Detail Design:

# Architecture:

### For my project on skin disease detection using images, combining deep learning and machine learning for skin lesion analysis, a client-server architecture is an ideal choice.

### Client-Server 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 Dart using Flutter for a seamless and intuitive user experience.

1. **Communication Layer:**

* Facilitates communication between the frontend and backend using RESTful APIs created with Django.
* Ensures centralized processing and consistent application of the image analysis and disease detection logic.

1. **Backend Layer:**

The backend is the core of the system, performing critical tasks related to processing and analysis:

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

**Deep Learning Model Inference**:

Process the pre-processed images using a pretrained Convolutional Neural Network.

Extracts hierarchical features (edges, textures, shapes) to classify images.

**Heatmap Visualization**

Generates Grad-CAM or similar heatmaps to visualize regions of interest during predictions.

1. **Data Layer:**

The data layer handles secure storage and efficient retrieval of system data using **MongoDB**:

* User uploaded images
* Preprocesses images.
* Model analysis results (e.g., classifications, confidence scores, and heatmaps).
* Patient record and reports.

1. **Reporting Layer:**

Provides functionality for delivering insights and results:

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

# Graphic Description of Client-Server Architecture:

**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 users with limited functionality to interact with the app without creating an account.
* **Request Previous Result**: Enables users to query and view previously processed results. This request is routed through the Communication Layer to fetch data from the backend. (only for Sign up users)
* **Upload Image**: Allows users to upload images for analysis. The image is sent 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 image upload requests by transferring the image data to the backend for preprocessing and analysis.
* **Request Previous Result**: Fetches user-specific analysis results from the database by coordinating with the Backend Layer.

**Backend:**

* **Preprocess Image**: Prepares the uploaded image for machine learning analysis by resizing, normalizing, or extracting key features.
* **DL Server (Deep Learning Model)**: Processes preprocessed images using pretrained CNN models (e.g., ResNet50, InceptionV3). The analyzed results are returned to the backend.
* **Analyze Image**: Coordinates with the DL server to process the image, interprets results, and formats them for storage or display.
* **Retrieve User Result**: Retrieves past analysis data from the database based on user requests, enabling users to view their history or trends.
* **Analysis Results Management**: Organizes and prepares results for saving in the database or presenting 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.

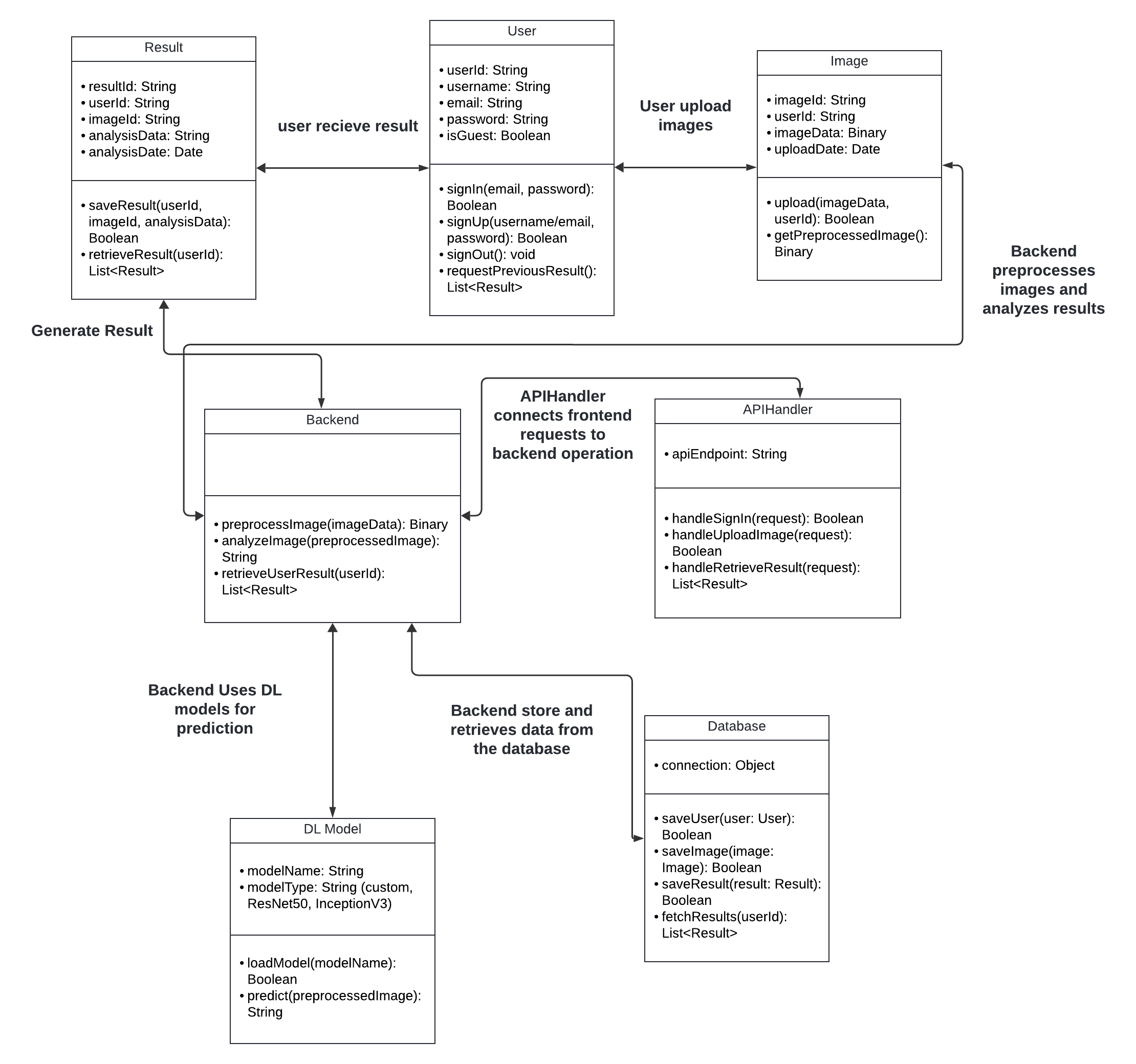
**Data Layer:**

* **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, including disease classifications, confidence scores, and visualizations (e.g., heatmaps), in a clear format.

