Detail Design:

# Architecture:

For my project skin disease detection using picture combining deep learning and machine learning for skin lesion analysis, a modular and layered architecture is ideal.

### High-Level Architecture

1. **User Interface Layer:**

**Purpose**: enable user interactions with the system such as uploading images, viewing results and exporting reports.

**Components:**

Android app: Built with Java using Android Studio.

**Web Interface:** users upload pre-captured images using a browser.

1. **Communication Layer:**

Both platforms communicate with a **Backend Server** using **RESTful APIs**. This ensures consistency and centralizes the image processing and disease detection logic.

1. **Backend Layer:**

The backend handles the core functionality of the system, including preprocessing, disease detection, and result generation:

**Image Preprocessing**: Uses OpenCV for resizing, normalization, and segmentation.

**Model Inference**: Combines predictions from:

**Deep Learning Models** (CNN with PyTorch).

**Machine Learning Models** (Random Forest, SVM with Scikit-learn).

And it may use ensemble techniques to aggregate predictions.

**Result Generation**: Generates confidence scores, disease names, and visualizations (e.g., heatmaps).

1. **Data Layer MongoDB stores:**

The data layer handles data storage and retrieval:

* User uploaded images
* Preprocesses images.
* Model analysis results.
* Patient record and reports.

1. **Reporting Layer:**

This layer provides the following functionalities:

**Mobile app:** Displays results such as disease type, confidence score and allows report downloads.

**Web Interface:** Allows report download and visualization on larger screens

# A diagram of a diagram Description automatically generated with medium confidenceGraphic Description:

**Frontend:**

* **Sign In/Up**: Allows users to create an account or sign in to access personalized features. This process connects to the **Communication Layer** to handle user credentials and account creation.
* **Guest Sign In**: Provides a way for users to interact without creating an account, likely with limited functionality.
* **Request Previous Result**: Users can query previously processed results, sent through the Communication Layer to fetch saved analysis results.
* **Upload Image**: Enables users to upload an image for analysis. This action passes the image to the Communication Layer for preprocessing and analysis.

**Communication Layer:**

* **Sign In/Up and Guest Sign In**: Authenticates the user or creates a new account, passing the request to the Backend Layer for validation and account handling.
* **Upload Image**: Handles the upload request by passing the image data to the backend's preprocessing module for further analysis.
* **Request Previous Result**: Fetches user-specific analysis results from the database by interacting with the Backend Layer.

**Backend:**

* **Preprocess Image**: Prepares the uploaded image for machine learning analysis by resizing, normalizing, or extracting key features.
* **ML Server (Machine Learning Model)**: The ML server processes the preprocesses image using models like CNN (ResNet50 or InceptionV3). The analysed data is sent back to the backend.
* **Analyze Image**: Coordinates with the ML server to process the image, interprets results, and formats them for storage or display.
* **Retrieve User Result**: Fetches past analysis data from the database based on user requests. Useful for displaying history or trends.
* **Analysis Results**: Manages and organizes results before they are saved in the database or displayed to the user.
* **Check User:** Verifies if a user exists in the database and fetches their data or creates a new user account if not exists.

