School of Computer Science

Project Proposal - IoT Seminar 2024

Doggo: Mobile Application for Dog Owners

Last Updated: 2024-05-17

Mentor: Gili Kamma

Team Name: Nesher

Team Members: Nizan Naor, Shir Tzfania, Raz Olewsky

Table of Contents

Problem Definition

The role of pets, particularly dogs, in human lives has evolved significantly, with dogs often regarded as integral members of the family. However, along with the joy and companionship they bring, dog ownership also entails significant responsibilities, including the maintenance of their health, well-being, and safety. Despite the emotional and social benefits of pet ownership, many dog owners encounter challenges in effectively managing these aspects of care.

One prominent challenge faced by dog owners is the difficulty in monitoring and managing their pet's health and activity levels. Existing methods of tracking a dog's exercise routines, rest patterns, and overall well-being often lack precision and efficiency, leading to a lack of actionable insights for owners. Additionally, concerns regarding the safety and security of pets, particularly when they are outdoors or left unattended, persist among owners.

Furthermore, dog owners express a desire for greater opportunities for socialization and community engagement with other pet owners. Traditional methods of connecting with fellow dog owners, such as visiting dog parks or attending local pet events, may be limited by factors such as geographical proximity or time constraints, hindering the establishment of meaningful connections.

Solution Overview

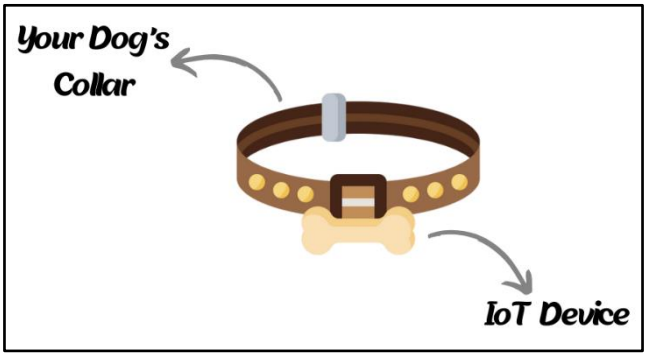
In response to these challenges, our academic project proposes the development of Doggo, a comprehensive mobile application designed to address the multifaceted needs of dog owners.

Doggo is a mobile application that helps dog owners effortlessly monitor and manage their pet's health, activity, and well-being. Doggo empowers the owners to stay connected with their furry friends like never before. Whether you're tracking your dog's daily exercise, planning outdoor adventures with GPS mapping, or scheduling important vet appointments. Doggo provides feedback, motivation, and rewards to dogs for their fitness activities and achievements.

Doggo comes with a compact attachment that easily fits onto your dog's collar or harness, tracking their activity and seamlessly communicating with the mobile app.

The app also provides a social network connecting dog owners, allowing them to make new friends, schedule common trips, and see who is currently in a specific dog park or garden, fostering community engagement and enhancing the overall pet ownership experience.

Illustration:



Existing Alternatives

1. **Pawtrack:** Link
    - Focus: Cat tracking and monitoring.
    - Features: Tracks cat's location, monitors activity, and provides health insights. It has a mobile app for easy access to information.

2. **FitBark:** Link
    - Focus: General pet health and fitness.
    - Features: Monitors pet activity, tracks sleep, and provides health insights. It also has a social component for connecting with other pet owners.

3. **Whistle GO:** Link
    - Focus: Pet tracking and health monitoring.
    - Features: GPS tracking, health monitoring, and activity tracking. It offers a mobile app for real-time updates on the pet's location and well-being.

User Demographics

The expected user demographics encompass a diverse spectrum of dog owners, characterized by various socio-demographic factors and pet ownership experiences.

1. **Age Distribution** - Doggo is anticipated to attract users across different age cohorts, ranging from young adults to seniors. While younger demographics may gravitate towards the application for its technological innovation and social networking features, older users may value its utility in managing the health and well-being of their pets.

