Afeka – Academic College of Engineering in Tel-Aviv January 2, 2025
Computer Science Gal Alfandary 206396731, Lina Flat 322690181

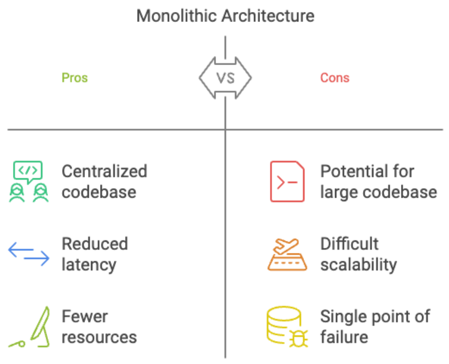
Instructor: Dr. Sharon Yalov-Handzel.

11402: Detailed design -Lost & Pawnd

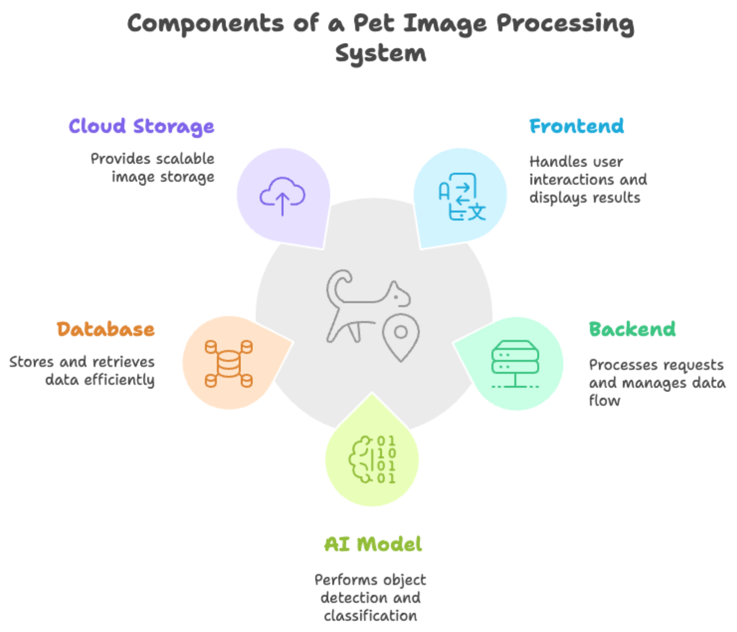
Introduction
Project Overview: "Lost & Pawnd" is a mobile application designed to assist pet owners in locating lost pets or reuniting found pets with their owners. The application leverages AI-powered image recognition technology, supported by a YOLO-based model, to compare uploaded pet images against a database for potential matches. The app offers features for uploading images, managing pet posts, and providing real-time notifications for matches.

Architecture
Architecture Type: Monolithic Architecture
The project follows a **Monolithic Architecture**, where all components, including the frontend, backend, and database, are integrated into a single application. This architecture is ideal for the following reasons:

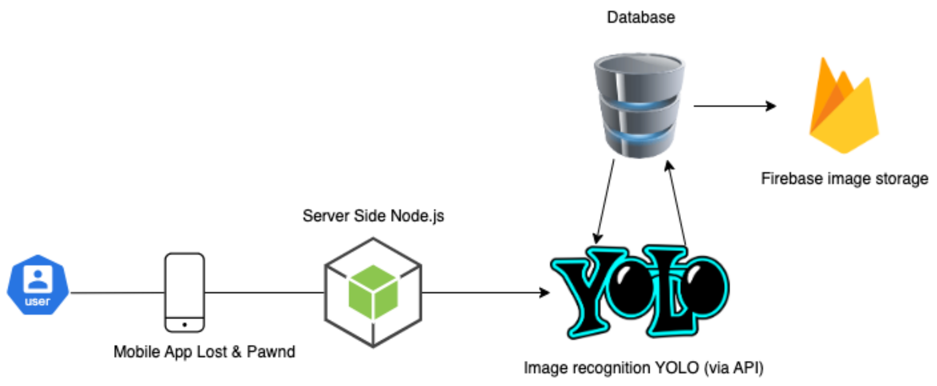
- **Simplicity:** Centralized codebase ensures ease of development and maintenance.
- **Performance:** Direct communication between components reduces latency.
- **Cost-Effectiveness:** Monolithic systems require fewer resources compared to distributed systems.



