questions and refuses off-topic queries.

4. Reporting System

- Users can download professional PDF reports for their screenings.
- Reports include user details, the specific inputs provided, the AI's prediction, and structured medical suggestions.

Project Workflow

1. **Authentication:** User registers/logs in through **auth_service**.
2. **Input Submission:** User provides clinical data via a form (Heart) or uploads an MRI image (Brain).
3. **Service Processing:** The **PredictionService** receives the input, triggers the appropriate model via **ModelManager**, and logs the result in **prediction_logs**.
4. **Result Display:** The system displays the prediction, probability, and medical suggestions.
5. **AI Consultation:** The user can ask the Chatbot about their symptoms; the bot reviews their history in **prediction_logs** to provide specific insights.


Start / User Access
Login / Register Web Page (Input Credentials)
Authentication Service (Validate User)
SQLite Database (Users Table)
Success
User Dashboard (Choose Action)
Settings Page (Input Changes)
Heart Disease Form (Input Medical Data)
Brain MRI Upload Page (Input Image)
Chatbot Interface (Input Query)
User Settings Service (Update Profile/Clear History)
Prediction Service & Heart Model (Random Forest Processing)
SQLite Database (Prediction Logs)
Prediction Service & Brain Model (CNN Image Processing)
Chatbot Service (Prepare Context & Prompt)
External Groq LLM API
(Fetch History / Save Chat)
View Heart Result Page
Retrieve Data
View Brain Result Page
View AI Response Page
(Optional)
Report Service (Generate PDF)
PDF Report File (Output)
End / Logout

Object-Oriented Programming (OOP) Concepts

The project extensively uses OOP to ensure the code is modular, reusable, and scalable.

Abstraction:

- The **BaseDiseaseModel** abstract base class defines a standard interface (**load_model** and **predict**) that all disease models must implement. This allows the system to interact with any disease model without knowing its internal complexities.
- **BaseService** acts as an abstract foundation for all business logic services, providing a shared database manager attribute.

Inheritance:

- **HeartDiseaseModel** and **BrainTumorModel** inherit from **BaseDiseaseModel**, reusing common logic while implementing their specific prediction algorithms.
- **PredictionService**, **AuthService**, **UserSettingsService**, **ReportService** and **ChatbotService** inherit from **BaseService** to gain standard access to database utilities.

Encapsulation:

ModelManager (Data Hiding):

- Mechanism: Restricts direct access to AI model instances to prevent redundant loading and inconsistent states.
- Key Attributes:
  - self._heart_model, self._brain_model (Private/Protected).
- Key Methods:

    - get_heart_model(), get_brain_model(). These public getters manage the “lazy loading“ logic internally.

DatabaseManager (Logic Abstraction):

- Mechanism: Bundles connection handling and SQL execution, hiding low-level sqlite3 complexity from the application layer.
- Key Attributes: **self.db_path**.
- Key Methods: **execute(), fetch_one()**. These methods handle the entire lifecycle of opening connections, committing transactions, and closing resources.

User Model (Behavior & Security Encapsulation):

- Mechanism: Encapsulates cryptographic logic within the object, preventing raw password handling and exposure.
- Key Attributes: **password_hash** (Stores the hash, not the plain text).
- Key Methods: **set_password()**, **check_password()**. External code uses these methods to interact with credentials without understanding the underlying hashing algorithms.

BrainTumorModel (Internal Helpers):