2. **Geographical Reach** - Doggo's primary user base is expected to be concentrated in urban areas, where the demand for pet care solutions and dog-friendly amenities is particularly pronounced. Urban users represent a significant segment of Doggo's target audience, as they seek the application for its convenience in navigating dog-friendly amenities and services within densely populated city environments .

3. **Occupational Diversity** - The application's versatility and user-centric design ensure that it fits seamlessly into the lifestyles of users across diverse occupational backgrounds. Whether professionals with demanding schedules, students balancing academics and extracurricular activities, retirees enjoying leisurely pursuits, or homemakers managing household responsibilities, Doggo accommodates the needs of the entire family, fostering a shared commitment to pet care and well-being.

4. **Pet Ownership Experience** - Doggo caters to individuals with varying levels of pet ownership experience, from first-time dog owners to seasoned enthusiasts. Novice owners may rely on the application for guidance in pet care basics, such as nutrition, grooming, and training, while experienced owners may leverage its advanced features for optimizing their pet's health, activity levels, and social interactions.

Features & User Flow

Doggo users will benefit from a plenty of functionalities, including:

1. **Tracking Dog's Activity** – Doggo allows users to effortlessly monitor their dog's activity levels throughout the day, providing insights into their exercise routines, rest periods, and overall health.

2. **Setting Fitness Goals** - With Doggo, owners can set personalized fitness goals for their dogs, whether it's achieving a certain number of steps per day or increasing active playtime, helping to promote a healthier lifestyle for their furry friends.

3. **Looking for Dog's Useful Places** – Doggo provides a directory of dog-friendly places such as parks, beaches, pet stores, and veterinary clinics, making it easy for owners to discover new locations to enjoy with their companions.

4. **Save Important Dog's Properties** – Users can use Doggo to store and organize important information about their dog, such as medical records, vaccination history, dietary preferences, and emergency contacts, all in one convenient location.

5. **Looking for New Friends** – Doggo's social network feature enables users to connect with other dog owners in their area, facilitating the opportunity to arrange playdates, meetups, and form new friendships for both dogs and owners alike.

6. **Receiving Notifications When Dog Has Left the House** – Doggo offers a feature that sends notifications to owners when their dog leaves a predefined safe zone, helping to ensure the pet's safety and providing peace of mind to the owner.

7. **Chat With Other Dog Owners -** Through Doggo's integrated chat functionality, users can engage in conversations with other dog owners, exchanging tips, advice, and experiences related to pet care and ownership.

8. **Looking for Answers to Frequent Questions** - Doggo includes a comprehensive FAQ section where users can find answers to common questions about dog care, training, behavior, health, and more, providing valuable resources and support to pet owners.

Dependencies

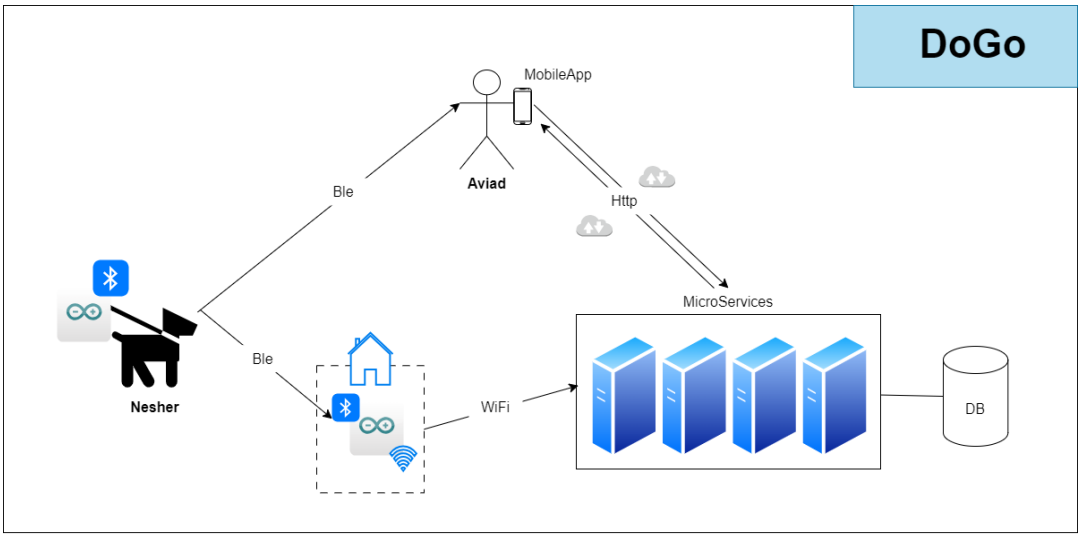
The application would use:

1. **Google Maps API** for indicating the dog's location, activities, and facilities around.

2. **Chat GPT API** to get answers for general questions (FAQ).

3. **Embedded:** Arduino BLE, Arduino IoT, charged battery + cable, USB cable, holder

Scheme of the communication between the features:



Technology Stack

1. **Backend:**
    - Language: Python
    - Framework: Flask
    - Architecture: Micro-services
    - Data base: SQL using postgresSQL

2. **Frontend:**
    - Language: Dart
    - Framework: Flutter
    - Mobile application: Android

3. **Embedded:**
    - Arduino NANO 33 – REV2, Arduino NANO 33 – IoT
    - Language: Arduino programming language (C++)
    - IDE: Arduino IDE

Interfaces

Frontend to Backend:

1. User Authentication:

- POST /api/auth/register – Register a new user.

  Json Message:

  ```
  {
    "username": "example_username",
    "email": "example@example.com",
    "password": "example_password",
    "first_name": "John",
    "last_name": "Doe",
    "phone_number": "123456789",
    "birthdate": "2019-05-20",
    "main_owner": true
  }
  ```

- POST /api/auth/login – Log in an existing user.

  Json Message:

  ```
  {
    "username": "example_username",
    "password": "example_password"
  }
  ```

- POST /api/auth/logout – Log out the current user.

  Json Message:

  ```
  {
    "username": "example_username"
  }
  ```

- GET /api/auth/user/profile – Get the user's profile information.
- PUT /api/auth/user/profile – Update the user's profile information.

  Json Message:

  ```
  {
    "password": "example_password",
    "email": "new_email@example.com",
    "phone_number": "987654321"
  }
  ```

2. Dog Settings:

- POST /api/dogs – Add a new dog.

  Json Message:

  ```
  {
    "name": "Buddy",
    "breed": "Golden Retriever",
    "birthday": "2019-05-20",
    "weight": 25,
    "image": "base64_encoded_image_data"
    "home_latitude": 37.7749,
    "home_longitude": -122.4194,
    "vet_name": "Dr. Strange",
    "vet_phone": 972521234524,
    "vet_latitude": 37.7749,
    "vet_longitude": -122.4194,
    "pension_name": "The best pension"
  }
  ```

- GET /api/dogs/{dog_id} - Get dog's profile information.
- PUT /api/dogs/{dog_id} – Update dog's profile.

  Json Message:

  ```
  {
    "name": "New Name",
   "weight": 30,
    "image": "base64_encoded_image_data "
    "home_latitude": 37.7749,
    "home_longitude": -122.4194
    "vet_name": "Dr. Strange",
    "vet_phone": 972521234524,
    "vet_latitude": 37.7749,
    "vet_longitude": -122.4194,
    "pension_name": "The best pension"
  }
  ```

- DELETE /api/dogs/{dog_id} – Delete a dog.

3. Tracking Dog's Outdoor Activity:

- POST /api/dogs/{dog_id}/activities – Log a new activity for the dog.

  Json Message:

  ```
  {
    "activity_type": "Walk",
    "activity_datetime": "2024-05-20T08:00:00",
    "duration": "01:00:00",
    "distance": 2.5,
    "calories_burned": 150
  }
  ```

- GET /api/dogs/{dog_id}/activities – Get the dog's activity log info.
- DELETE /api/dogs/{dog_id}/activities/{activity_id} – Delete a specific activity.