## 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.

**DLModel**: the DLModel class encapsulates the deep 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 deep 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.

**Arrows:**

**Result<->User**: This indicates that a User class is associated with the Result class, meaning users receive multiple results after processing their uploaded images.

**User<->Image** : This indicates that the User class is associated with the Image class, meaning users interact with images by uploading them.

**Backend<->Image/Result:** The Backend class depends on the Image and Result classes because it processes images and generates results. This relationship shows that changes in the Image or Result classes might affect the Backend class's functionality.

**Backend<-> DL Model:** The Backend class depends on the DLModel class to perform predictions. The DL models are the core processing unit that analyzes data (images) and generates predictions (results).

**Backend<->Database:** The Backend class interacts with the Database class to store and retrieve information, such as uploaded images, user data, and results.

**APIHnadler<->Backend:** The APIHandler class serves as a bridge between frontend requests and the backend. It depends on the Backend class to perform operations like uploading images, retrieving results, or handling user data.

## Algorithm description

**Deep Learning Algorithm**

**Convolutional Neural Networks (CNNs)**

CNNs are a cornerstone of image analysis and have revolutionized tasks involving unstructured data like images. They leverage convolutional layers, pooling layers, and fully connected layers to learn hierarchical patterns and generate accurate predictions.

**Key Features and Advantages**:

* **Feature Extraction**: CNNs automatically identify low-level features (e.g., edges and textures) in early layers and learn complex, high-level patterns (e.g., shapes, lesions) in deeper layers.
* **Spatial Hierarchies**: Convolutional and pooling layers enable CNNs to capture spatial relationships, which are critical for analyzing image structure.
* **Transfer Learning**: Pretrained models like ResNet50 and InceptionV3 can be fine-tuned to adapt to specific tasks, leveraging their robust feature extraction capabilities.

**Application in This Project**:  
In this project, CNNs will be used to analyze user-uploaded images of skin lesions for the following tasks:

* **Disease Detection**: CNNs will identify whether a lesion is benign or malignant and classify it into specific categories, such as melanoma or other skin conditions.
* **Heatmap Generation**: Grad-CAM or similar techniques will generate visual heatmaps, highlighting regions in the image that contributed most to the prediction, aiding interpretability.
* **Improved Diagnostic Accuracy**: By training the model on datasets like the ISIC archive, CNNs can provide reliable predictions, potentially matching or surpassing human diagnostic performance.

**Preprocessing Techniques**:

* **Image Augmentation**: Techniques like rotation, flipping, zooming, and normalization will ensure the model is robust to variations in input images.
* **Resizing**: Images will be resized to match the input dimensions required by the selected CNN architecture (e.g., 224x224 for ResNet50).
* **Normalization**: Pixel values will be normalized to a standard range to improve training stability.

By leveraging CNNs' ability to recognize patterns in skin lesions, the system aims to provide fast, accurate, and interpretable results, making it a valuable tool for early detection and diagnosis of skin diseases.

**Steps for the Algorithm:**

**Data Collection and Preparation:**

* Obtain a high-quality labeled dataset of skin lesion images (e.g., ISIC dataset).
* Split the dataset into training, validation, and testing sets.

**Data Preprocessing:**

* Resize images to the required dimensions (e.g., 224x224).
* Normalize pixel values for uniform scaling (0-1).
* Use data augmentation techniques to enrich the dataset and reduce overfitting.

**Model Selection and Architecture:**

* Use a pretrained CNN model (e.g., ResNet50, InceptionV3) for transfer learning.
* Replace the output layer with a custom fully connected layer for binary classification.

Fine-tune the model by freezing earlier layers and training only the final layers for specialization.

**Model Training:**

* Train the model on the prepared dataset using a suitable optimizer and loss function.

Monitor training and validation metrics (accuracy, precision, recall) using a library like **TensorBoard** or **Matplotlib** for visualization.

**Model Evaluation:**

* Evaluate the model on the test set using PyTorch.

Use libraries like **scikit-learn** for metrics such as:

* **Confusion Matrix**: Compare predictions with true labels.
* **Precision, Recall, F1-Score**: Use classification\_report for detailed analysis.

**Heatmap and Visualization:**

* Generate heatmaps for interpretability using **Grad-CAM** in PyTorch

**Deployment and Integration:**

* Host my model using **TorchServe** to handle REST API requests for predictions**.**

Integrate the REST API with Django:

* Use Django to build endpoints that accept image uploads and forward them to TorchServe for predictions.
* Retrieve predictions from TorchServe and return responses to the client.

**Database Integration (MongoDB Atlas):**

* Use **PyMongo** or **Django MongoDB libraries** to interact with MongoDB Atlas.
  + Store user-uploaded images and metadata (user info, predictions, timestamps).

**User Interface and Result Display:**

* Show users predictions with confidence scores and optional heatmaps.
* Provide insights into the detected conditions, such as whether a lesion is benign or malignant.

**Continuous Updates(optional):**

* Periodically collect new labeled data.
* Retrain the CNN model using additional data and update the TorchServe deployment.