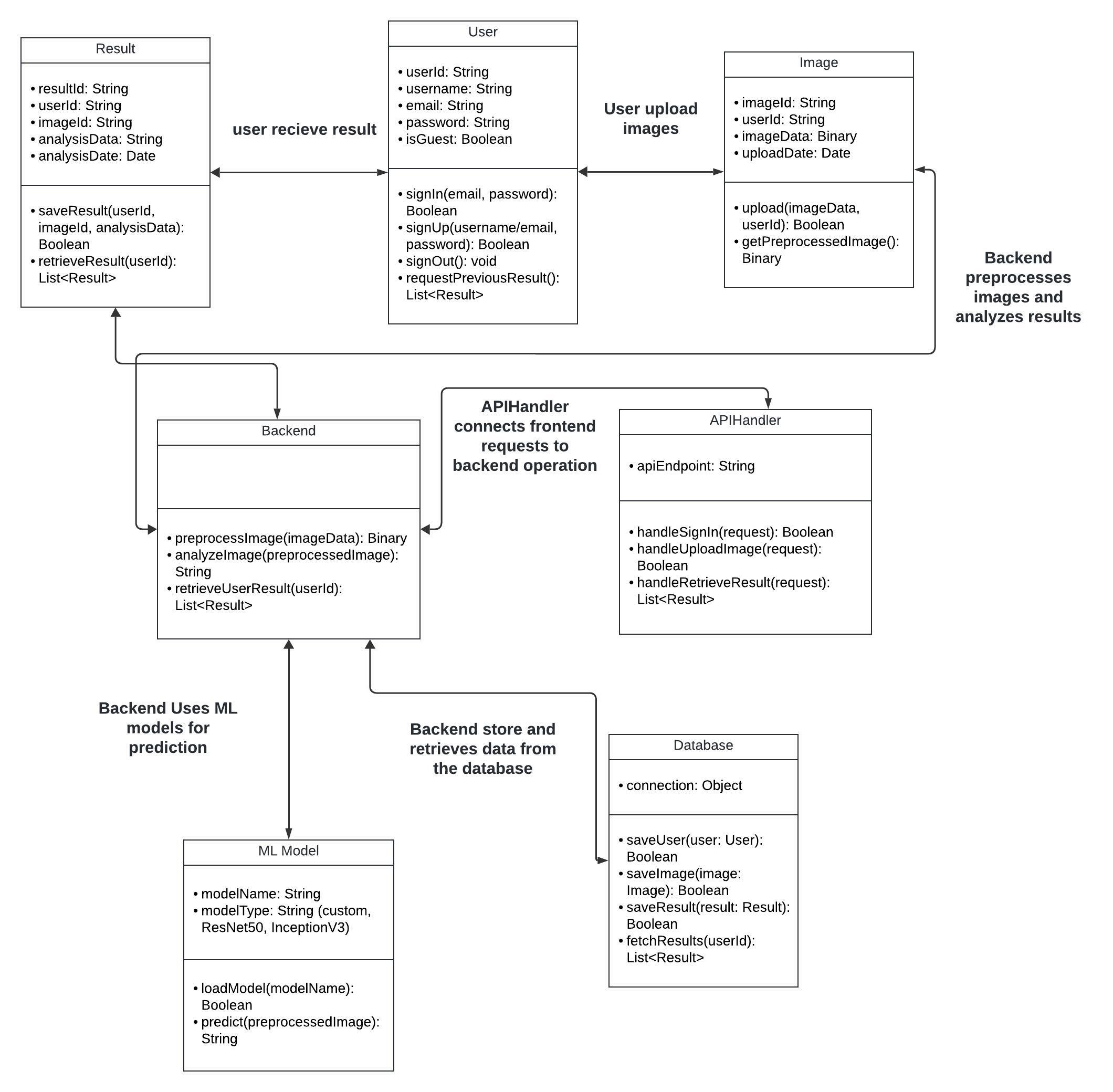
**Data Layer:**

* **Stores User Data**: Manages storage for user credentials, uploaded images, and processed analysis results. Interacts with the backend to save and retrieve information.

**Reporting Layer:**

* **Display Result**: Presents processed analysis results to the user in a clear and interpretable format. This step concludes the user's workflow.

## Class Diagram:



**Classes**

**User**: Represent a user in the system, responsible for managing user accounts, authentication, and user-specific data.

**Image**: represent an image uploaded by a user, containing both the raw image data and associated metadata such as the user who uploaded it and the upload date.

**Result**: represents the outcome of an image analysis process. It stores the analysis data generated after an image is processed by a machine learning model in the database through the backend.

**Backend**: The Backend class acts as a processing engine that handles the various tasks related to image manipulation and analysis. Its main responsibility is to preprocess images, making them suitable for analysis, and to invoke machine learning models to analyze the preprocessed images.

**MLModel**: the MLModel class encapsulates the machine learning model that performs the analysis on images. Its purpose is to abstract the model’s functionality and provide an interface for interacting with different types of machine learning models, such as deep learning architectures like ResNet50 or InceptionV3.

**Database**: The Database class is responsible for all interactions with the database, ensuring that data is correctly stored, retrieved, and managed.

It provides a centralized mechanism for saving users, images, and analysis results to the database, ensuring data consistency and integrity across the system.

**APIHandler**:

The APIHandler class acts as an intermediary between the system's frontend and backend, handling HTTP requests from external sources such as client applications.

## Algorithm description

**Machine Learning Algorithms**

**Random Forest:**

Random Forest is a robust ensemble algorithm that combines the predictions of multiple decision trees to improve accuracy and reduce overfitting. In your project, you can use Random Forest to preprocess data by identifying important features or for metadata classification. For example, if your project includes structured data about images (e.g., user-uploaded metadata such as image category, resolution, or patient history), Random Forest can classify this data to assist in preprocessing or complement the image analysis pipeline. Its ability to handle noise and imbalanced datasets is a significant advantage.

**SVM:**

SVM is a reliable algorithm for classification and regression, especially when dealing with smaller datasets. It finds the optimal hyperplane to separate data into distinct classes. In your project, SVM can be employed to analyze specific features extracted from images, such as identifying basic patterns or categories (e.g., whether an image likely contains abnormalities). It can also serve as a secondary classifier to refine results when integrated with deep learning outputs from CNNs. The simplicity of SVM makes it computationally efficient for less complex tasks.

**Deep Learning Algorithm**

**CNN:**

CNNs are highly effective for deep learning tasks involving image data. They automatically learn and extract features such as edges, textures, and complex patterns to classify or analyze images with high accuracy. In your project, CNNs can be leveraged to process user-uploaded images to detect specific conditions, lesions, or abnormalities. For instance, CNNs could identify diseases based on patterns in medical images and generate heatmaps for visualization. Their ability to handle unstructured data makes them the backbone of your image analysis system.