4. Tracking Dog's Fitness:

- GET /api/dogs/{dog_id}/fitness/{date} – Get the dog's fitness info (steps, calories, distance) for a specific date.
- PUT /api/dogs/{dog_id}/fitness/steps – Update the dog's steps through the day.

  Json Message:

  ```
  {
    "steps": 5000
  }
  ```

- PUT /api/dogs/{dog_id}/fitness/distance – Update the distance the dog reaches through the day.

  Json Message:

  ```
  {
    "distance": 3.5
  }
  ```

5. Setting Fitness Goals:

- POST /api/dogs/{dog_id}/goals – Create a new fitness goal for the dog.

  Json Message:

  ```
  {
    "goal_type": "Daily Steps",
    "target_value": 10000,
    "start_date": "2024-05-01",
    "end_date": "2024-05-31",
    "description": "bla bla"
  }
  ```

- GET /api/dogs/{dog_id}/goals – Retrieve the dog's current fitness goals.
- PUT /api/dogs/{dog_id}/goals/{goal_id} – Update a specific fitness goal.

  Json Message:

  ```
  {
    "goal_type": "Daily Steps",
    "target_value": 10000,
    "start_date": "2024-05-01",
    "end_date": "2024-05-31",
    "description": "bla bla"
  }
  ```

- DELETE /api/dogs/{dog_id}/goals/{goal_id} – Delete a specific fitness goal.

6. Receiving Notifications:

- POST /api/notifications – Subscribe to notifications for when the dog leaves a predefined safe zone.

  Json Message:

  ```
  {
    "notification_type": "Safe Zone Alert",
    "enabled": true
  }
  ```

- DELETE /api/notifications – Unsubscribe from notifications.

7. Chat with Other Dog Owners:

- GET /api/chats – Retrieve all chat rooms for the current user.
- POST /api/chats/{user_id} – Create a new chat room with the specified user. Based on user_id only – **don't need json message.**
- GET /api/chats/{chat_id}/messages – Retrieve messages from a specific chat room.
- POST /api/chats/{chat_id}/messages – Send a message to a specific chat room.

  Json Message:

  ```
  {
    "message": "Hello, let's chat!"
  }
  ```

- DELETE /api/chats/{chat_id} – Delete a specific chat room.

8. Social Networking:

- GET /api/social/friends – Get a list of friends for the authenticated user.
- POST /api/social/friends/{user_id} – Add a new friend for the authenticated user. Based on user_id only – **don't need json message.**
- DELETE /api/social/friends/{user_id} – Remove a friend from the authenticated user's list.

9. Nutrition Information:

- GET /api/dogs/{dog_id}/nutrition - Retrieve the nutrition information for a specific dog.
- POST /api/dogs/{dog_id}/nutrition - Create a new record for a dog's nutrition.

  Json Message:

  ```
  {
    "food_brand": "Royal Canin",
    "food_type": "Dry",
    "food_amount_grams": 200,
    "daily_snacks": 1,
    "notes": "Recommended by the vet"
  }
  ```

- PUT /api/dogs/{dog_id}/nutrition - Update the nutrition information for a specific dog.

  Json Message:

  ```
  {
    "food_brand": "Royal Canin",
    "food_type": "Dry",
    "food_amount_grams": 200,
    "daily_snacks": 1,
    "notes": "Recommended by the vet"
  }
  ```

- DELETE /api/dogs/{dog_id}/nutrition - Delete the nutrition information for a specific dog.

10. Vaccinations Information:

- GET /api/dogs/{dog_id}/vaccinations - Retrieves all vaccination records for a specific dog.
- POST /api/dogs/{dog_id}/vaccinations - Creates a new vaccination record for a specific dog.

  Json Message:

  ```
  {
    "vaccination_type": "Rabies",
    "vaccination_date": "2024-05-15",
    "veterinarian": "Dr. Smith",
    "next_vaccination": "2025-05-15",
    "notes": "Dog's first rabies shot"
  }
  ```

- PUT /api/dogs/{dog_id}/vaccinations/{vaccination_id} - Updates a specific vaccination record for a dog.

