



School of Computer Science

Project Design - IoT Seminar 2024



Doggo: Mobile Application for Dog Owners

Date : 2024-09-10

Mentor : Gili Kamma

Project Number : 15006502

Team Members : Nizan Naor, Shir Tzfania, Raz Olewsky

GitHub Repository : shir-tz/Doggo

Project Demo : [YouTube Link](#)

Table of Contents

Use Cases	3
Requirements	4
Technology Stack.....	5
Implementations	6
Endpoints (API)	15
Database Schemes.....	37

Use Cases

1. **Steps Data Collection:**

The system is designed to collect and record step data from an IoT device attached to the dog's collar. This device continuously monitors and tracks the total number of steps the dog takes throughout the day. This data is captured in real-time, providing a comprehensive overview of the dog's daily physical activity.

2. **Activity Data Monitoring and Progress Summaries:**

The system regularly updates and compares the dog's activity data with user-set goals. It provides detailed progress summaries showing how the dog is performing over time. Users receive these summaries to see how well their dog is hitting its activity targets and to make any necessary changes to improve the dog's fitness plan.

3. **Comprehensive Management of Dog's Essential Information:**

The system allows users to manage and track their dog's essential information, including physical attributes, health data, and medical history.

4. **Integrated Map Feature for Locating Dog-Friendly Locations:**

The system offers users a fully integrated map feature designed to help them easily locate nearby dog-friendly places. This includes parks, pet-friendly restaurants, pet stores, and other relevant locations where dogs are welcome.

Requirements

1. **Track Daily Fitness Data:**

The system tracks the dog's activity by monitoring its steps, calories burned, and distance covered using an IoT device. This data provides a comprehensive overview of the dog's daily physical activity, helping users to monitor and maintain their dog's health and fitness.

2. **Monitor Progress Towards Goals:**

The system must regularly compare the dog's activity data from the IoT device with user-defined goals and generate detailed progress summaries. Users will be able to track their dog's performance on a daily, weekly, and monthly basis, providing a comprehensive view of the dog's activity over time. This ensures users can monitor progress and adjust goals as needed.

3. **Manage Dog's properties:**

Users will be able to store and manage their dog's health and property data, such as medical records, dog care details, and other relevant information. This ensures secure storage and easy access for ongoing monitoring and updates.

4. **Locate Dog-Friendly Places:**

Users will be able to find dog-friendly locations, including parks, pet-friendly restaurants, pet stores, and more, using the map feature. They should be able to search, filter, and navigate to these destinations through the map, providing easy access to amenities and enhancing their dog's lifestyle and social experiences.

Technology Stack

1. Backend:

- Language: Python
- Framework: Flask
- Architecture: Monolith
- Data base: SQL using PostgreSQL
- Cloud: AWS

2. Frontend:

- Language: Dart
- Framework: Flutter
- Mobile application: Android

3. Embedded:

- Arduino NANO 33 – IoT, Charged Battery (soldered electric circle)
- Dog bone case – 3D printed
- Language: Arduino programming language (C++)
- IDE: Arduino IDE

Implementations

1. **Backend:**

Overview –

The back end of the app is built using Flask, a lightweight Python web framework. It serves as the main hub for handling API requests, managing data, and communicating with the database.

- **Structure and Organization**

The backend follows a modular structure for maintainability and scalability.

Routes are organized into separate files based on different entities, such as users, dogs, and collars, to make it easier to manage and extend the codebase.

routes folder contains all route definitions organized by feature. For example:

- user_routes.py - Handles all user-related routes (e.g., registration, login).
- dog_routes.py - Manages routes related to dog data (e.g., add dog, steps).

- **Database Interaction**

The backend connects to a PostgreSQL database (hosted on AWS RDS) using the psycopg2 library. It handles various CRUD operations, such as retrieving, updating, and deleting data related to users and dogs. The database schema is designed to store user information, dog's activity data and more.

- **Dockerized Application and Deployment**

The backend is containerized using Docker, making it easy to deploy in any environment.

In addition, the backend is deployed on an AWS EC2 instance with Docker Hub used to host the images. This setup ensures scalability and high availability.

- **Backend Functional Workflow**

- Conversion Tables for Accurate Calculations:

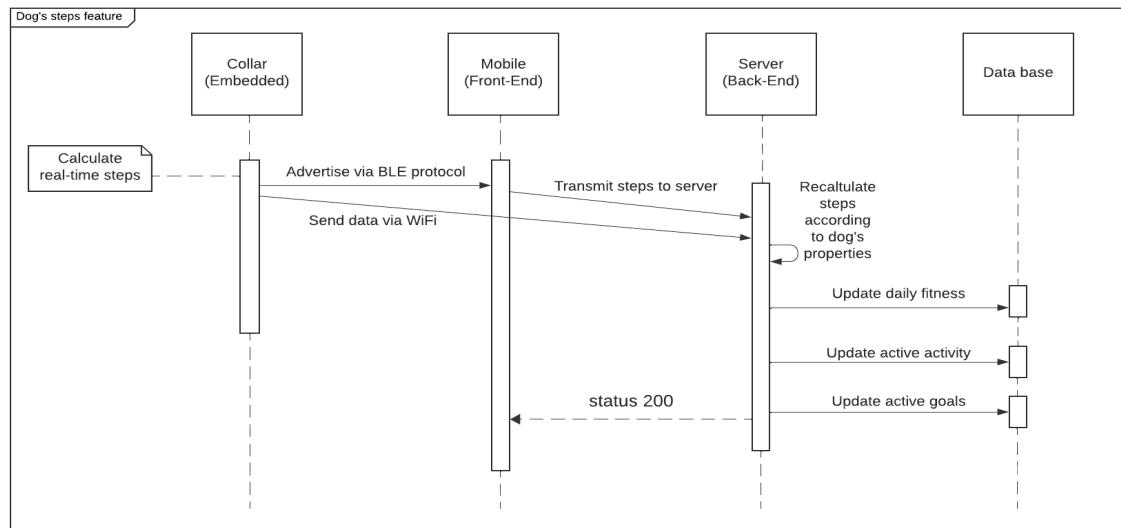
The backend uses conversion tables based on the dog's size and weight to adjust step count, calculate distance, calories burned, and bcs. Each dog category (toy, small, medium, large, extra-large) has specific ranges for weight, height, and step length to ensure precision.