System Components


Components of a Pet Image Processing System
Cloud Storage
Provides scalable image storage
Frontend
Handles user interactions and displays results
Database
Stores and retrieves data efficiently
Backend
Processes requests and manages data flow
AI Model
Performs object detection and classification

1. **Frontend:**
   - **Technology:** React Native.
   - **Responsibilities:**
     - Handle user input (e.g., uploading images, searching for matches).
     - Display results and notifications.
2. **Backend:**
   - **Technology:** Node.js.
   - **Responsibilities:**
     - Process user requests.
     - Interface with the AI model and database.
     - Manage authentication and data validation.
3. **AI Model:**
   - **Model:** YOLO (You Only Look Once).
   - **Responsibilities:**
     - Perform object detection and classification on pet images.
     - Generate feature vectors for similarity matching.
4. **Database:**
   - **Technology:** PostgreSQL.
   - **Responsibilities:**
     - Store user, pet, and post data.
     - Enable efficient search and retrieval of records.
5. **Cloud Storage:**
   - **Technology:** Firebase.
   - **Responsibilities:**
     - Store pet images for scalability and reliability.



Data Design

- Pet Records: Stored in a **PostgreSQL** database, including fields like pet_id, species, breed, photo_url, and location.
- User Data: Includes user_id, name, email, location, and associated pet posts.
- Image Storage: Pet images are stored in no SQL database - firebase, referenced by URLs in the database.

Tables and Relationships

1. **User**
   - userID: int
   - name: String
   - email: String
   - password: String
   - phoneNumber: String
   - lastLogin: DateTime
   - location: String
   - role: String (e.g., admin, moderator, user)

   Methods:
   + register(): void
   + login(): boolean
   + updateProfile(name: String, location: String): void

2. **Post**
   - postID: int
   - userID: int
   - petID: int
   - dateCreated: Date
   - location: String

Methods:
+ createPost(userID: int, petID: int, ...): void

+ updatePost(postID: int, location: String): void
+ deletePost(postID: int): void
+ searchPosts(location: String, species: String): List<Post>

3. Pet

- petID: int
- species: String
- breed: String
- color: String
- description: String
- photo: Image
- status: String
- lastSeenLocation: String
- status: enum (e.g., Lost, Found, Adopted).

Methods:

+ createPetProfile(species: String, breed: String, ...): void
+ updatePetProfile(description: String, photo: Image): void
+ getPetDetails(): Pet

4. Notification

- notificationID: int
- userID: int
- content: String
- isRead: boolean

Methods:

+ sendNotification(userID: int, content: String): void
+ markAsRead(notificationID: int): void

5. System

- systemName: String
- activeUsers: List<User>

Methods:

+ processImage(image: Image): List<Match>
+ notifyUser(userID: int, notification: Notification): void

6. **DB Manager**

- databaseID: int
- connection: boolean

Methods:

+ save(entity: Object): boolean
+ retrieve(query: String): List<Object>
+ delete(entityID: int): boolean

7. **Match**

- matchID: int
- postID: int
- matchedPetID: int
- similarityScore: float

Methods:

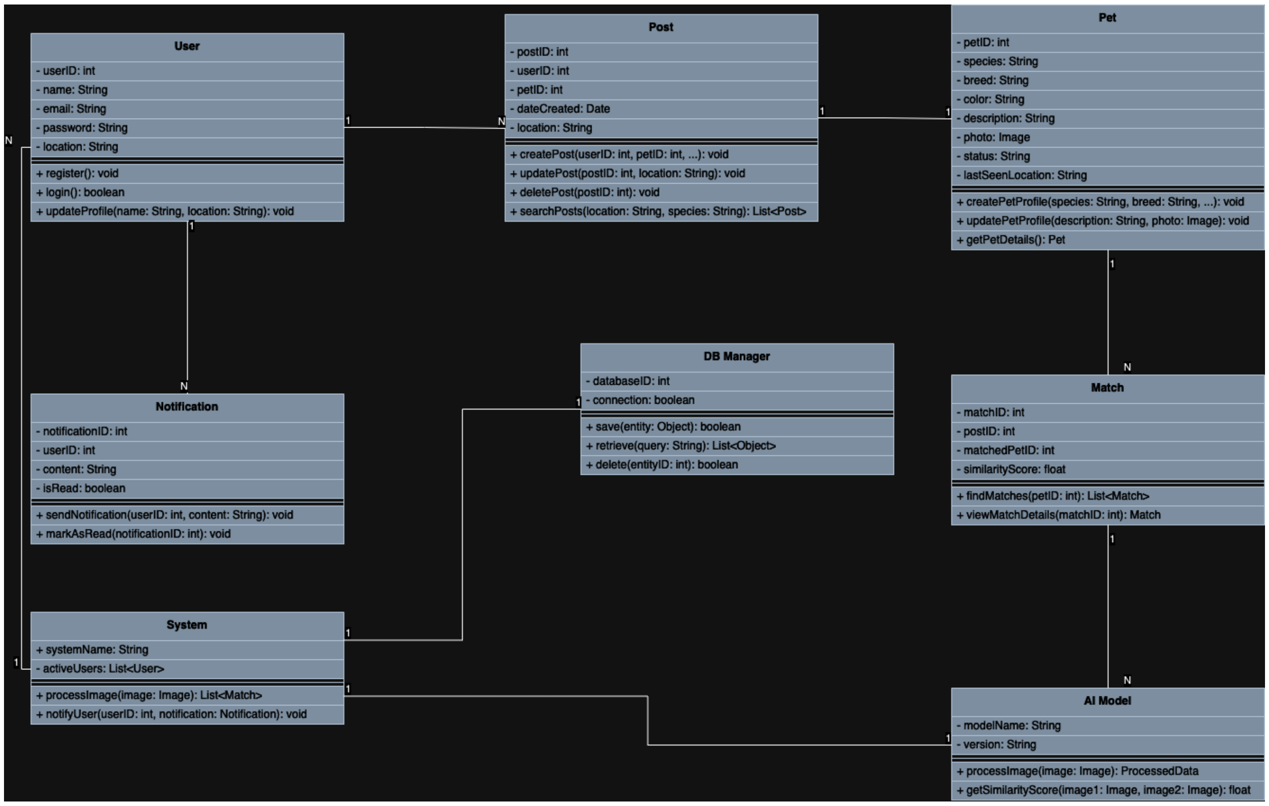
+ findMatches(petID: int): List<Match>
+ viewMatchDetails(matchID: int): Match

8. **AI Model**

- modelName: String
- version: String

Methods:

+ processImage(image: Image): ProcessedData
+ getSimilarityScore(image1: Image, image2: Image): float


User
- userID: int
- name: String
- email: String
- password: String
- location: String
+ register(): void
+ login(): boolean
+ updateProfile(name: String, location: String): void
Post
- postID: int
- userID: int
- petID: int
- dateCreated: Date
- location: String
+ createPost(userID: int, petID: int, ...): void
+ updatePost(postID: int, location: String): void
+ deletePost(postID: int): void
+ searchPosts(location: String, species: String): List<Post>
Pet
- petID: int
- species: String
- breed: String
- color: String
- description: String
- photo: Image
- status: String
- lastSeenLocation: String
+ createPetProfile(species: String, breed: String, ...): void
+ updatePetProfile(description: String, photo: Image): void
+ getPetDetails(): Pet
DB Manager
- databaseID: int
- connection: boolean
+ save(entity: Object): boolean
+ retrieve(query: String): List<Object>
+ delete(entityID: int): boolean
Notification
- notificationID: int
- userID: int
- content: String
- isRead: boolean
+ sendNotification(userID: int, content: String): void
+ markAsRead(notificationID: int): void
Match
- matchID: int
- postID: int
- matchedPetID: int
- similarityScore: float
+ findMatches(petID: int): List<Match>
+ viewMatchDetails(matchID: int): Match
System
+ systemName: String
- activeUsers: List<User>
+ processImage(image: Image): List<Match>
+ notifyUser(userID: int, notification: Notification): void
AI Model
- modelName: String
- version: String
+ processImage(image: Image): ProcessedData
+ getSimilarityScore(image1: Image, image2: Image): float
1
N
1
1
1
N
N
1
1
1
1
1
N
1
N
1
1