  Json Message:

  ```
  {
    "vaccination_type": "Rabies",
    "vaccination_date": "2024-05-15",
    "veterinarian": "Dr. Smith",
    "next_vaccination": "2025-05-15",
    "notes": "Dog's first rabies shot"
  }
  ```

- DELETE /api/dogs/{dog_id}/vaccinations/{vaccination_id} - Deletes a specific vaccination record for a dog.

11. Device Sync:

- PUT /api/devices/sync – Sync the user’s dog's wearable device with the mobile.

  Json Message:

  ```
  {
    "device_id": "123456789",
    "timestamp": "2024-05-20T12:00:00Z",
    "connection_status": "connected"
  }
  ```

- PUT /api/devices/battery - Updates the battery level.

  Json Message:

  ```
  {
    "device_id": "123456789",
    "timestamp": "2024-05-20T12:00:00Z",
    "battery_level": "40%"
  }
  ```

12. FAQ:

- GET /api/faq – Retrieve frequently asked questions and their answers.

13. GPS:

- GET /api/location – Retrieve dog's real-time location based on mobile GPS.

Message Structure - Collar Embedded to Mobile/Home Station Embedded:

The Collar Embedded will send by broadcast several types of messages using ble (Advertising). The mobile and the home station embedded will listen.

There is the combined message structure:

Frequency: Sent periodically every 3 minutes.

```
typedef struct {
  uint16_t deviceID; // 2 bytes
  uint16_t steps; // 2 bytes
  float distance; // 4 bytes
  uint8_t batteryLevel; // 1 byte
} CollarMessage;
```

Home Station Embedded to Backend:

1. Tracking Dog's Fitness:
   - PUT /api/dogs/{dog_id}/fitness/steps – Update the dog's steps through the day.

     Json Message:

     ```
     {
       "steps": 5000
     }
     ```
   - PUT /api/dogs/{dog_id}/fitness/distance – Update the distance the dog reaches through the day.

     Json Message:

     ```
     {
       "distance": 3.5
     }
     ```

2. Device Sync:

- PUT /api/devices/sync – Sync the user’s dog's wearable device with the mobile.

  Json Message:

  ```
  {
    "device_id": "123456789",
    "timestamp": "2024-05-20T12:00:00Z",
    "connection_status": "connected"
  }
  ```

- PUT /api/devices/battery - Updates the battery level.

  Json Message:

  ```
  {
    "device_id": "123456789",
    "timestamp": "2024-05-20T12:00:00Z",
    "battery_level": "40%"
  }
  ```

Database Schemes

Tables

sql db (PostgreSQL)

In this schema:

1. Users Table:
    - **Responsibility**: The Users table stores information about the users of the application.
    - **Usage**: This table is used to manage user accounts and authentication. It stores essential user details such as username, email, password hash, first name, last name, phone number, last activity timestamp, and whether the user is a main owner of a dog.
2. Responsibles Table:
    - **Responsibility**: The Responsibles table maintains information about individuals responsible for specific dogs, such as main owners and veterinarians.
    - **Usage**: This table facilitates the association between dogs and their respective responsible individuals. It stores details like the dog's ID, the main owner's ID, veterinarian's name, and contact information.
3. Dogs Table:
    - **Responsibility**: The Dogs table stores information about individual dogs, including their characteristics and attributes.
    - **Usage**: This table serves as the central repository for dog-related data. It stores details such as the dog's name, breed, birthday, weight, home location coordinates, and an optional image of the dog.