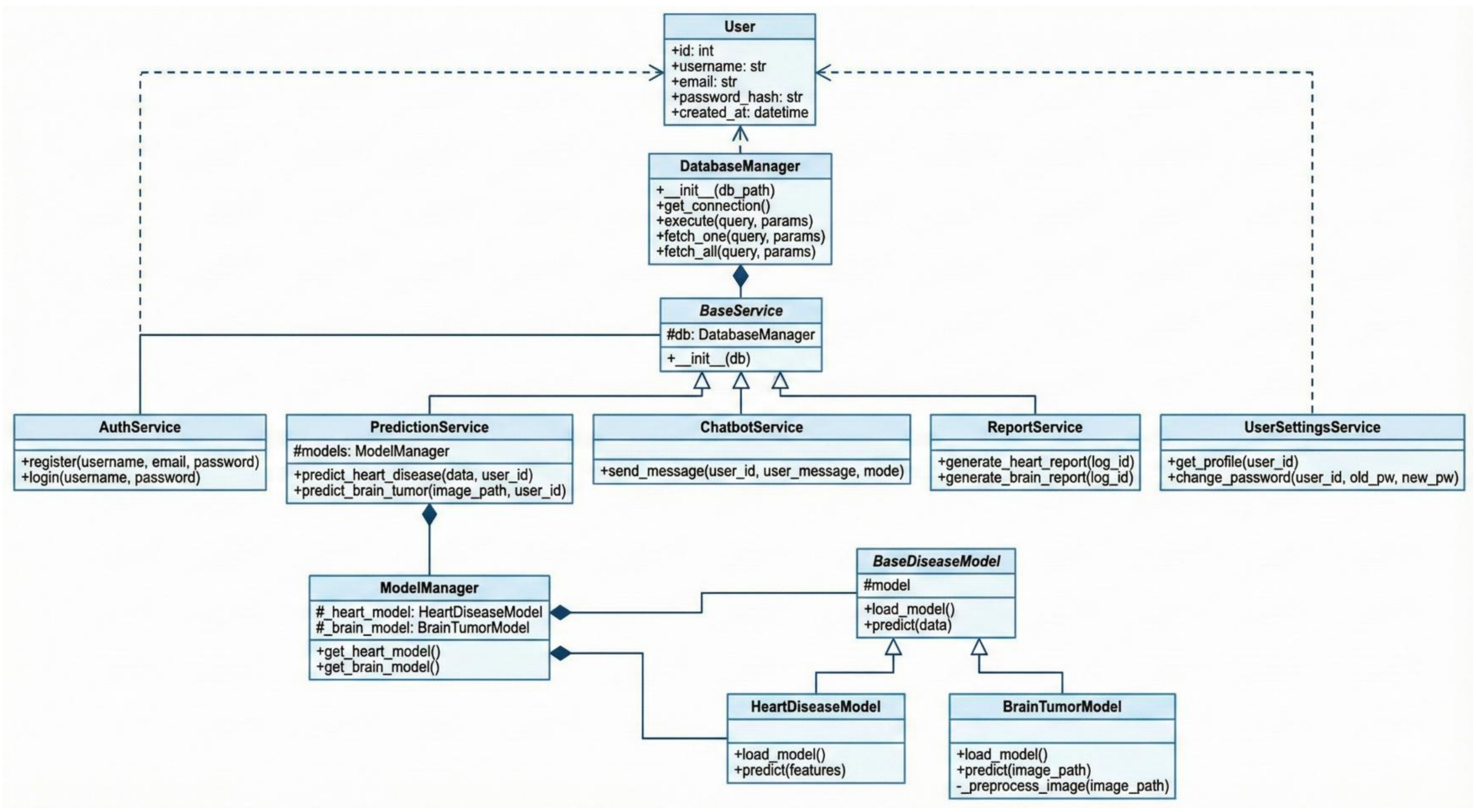
- Mechanism: Hides the complexity of image preprocessing (resizing, normalization) and deep learning framework interactions.
- Key Attributes:
    - **self._model** (The private Keras model instance).
- Key Methods:
    - **_preprocess_image()** (Private helper), **predict()** (Public interface). The class exposes a simple API while handling complex data transformations internally.

PredictionService & AuthService (Service Layer Encapsulation):

- Mechanism: Acts as a facade to orchestrate complex business workflows, isolating the web controllers from backend logic.
- Key Methods:
  - **predict_heart_disease()**: Bundles input validation, feature calculation (BMI), model inference, and database logging.
  - **register():** Encapsulates the complete user creation flow, including duplicate checks and database insertion.

Polymorphism:

- The **ModelManager** can handle different models through their shared **BaseDiseaseModel** interface, allowing for polymorphic behavior when calling prediction methods.



Database Schema

The system uses two primary tables:

1. Users: Stores username, email, and hashed password.
2. Prediction Logs: Stores every screening performed, including:
    - **user_id:** Links the result to a specific user.
    - **model_type:** (heart_disease or brain_tumor_multiclass).
    - **input_summary:** A snapshot of the data used (e.g., BP values or image path).
    - **prediction_result** and **probability:** The Models output.

Conclusion

The Multi Disease Detection System demonstrates a robust implementation of modern software engineering and machine learning. By leveraging OOP principles like abstraction and inheritance, the system is designed to easily accommodate additional disease models in the future, providing a scalable solution for automated medical screening.