Dog Size	Weight Range (kg)	Factor Range	Height Range (cm)	Step Length Range (cm)
Toy Dog	2 - 5	0.5 - 0.7	0 - 25	25 - 35
Small Dog	5 - 10	0.7 - 0.9	25 - 35	35 - 45
Medium Dog	10 - 25	0.9 - 1.1	35 - 50	45 - 55
Large Dog	25 - 40	1.1 - 1.3	50 - 70	55 - 65
Extra Large Dog	40 - 80	1.3 - 1.6	70 - 100	65 - 80

- Fitness Update Workflow:

The backend receives the initial step count from either the IoT collar or mobile device. Using the conversion tables, it adjusts the count for accuracy, calculates the distance traveled and calories burned, and then updates the active activity and any existing fitness goals.

The following sequence diagram illustrates the workflow of the fitness update process:



- **Background Task Execution:**

In the backend, a single thread handles multiple background tasks.

The thread runs every 10 seconds, performing actions like:

- **Collar Connection Check:** If no steps are received from a collar for 5 hours, the system disconnects the collar.
- **Activity Monitoring:** The system checks for active activities and ends them if they've been running for 5 hours continuously.
- **Goal Management Background Task:**
 1. Daily Goals: At the start of each new day, the system finishes the daily goals from the previous day.
 2. Weekly Goals: At the start of each new week, the system finishes the weekly goals from the previous week.
 3. Monthly Goals: On the 1st of each month, the system finishes the monthly goals from the previous month.

After finishing the current goals, the system uses goal templates to create new goals for the day, week, or month, ensuring users have updated targets to work towards.

This thread ensures that system tasks are handled regularly and automatically without blocking other processes.

2. **Frontend:**

Overview –

The front-end implementation of Doggo is designed using Flutter, which provides a single codebase for both iOS and Android platforms. Flutter's rich set of pre-designed widgets and its reactive framework allows for the creation of a highly responsive user interface. The design follows a modern, user-friendly approach with an emphasis on performance.

Technologies and Frameworks –

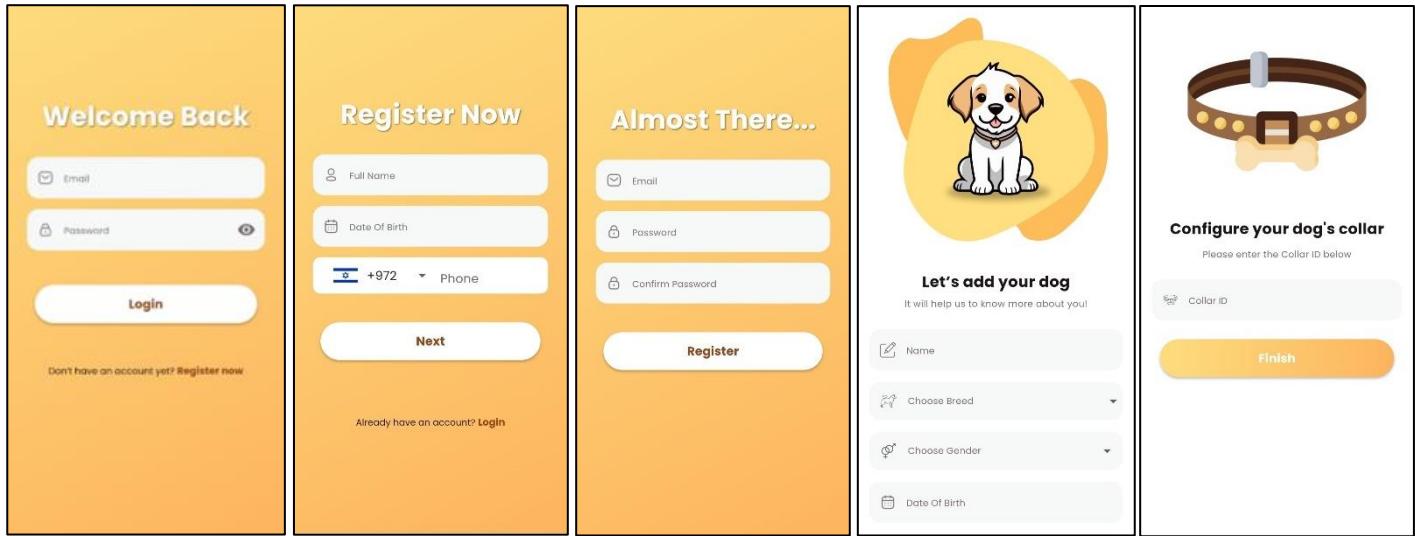
The Doggo App is built exclusively for **Android** using the **Flutter SDK**, ensuring a robust and responsive design. It leverages several third-party packages for functionality:

- **UI Components:** The app uses Carousel Slider and Simple Circular Progress Bar to provide an engaging and user-friendly interface.
- **Mapping and Location:** Flutter Map and Geolocator are integrated for location-based features, enabling users to search dog-friendly places and set favorites.
- **Networking and Data:** The app relies on the HTTP package for backend communication and Shared Preferences for local data storage.
- **Bluetooth Integration:** Flutter Reactive BLE handles Bluetooth communication, allowing the app to connect to and manage dog collars.
- **Other Packages:** Permission Handler ensures the app properly manages device permissions, and Table Calendar helps with event scheduling. The app also supports international phone number input through Intl Phone Number Input.

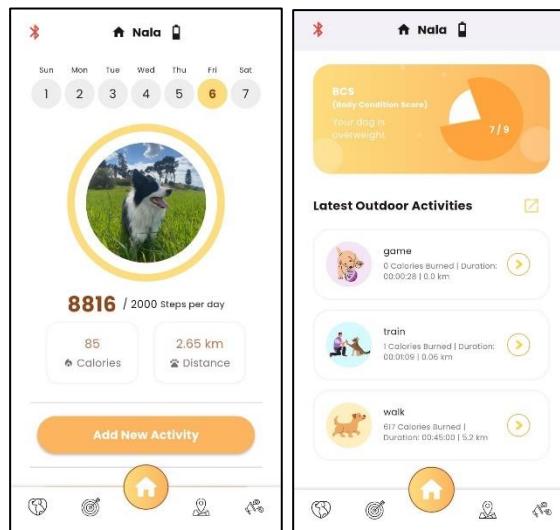
Custom assets like icons, fonts, and images are embedded to maintain a cohesive brand identity.

Main Screens –

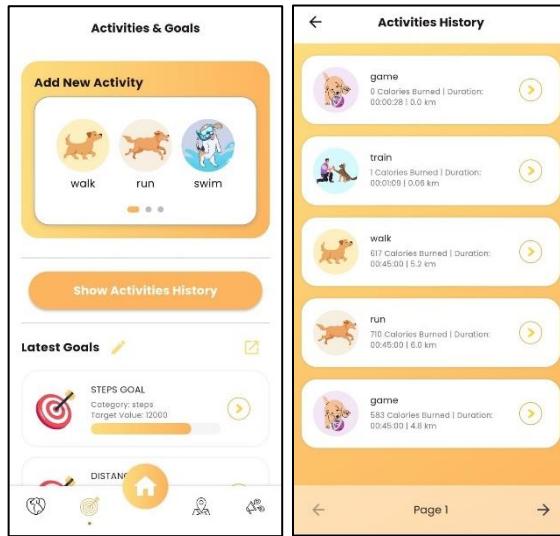
- Login & Sign-Up Screens:** These screens enable users to sign up or log in using their email and phone number. The phone number input is powered by the `intl_phone_number_input` package, ensuring accurate validation and formatting for various countries. During the sign-up process, users also add their dog's information and configure the dog to a collar.



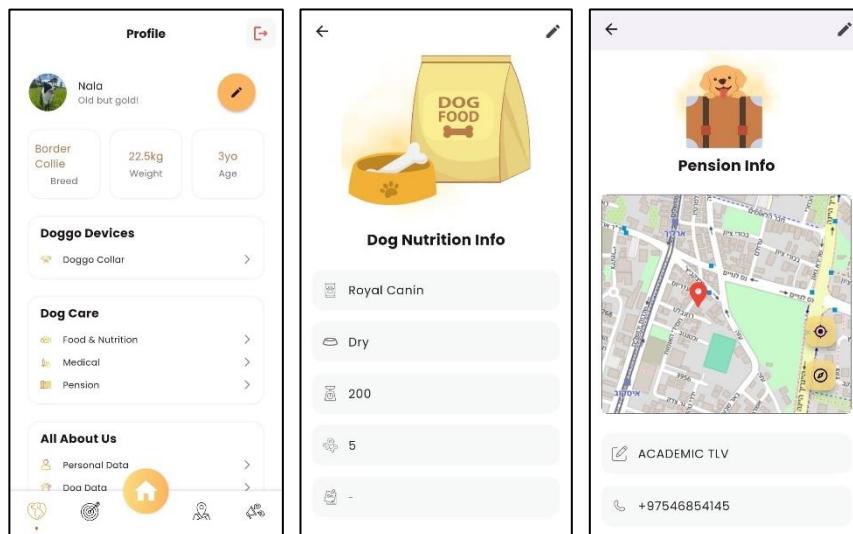
- Home Screen:** The home screen provides users with an intuitive overview of key information. It includes widgets such as progress indicators to display the user's and their dog's activity progress, a dynamic list showcasing user and dog-related data, and quick navigation options for accessing essential features.



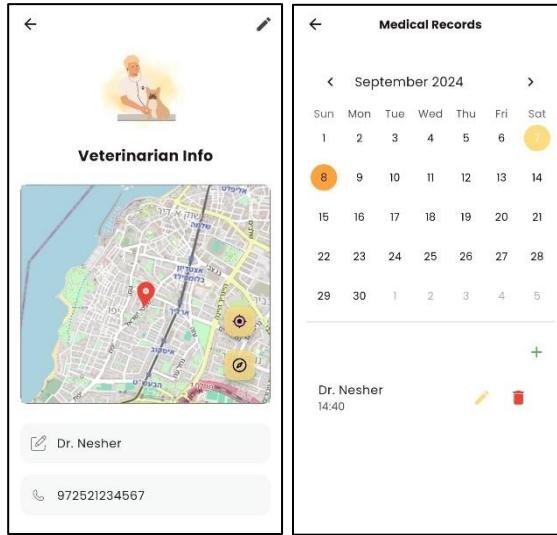
- **Activities & Goals Screen:** Users can track their dog's daily activities, set fitness goals, and monitor progress over time on this screen.



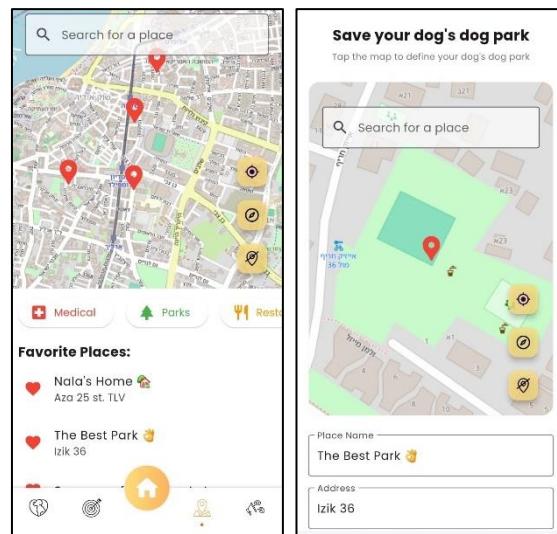
- **Profile Screen:** This screen displays the user's personal details, including their name, phone number, and other relevant information, with options to update them. Validation is managed using custom methods, and errors are displayed via the `errorText` property in input fields. The screen also includes sections for the dog's nutrition, medical records, and quick access to pension and vet information.



- **Medical Screens:** Designed to store and present the dog's medical information, these screens offer an organized view of vet information medical history, and appointments. Users can update this information as needed and view critical medical details immediately.



- **Map Screen:** The map screen, integrated with flutter_map (open street map API), allows users to search for nearby dog-friendly places, including medical facilities, restaurants, parks, pensions and more. Users can filter locations by category.



Services –

- **BLE Service:**

This service manages Bluetooth Low Energy (BLE) communication between the app and the dog collar. It handles device scanning, connecting to the collar, and reading the battery level and step count characteristics. The service ensures real-time data updates through periodic readings (each 5 seconds) and sends the information to the backend via HTTP requests.

- **HTTP Service:**

Responsible for making HTTP requests to the backend, this service facilitates sending key data like battery level and step count to the server. It uses `http` package for communication with the backend API.

- **Preferences Service:**

This service manages local storage using shared preferences. It stores and retrieves essential user and dog information, such as user and dog IDs, ensuring persistent data access across the app.

3. **Embedded:**

This Arduino code for Doggo uses an Arduino Nano 33 IoT to track and transmit data related to the dog's steps and battery level using both BLE and Wi-Fi. Here's a simplified breakdown of how it works:

- **BLE Services:** Two BLE services are defined—one for battery level and another for step counting. The Arduino advertises these services for nearby devices to connect and read data.
- **Wi-Fi Setup:** Wi-Fi credentials are defined, allowing the Arduino to connect to a remote server when BLE is not available. It sends step and battery data to the server via HTTP every 10 seconds.
- **Step Detection:** Steps are detected using an Inertial Measurement Unit (IMU) sensor (BMI270/LSM6DS3 – motion sensors). When movement (above a threshold) is detected, based on accelerometer, the step count is increased.
- **Connection Management:** The code continuously switches between BLE and Wi-Fi based on availability. If the dog owner is near the sensor and connected to the app, the code will prefer connection via BLE and send the data locally. If not, it tries to connect to Wi-Fi and sends data to the server.
- **Battery Monitoring:** The battery level is periodically updated and sent either via BLE or HTTP, depending on the connection state.

In essence, the Arduino alternates between two wireless technologies, BLE and Wi-Fi, to monitor and report your dog's activity and battery status.

Endpoints (API)

Mobile to backend

1. User Authentication

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

Json Message	<pre>{ "email": "test@gmail.com", "password": "123123", "name": "Nizan Naor", "date_of_birth": "1995-04-11", "phone_number": "+972546854144" }</pre>
Response	<pre>{ "user_id": 1 }</pre> status 201

- PUT /api/user/login – Log in an existing user.

Json Message	<pre>{ "email": "example@example.com", "password": "example_password" }</pre>
Response	<pre>{ "user_id": 1, "dog_id": 12 }</pre> status 200

- PUT /api/user/logout – Log out the current user.

Query Params	user_id: 1
Response	status 200

2. User's Information

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

Query Params	user_id: 1
Response	{ "email": "user@gmail.com", "email": "user@gmail.com", "password": "123", "name": "John", "date_of_birth": "Sat, 04 Nov 1995 00:00:00 GMT", "phone_number": "+972549034113" } status 200

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

Json Message	{ "user_id": 123, "email": "new_email@example.com", "password": "example_password", "name": "John", "date_of_birth": "1995-05-11" "phone_number": "+972545366987" }
Response	{ "user_connection": True/False } status 200

- **DELETE /api/user/profile** – Delete user from the system --> Delete user's dogs.

Json Message	<pre>{ "user_id": 123, "email": "new_email@example.com", "password": "example_password", "name": "John", "date_of_birth": "1995-05-11" "phone_number": "+972456321545" }</pre>
Response	status 200

- **GET /api/user/connection** – Check if user is logged in.

Query Params	user_id: 123
Response	status 200

- **GET /api/user/check_password** – Check user's password.

Query Params	user_id: 123 password: nesher123
Response	<pre>{ "is_valid": True }</pre> status 200

3. Dog Settings

- **POST /api/dog/add** – Add a new dog.

Json Message	<pre>{ "user_id": 5, "name": "Buddy", "description": "My best friend!", "breed": "Golden Retriever", "gender": "male", "date_of_birth": "2019-05-20", "size": "large", "color": "brown" }</pre>
--------------	---

	<pre> "weight": 25.5, "height": 50, "description": "Best dog ever!" } </pre>
Response	<pre> { "dog_id": 2 } </pre> <p>status 201</p>

- GET /api/dog/profile - Get dog's profile information.

Query Params	dog_id: 2
Response	<pre> { "name": "Buddy", "breed": "Golden Retriever", "gender": "male", "date_of_birth": "Sat, 04 Nov 1995 00:00:00 GMT", "weight": 25.5, "height": 30, "description": "Best dog ever!" } </pre> <p>status 200</p>

- PUT /api/dog/profile – Update dog's profile.

Json Message	<pre> { "dog_id": 1, "name": "New Name", "breed": "Golden Retriever", "gender": "Male", "date_of_birth": "2019-05-20", "weight": 30.5, "height": 100, } </pre>
--------------	--

	“description”: “Best dog ever!” }
Response	status 200

- DELETE /api/dog/delete – Delete a dog.

Query Params	dog_id: 1
Response	status 200

4. Dog's Care Info

- PUT /api/dog/pension – Add/Update pension.

Json Message	<pre>{ "dog_id": 1, "pension_name": "BLA", "pension_phone": "+97211222222", "pension_latitude": 32.06383467209801, "pension_longitude": 34.784212577883665 }</pre>
Response	status 200

- PUT /api/dog/vet – Add/Update dog's vet info.

Json Message	<pre>{ "dog_id": 1, "vet_name": "New Name", "vet_phone": "+97211222222", "vet_latitude": 32.06383467209801, "vet_longitude": 34.784212577883665 }</pre>
Response	status 200

- GET/api/dog/pension – Get dog's pension info.

Json Message	dog_id: 16
Response	<p>Json Message:</p> <pre>{ "pension_name": BLA, "pension_phone": "+97211222222", "pension_latitude": 32.06383467209801, "pension_longitude": 34.784212577883665 }</pre> <p>Else:</p> <p>204 (content is missing)</p>

- GET/api/dog/vet – Get dog vet info.

Query Params	dog_id: 11
Response	<pre>{ "vet_name": "New Name", "vet_phone": +972545633989, "vet_latitude": 32.06383467209801, "vet_longitude": 34.784212577883665 }</pre> <p>Status 200</p> <p>Else:</p> <p>204 (content is missing)</p>

5. Collars

- PUT /api/collar/add - Add new collar.

Json Message	<pre>{ "collar_id": "123", "dog_id": 11 }</pre>
Response	status 200

- GET /api/collar/get – Get collar id by dog id.

Query Params	dog_id: 1
Response	<pre>{ "collar_id": 65 }</pre> <p>status 200</p>

- GET /api/collar/connectionStatus – Is the collar connected and how?

Query Params	collar_id: 12
Response	{ "ble_connected": True, "wifi_connected": False } status 200

- GET /api/dog/collar/availability – Is collar available?

Query Params	collar_id: 12
Response	{ "availability": True } status 200

- PUT /api/collar/disconnect – Disconnect collar from dog.

Query Params	collar_id: 15
Response	status 200

- PUT /api/collar/battery - Updates the battery level by dog id.

Query Params	{ "dog_id": 5, "battery_level": 40 }
Response	status 200

- GET /api/collar/battery - gets the battery level by collar id.

Query Params	collar_id: 13
Response	{ "battery_level": 58 } status 200

6. Dog's Activities

- GET /api/dog/activities/all – Get n activities.

Query Params	dog_id: 6 limit: 8 offset: 5
Response	List of: { "activity_id": 12, "activity_type": "run", "calories_burned": 0, "distance": 0.01, "duration": "00:45:00", "end_time": "Fri, 23 Aug 2024 09:45:00 GMT", "start_time": "Fri, 23 Aug 2024 09:00:00 GMT", "steps": 20 } status 200

- GET /api/dog/activities – Get a chosen activity.

Query Params	activity_id: 1
Response	{ "calories_burned": 0, "distance": 0.01, "duration": "00 days 00:45:00", "end_time": "Fri, 23 Aug 2024 09:45:00 GMT", "start_time": "Fri, 23 Aug 2024 09:00:00 GMT", "steps": 20 } Status 200

- POST /api/dog/activities – add new activity

Query Params	activity_id: 1
Response	<pre>{ "activity_id": 3 }</pre> Status 201

- PUT /api/dog/activities/end – End an activity.

Query Params	activity_id: 1
Response	status 200

- DELETE /api/dog/activities – Delete a specific activity.

Query Params	activity_id: 1
Response	status 200

7. Dog's Fitness

- GET /api/dog/fitness – Get dog's daily fitness.

Query Params	dog_id: 6 date: 2024-08-31
Response	<pre>{ "distance": 12.3, "steps": 2333, "calories_burned": 3333 }</pre> status 200

- PUT /api/dog/fitness – Update the dog's steps, distance and calories through the day (Receiving data from mobile)

Query Params	dog_id: 6 steps: 6416
Response	status 200

- GET /api/dog/bcs – Get bcs of chosen dog.

Query Params	dog_id: 4
Response	5, status 200

8. Dog's Goals

- GET /api/dog/goal_templates/all – Get dog's goal templates.

Query Params	dog_id: 4
Response	<p>List of:</p> <pre>{ "dog_id": 123, "template_id": 2, "frequency": "daily", "category": "steps", "target_value": 1000, (if it's distance --> float. Unless --> int) }</pre> <p>status 200 or 204</p>

- GET /api/dog/goal_templates – Get chosen goal template.

Query Params	template_id: 4
Response	<pre>{ "dog_id": 123, "template_id": 2, "frequency": "daily", "category": "steps", "target_value": 1000, (if it's distance --> float. Unless --> int) }</pre> <p>status 200</p>

- GET /api/dog/goals/all – Get dog's goals.

Query Params	dog_id: 41 Limit: 60 Offset: 5
Response	<p>List of:</p> <pre>{ "goal_id": 123, "start_date": "Mon, 09 Sep 2024 00:00:00 GMT", "end_date": "Mon, 16 Sep 2024 00:00:00 GMT", "category": "steps", "current_value": 200, (for steps/calories_burned : int. For distance: float) "target_value": 1000, "done": False (would be True after completing) "is_finished": False }</pre> <p>status 200 or 204 if not found</p>

- GET /api/dog/goals – Get goal log.

Query Params	goal_id: 4
Response	<pre>{ "start_date": "Mon, 09 Sep 2024 00:00:00 GMT", "end_date": "Mon, 16 Sep 2024 00:00:00 GMT", "category": "steps", "current_value": 200, (for steps/calories_burned : int. For distance: float) "target_value": 1000, "done": False (would be True after completing) "is_finished": False (if goal passed the end_date --> True) }</pre> <p>status 200</p>

- POST /api/dog/goals/add – Create a new goal (new goal template).

Json Message	<pre>{ "dog_id": 2, "target_value": 2000, "frequency": "weekly", "category": "steps" }</pre>
Response	Status 201

- GET /api/dog/dailyStepsGoal – Retrieve the dog's daily steps goal.

Query Params	Dog_id: 15
Response	<pre>{ "daily_steps_goal": 1000 }</pre> <p>status 200 (default value = 2000)</p>

- PUT /api/dog/goals – Update a goal template --> updating also the active goal using the template's data.

Query Params	Template_id: 5 Target_value: 600
Response	Status 200

- DELETE /api/dog/goal_templates – Delete a goal template --> Deleting also the active goal using the template's data.

Query Params	Template_id: 5
Response	Status 200

- DELETE /api/dog/goals – Delete a goal.

Query Params	goal_id: 5
Response	Status 200

9. Dog's Nutrition

- GET /api/dog/nutrition - Retrieve the nutrition information for a specific dog.

Json Message	{ "dog_id": 16, "food_brand": "Royal Canin", "food_type": "Dry", "food_amount_grams": 200, "daily_snacks": 1, "notes": "Recommended by the vet" }
Response	Status 200 Status 204 if there is no nutrition record for chosen dog

- PUT /api/dog/nutrition - Create/Update a new record for a dog's nutrition.

Json Message	{ "dog_id": 16, "food_brand": "Royal Canin", "food_type": "Dry", "food_amount_grams": 200, "daily_snacks": 1, "notes": "Recommended by the vet" }
Response	Status 201

- DELETE /api/dog/nutrition - Delete the nutrition information for a specific dog.

Query Params	dog_id: 5
Response	Status 200

10. Dog's Vaccinations

- GET /api/dog/vaccinations - Retrieves all vaccination records for a specific dog.

Query Params	dog_id: 16 vaccination_type: "Distemper" limit: 3 offset: 0
Response	List of: { "vaccination_id": 4, "vaccination_date": "Mon, 16 Sep 2024 00:00:00 GMT", "vaccination_type": "Distemper", "dosage": "4", "vet_name": "Dr. Nesher", "next_vaccination": Mon, 16 Sep 2024 00:00:00 GMT, } ... Status 200

- POST /api/dog/vaccinations - Creates a new vaccination record for a specific dog.

Json Message	{ "dog_id": 16, "vaccination_id": 4, "vaccination_date": "Mon, 16 Sep 2024 00:00:00 GMT", "vaccination_type": "Distemper", "dosage": "4", "vet_name": "Dr. Nesher", "next_vaccination": Mon, 16 Sep 2024 00:00:00 GMT, }
Response	Status 201

- PUT /api/dog/vaccinations - Updates a specific vaccination record.

Json Message	<pre>{ "vaccination_id": 4, "vaccination_date": "Mon, 16 Sep 2024 00:00:00 GMT", "vaccination_type": "Distemper", "dosage": "4", "vet_name": "Dr. Nesher", "next_vaccination": "Mon, 16 Sep 2024 00:00:00 GMT" }</pre>
Response	Status 200

- DELETE /api/dog/vaccinations - Deletes a specific vaccination record for a dog.

Query Params	vaccination_id: 5
Response	Status 200

11. Dog's Medical Records

- GET /api/dog/medical_records/days - Retrieves a dictionary: for each day of the chosen month – True if there is at least one medical record for the chosen dog.
Unless - False

Query Params	dog_id: 15 Month: 5 Year: 1999
Response	<pre>{ "1": false, "2": true, . . "30": true, 31: false }</pre> status 200

- GET /api/dog/medical_records/by_date - Retrieves medical records according to date.

Query Params	dog_id: 15 Date: 1999-02-15
Response	<p>List of dictionaries:</p> <pre>[{ "record_id": 16, "dog_id": 16, "vet_name": "Dr. Smith", "address": "9101 Maple Road, Springfield, IL", "record_datetime": "Sun, 21 Jul 2024 11:22:20 GMT", "description": "Treated for flea infestation and ear infection" }]</pre>

- GET /api/dog/medical_records - Returns a medical record according to record_id.

Query Params	record_id: 3
Response	<p>Json Message:</p> <pre>{ "dog_id": 52, "record_id": 16, "vet_name": "Dr. Smith", "address": "9101 Maple Road, Springfield, IL", "record_datetime": "2024-06-15 09:45:00", "description": "Treated for flea infestation and ear infection" }</pre> <p>status 200</p>

- POST /api/dog/ medical_records - Creates a new medical record for a specific dog.

Json Message	<pre>{ "dog_id": 52, "vet_name": "Dr. Smith", "address": "9101 Maple Road, Springfield, IL", "record_datetime": "2024-06-15 09:45:00", "description": "Treated for flea infestation and ear infection" }</pre>
Response	Status 201

- PUT /api/dog/medical_records- Updates a specific medical record.

Json Message	<pre>{ "record_id": 16, "vet_name": "Dr. Smith", "address": "9101 Maple Road, Springfield, IL", "record_datetime": "2024-06-15 09:45:00", "description": "Treated for flea infestation and ear infection" }</pre>
Response	Status 200

- GET /api/dog/ medical_records - Returns a medical record according to record_id.

Query Params	record_id: 3
Response	<pre>{ "address": "9101 Maple Road, Springfield, IL", "description": "Treated for flea infestation and ear infection", "dog_id": 101, "record_datetime": "Sat, 15 Jun 2024 09:45:00 GMT", "record_id": 3, "vet_name": "Dr. Sarah Davis" }</pre> <p>status 200</p>

12. Places

- GET /api/ places/all – Retrieve all places.

Query Params	
Response	list of : { "address": "Rokach Blvd, Tel Aviv", "place_latitude": 32.0853, "place_longitude": 34.7882, "place_name": "Yarkon Park", "place_type": "parks" }..... Status 200

- GET /api/ places/by_type – Retrieve all places by type.

Query Params	place_type: "medical"
Response	list of: { "address": "Rokach Blvd, Tel Aviv", "place_latitude": 32.0853, "place_longitude": 34.7882, "place_name": "Yarkon Park", "place_type": "parks" }....., Status 200

13. Dog's Favorite Places

- GET /api/favorite_places – dog's favorite places.

Query Params	dog_id : 15
Response	<pre>list of : { "address": "Rokach Blvd, Tel Aviv", "dog_id": 101, "place_latitude": 32.0853, "place_longitude": 34.7882, "place_name": "Yarkon Park", "place_type": "parks" }..... Status 200... or Status 204 (not found)</pre>

- GET /api/favorite_places – dog's favorite place by type.

Query Params	dog_id: 1 place_type: "home"
Response	<pre>{ "address": "Rokach Blvd, Tel Aviv", "dog_id": 101, "place_latitude": 32.0853, "place_longitude": 34.7882, "place_name": "Yarkon Park", "place_type": "parks" } Status 200... or Status 204 (not found)</pre>

- PUT /api/favorite_places – Add dog's favorite place.

Json Message	<pre>{ "dog_id": "5", "place_name": "My dog's home", "place_latitude": 102, "place_longitude": 21, "address": "Aza 12", "place_type": "home" }</pre>
Response	Status 200

14. FAQ

- GET /api/faq/questions – Retrieve frequently asked questions.

Json Message	-
Response	<pre>{ "1": "Question 1...", "2": "Question 2...", },</pre> <p style="text-align: center;">Status 200</p>

- GET /api/faq/answer – Get answer for the chosen question.

Query Params	“faq_id”: 5
Response	Answer to the specific question

Collar Embedded to backend:

- PUT /api/dog/collar_data – Update the dog's steps, distance and calories through the day (Receiving data from mobile).
In addition, updates the battery level.

Form	collar_id: 6 battery: 48 steps: 6416
Response	status 200

Message Structure - Collar Embedded to Mobile:

The embedded collar will broadcast multiple types of messages using the BLE protocol (Advertising). The mobile device will listen to these broadcasts.

```
typedef struct {
    uint32_t collarID;
    int steps;
    int batteryLevel;
} CollarMessage;
```

Database Schemes

- users:

Stores user information, including login details and personal data like name, date of birth, and last activity timestamp.

<u>user_id</u>	Integer
email	VARCHAR (100)
password	VARCHAR (255)
name	VARCHAR (100)
date_of_birth	Date
phone_number	VARCHAR (15)
last_activity	Timestamp
logged_in	Boolean

- dogs:

Stores information about each dog, including breed, gender, physical measurements, and activity-related metrics.

<u>dog_id</u>	Integer
name	VARCHAR (50)
gender	VARCHAR (10)
date_of_birth	Date
weight	Float
height	Integer
last_steps	Integer

- users_dogs

Links users and their dogs, establishing a many-to-many relationship between users and dogs.

<u>dog_id</u>	Integer
<u>user_id</u>	Integer

- collars

Stores information about a dog's collar, including BLE and Wi-Fi connection statuses, battery level, and related services.

<u>collar_id</u>	VARCHAR (50)
<u>battery_service_uuid</u>	VARCHAR (50)
<u>battery_level_characteristic_uuid</u>	VARCHAR (50)
<u>step_service_uuid</u>	VARCHAR (50)
<u>step_count_characteristic_uuid</u>	VARCHAR (50)
<u>distance_service_uuid</u>	VARCHAR (50)
<u>distance_characteristic_uuid</u>	VARCHAR (50)
<u>dog_id</u>	Integer
<u>ble_connected</u>	Boolean
<u>wifi_connected</u>	Boolean
<u>battery_level</u>	Integer
<u>last_update</u>	Timestamp

- fitness

Tracks daily fitness statistics for each dog, including steps taken, distance covered, and calories burned.

<u>dog_id</u>	Integer
<u>fitness_date</u>	Date
<u>distance</u>	Float
<u>steps</u>	Integer
<u>calories_burned</u>	Float

- activities

Stores data on a dog's physical activities, such as type, duration, steps, and calories burned.

<u>dog_id</u>	Integer
<u>Activity_id</u>	Integer
<u>activity_type</u>	VARCHAR (30)
<u>start_time</u>	Timestamp
<u>end_time</u>	Timestamp
<u>duration</u>	Interval
<u>steps</u>	Integer
<u>distance</u>	Float
<u>calories_burned</u>	Float

- goal_templates

Provides predefined goal templates for dogs, specifying target values, frequency, and category.

<u>template_id</u>	Integer
<u>dog_id</u>	Integer
<u>target_value</u>	Float
<u>frequency</u>	VARCHAR (50)
<u>category</u>	VARCHAR (50)

- goals

Contains goal-setting information for dogs, tracking progress and target completion for specific categories.

<u>goal_id</u>	Integer
<u>dog_id</u>	Integer
<u>start_date</u>	Date
<u>end_date</u>	Date
<u>current_value</u>	Float
<u>target_value</u>	Float
<u>category</u>	VARCHAR (50)
<u>done</u>	Boolean
<u>is_finished</u>	Boolean
<u>template_id</u>	Integer

- care_info

Contains details about a dog's veterinarian and pension, including their contact information and geographic coordinates.

<u>dog_id</u>	Integer
<u>vet_name</u>	VARCHAR (50)
<u>vet_phone</u>	VARCHAR (15)
<u>vet_latitude</u>	Float
<u>vet_longitude</u>	Float
<u>pension_name</u>	VARCHAR (50)
<u>pension_phone</u>	VARCHAR (15)
<u>pension_latitude</u>	Float
<u>pension_longitude</u>	Float

- medical records

Logs a dog's medical records, including vet visits, address, and detailed descriptions.

<u>record_id</u>	Integer
<u>vet_name</u>	VARCHAR (50)
<u>address</u>	VARCHAR (50)
<u>record_datetime</u>	Timestamp
<u>description</u>	VARCHAR (255)
<u>dog_id</u>	Integer

- nutrition

Records a dog's nutritional data, including food brand, type, daily intake, and additional notes.

<u>dog_id</u>	Integer
<u>food_brand</u>	VARCHAR (100)
<u>food_type</u>	VARCHAR (100)
<u>food_amount_grams</u>	Integer
<u>daily_snacks</u>	Integer
<u>notes</u>	VARCHAR (500)

- vaccinations

Logs vaccination records for each dog, including type, dosage, and future vaccination schedules.

<u>dog_id</u>	Integer
<u>vaccination_id</u>	Integer
<u>vaccination_date</u>	Date
<u>vaccination_type</u>	VARCHAR (100)
<u>dosage</u>	VARCHAR (30)
<u>vet_name</u>	VARCHAR (100)
<u>next_vaccination</u>	Date
<u>notes</u>	VARCHAR (500)

- Places

Stores data about places, including names, addresses, and geographic coordinates, categorized by type.

<u>Place_name</u>	VARCHAR (100)
<u>address</u>	VARCHAR (100)
<u>place_latitude</u>	Float
<u>place_longitude</u>	Float
<u>place_type</u>	VARCHAR (100)

- favorite places

Tracks a dog's favorite places, storing their details and geographic coordinates.

<u>dog_id</u>	Integer
<u>Place_name</u>	VARCHAR (100)
<u>address</u>	VARCHAR (100)
<u>place_latitude</u>	Float
<u>place_longitude</u>	Float
<u>place_type</u>	VARCHAR (100)

- FAQ

Contains frequently asked questions and corresponding answers, assisting users with common queries.

<u>faq_id</u>	Integer
<u>question</u>	VARCHAR (500)
<u>answer</u>	VARCHAR (500)

- notifications

Records a dog's nutritional data, including food brand, type, daily intake, and additional notes.

<u>notification_id</u>	Integer
<u>user_id</u>	Integer
<u>message</u>	VARCHAR (500)
<u>timestamp</u>	Timestamp
<u>Id_read</u>	Boolean
<u>type</u>	VARCHAR (30)