4. User_Dogs Table:
    - **Responsibility**: The User_Dogs table establishes the many-to-many relationship between users and dogs.
    - **Usage**: This table facilitates the association between users and the dogs they own or are responsible for. It enables users to have multiple dogs and vice versa.
5. Collars Table:
    - **Responsibility**: Stores data about collar devices, including connection status and last outdoor activity timestamp. Also maintains the association between collar devices and dogs.
    - **Usage**: Manages collar devices, tracks connection status, and ensures correct pairing with dogs.
6. Home_Station Table:
    - **Responsibility**: The Home_Station table stores information about the home station in each user's home.
    - **Usage**: Manages home stations devices, tracks connection status, and more.
7. Dogs_Nutrition Table:
    - **Responsibility**: The Dogs_Nutrition table stores information about the nutrition of each dog, including their food preferences and feeding habits.
    - **Usage**: This table allows tracking of each dog's nutritional intake, including details such as food brand, type, amount, daily snacks, and any additional notes.

8. Dogs_Vaccinations Table:

   - **Responsibility**: The Dogs_Vaccinations table records information about vaccinations administered to each dog.
   - **Usage**: This table facilitates the management of a dog's vaccination history, including vaccination dates, types, veterinarian details, next vaccination dates, and any relevant notes.

9. Outdoor_Activities Table:

   - **Responsibility**: The Outdoor_Activities table stores data related to outdoor activities and exercises performed by dogs.
   - **Usage**: This table tracks various aspects of outdoor activities, including activity type, date and time, duration, distance covered, and calories burned, providing insights into the dog's physical fitness and well-being.

10. Fitness Table:

    - **Responsibility**: The Fitness table stores fitness-related data for dogs, such as daily step count and distance covered.
    - **Usage**: This table records fitness metrics on a daily basis, enabling tracking of the dog's activity levels and progress over time.

11. Goals Table:

    - **Responsibility**: The Goals table maintains information about fitness and health goals set for individual dogs.
    - **Usage**: This table allows users to set specific goals for their dogs, such as target step counts or weight loss objectives, with defined start and end dates.

12. Friends Table:

- **Responsibility**: The Friends table manages relationships between dogs, facilitating social interactions and playdates.
- **Usage**: This table allows dogs to be connected as friends, enabling owners to organize meetups and socialize with other dogs.

13. Conversations Table and Messages Table:

- **Responsibility**: The Conversations and Messages tables handle messaging functionality between users.
- **Usage**: These tables support user-to-user messaging within the application, storing conversation details and individual messages exchanged between users.

14. The dog_park table:

- **Responsibility:** The dog_park table stores information about dog parks visited by dogs and their owners.
- **Usage:** This table records details such as the dog's ID, the owner's ID, and the geographic coordinates (latitude and longitude) of the dog park visited. It allows users to track the locations where their dogs have been taken for outdoor activities or socialization.

15. The FAQ table:

- **Responsibility:** The FAQ table serves as a repository for frequently asked questions and their corresponding answers.
- **Usage:** This table contains a list of common questions (stored in the 'question' column) along with their respective answers (stored in the 'answer' column). It provides users with quick access to information and assistance regarding various aspects of the Doggo application, including pet care, features, and troubleshooting. Each FAQ entry is uniquely identified by its FAQ_id.

The Scheme:

```
CREATE TABLE users (
  user_id SERIAL PRIMARY KEY NOT NULL,
  username VARCHAR(50) UNIQUE NOT NULL,
  email VARCHAR(255) UNIQUE NOT NULL,
  password_hash VARCHAR(255) NOT NULL,
  first_name VARCHAR(50) NOT NULL,
  last_name VARCHAR(50) NOT NULL,
  birthday DATE NOT NULL,
  phone_number VARCHAR(15) UNIQUE NOT NULL,
  last_activity TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  main_owner BOOLEAN
);

CREATE TABLE responsibles (
  dog_id INTEGER REFERENCES dogs(dog_id),
  main_owner_id INTEGER REFERENCES users(user_id),
  vet_name VARCHAR(100),
  vet_phone VARCHAR(15) NOT NULL,
  vet_latitude FLOAT,
  vet_longitude FLOAT,
  pension_name VARCHAR(100),
  PRIMARY KEY (dog_id)
);

CREATE TABLE dogs (
  dog_id SERIAL PRIMARY KEY NOT NULL,
  collar_id INTEGER REFERENCES collars(collar_id),
  name VARCHAR(50) NOT NULL,
  breed VARCHAR(50),
  birthday DATE NOT NULL,
  weight FLOAT,
  image BYTEA,
  home_latitude FLOAT NOT NULL,
  home_longitude FLOAT NOT NULL
);
```

```
CREATE TABLE user_dogs (
  user_id INTEGER REFERENCES users(user_id),
  dog_id INTEGER REFERENCES dogs(dog_id),
  PRIMARY KEY (user_id, dog_id)
);

CREATE TABLE collars {
  collar_id SERIAL PRIMARY KEY NOT NULL,
  connection_status VARCHAR(25) NOT NULL,
  last_outdoor_activity TIMESTAMP,
  dog_id INTEGER REFERENCES dogs(dog_id) UNIQUE
};

CREATE TABLE home_stations {
  station_id SERIAL PRIMARY KEY NOT NULL,
  connection_status VARCHAR(25) NOT NULL,
  user_id INTEGER REFERENCES users(user_id) UNIQUE
};

CREATE TABLE dogs_nutrition (
  dog_id INTEGER REFERENCES dogs(dog_id),
  food_brand VARCHAR(100),
  food_type VARCHAR(100),
  food_amount_grams INTEGER,
  daily_snacks INTEGER,
  notes TEXT,
  PRIMARY KEY (dog_id)
);

CREATE TABLE dogs_vaccinations (
  dog_id INTEGER REFERENCES dogs(dog_id),
  vaccination_date DATE NOT NULL,
  vaccination_type VARCHAR(100) NOT NULL,
  veterinarian VARCHAR(100),
  next_vaccination DATE,
  notes TEXT,
  PRIMARY KEY (dog_id, vaccination_date, vaccination_type)
);
```

```
CREATE TABLE outdoor_activities (
  activity_id SERIAL PRIMARY KEY,
  dog_id INTEGER REFERENCES dogs(dog_id),
  activity_type VARCHAR(50) NOT NULL,
  activity_datetime TIMESTAMP NOT NULL,
  duration INTERVAL,
  distance FLOAT,
  calories_burned INTEGER
);

CREATE TABLE fitness (
  dog_id INTEGER REFERENCES dogs(dog_id),
  date DATE NOT NULL,
  distance FLOAT,
  steps_today INTEGER,
  calories_burned INTEGER,
  PRIMARY KEY (dog_id, date)
);

CREATE TABLE fitness_goals (
  goal_id SERIAL PRIMARY KEY,
  dog_id INTEGER REFERENCES dogs(dog_id),
  goal_type VARCHAR(50),
  target_value INTEGER NOT NULL,
  start_date DATE,
  end_date DATE,
  description VARCHAR(1000)
);

CREATE TABLE friends (
  friendship_id SERIAL PRIMARY KEY,
  dog_1_id INTEGER REFERENCES dogs(dog_id),
  dog_2_id INTEGER REFERENCES dogs(dog_id),
  friendship_status VARCHAR(50)
);
```

```
CREATE TABLE chats(
  chat_id SERIAL PRIMARY KEY,
  user_id_1 INTEGER REFERENCES users(user_id),
  user_id_2 INTEGER REFERENCES users(user_id),
  last_message_datetime TIMESTAMP
);

CREATE TABLE messages (
  message_id SERIAL PRIMARY KEY,
  conversation_id INTEGER REFERENCES conversations(conversation_id),
  sender_id INTEGER REFERENCES users(user_id),
  message_content TEXT,
  sent_datetime TIMESTAMP
);

CREATE TABLE dog_park (
  dog_id INTEGER REFERENCES dogs(dog_id),
  user_id INTEGER REFERENCES users(user_id),
  dog_park_latitude FLOAT NOT NULL,
  dog_park _longitude FLOAT NOT NULL
);

CREATE TABLE FAQ (
  FAQ_id SERIAL PRIMARY KEY,
  question VARCHAR(500),
  answer VARCHAR(500)
);
```

Illustration: