THE API HUB - VetLocator

This project should be used for frontend development later on.

Rodney Niels Muyanga
cph-rm135@cphbusiness.dk

Michella Stuhr Bech
cph-mb606@cphbusiness.dk

Felicia Favrholdt
cph-ff62@cphbusiness.dk

sp-2-team-2
Datamatiker 3. semester E24
CPH BUSINESS LYNGBY

OVERVIEW

The Vetlocator helps users find available veterinarians, particularly in emergency situations outside regular business hours. Users can also retrieve a list of all veterinarians in their area to make informed decisions about which vet to contact. This API is designed for easy integration into applications that aim to support pet owners in urgent situations.

GITHUB repository link

https://github.com/RodneyMuyanga/SP-2-The-API-Hub

Base URL

insert base URL here

API Overview

insert routes overview here
/api/routes endpoint

Authentication

- API Key: All endpoints require an API key for access.
- Header: Include your API key in the request header as follows:
  Authorization: Bearer {API_KEY}

JWT

For securing our API with JWT (JSON Web Tokens), we implement role-based access control (RBAC) by assigning different roles to users and restricting access to certain endpoints based on these roles:

USER: This role represents regular users, such as pet owners, who will primarily use the API to find veterinarians and emergency services.

Permissions:
- View the list of veterinarians.
- View emergency veterinarian contact information.
- Check availability of veterinarians.
- View specific veterinarian details.

ADMIN: This role represents administrative users, such as clinic managers or system administrators, who need additional privileges, like managing veterinarian data.

Permissions:
- Everything a USER can do.
- Add, update, and delete veterinarian data.
- Manage veterinarian shifts and availability.
- Access user activity logs for troubleshooting or audit purposes. *(future feature)*

VET (Veterinarian): This role represents veterinarians themselves, who may need access to manage their own profile or availability, especially for shifts.
(can be developed later as a frontend development feature - profile image etc.)

Permissions:
- View and manage their own profile info
- Update shift availability.
- View feedback and ratings from users. *(future feature)*

Manual tests
For this project we will hand in a http file for manually testing the JSON output, the output will be pasted in the file for manual review.

Error Handling
Logging via SL4J, Javelin logging and manually creating exceptions.

Backend Development - TechStack and Architecture
Javalin, JPA, DTOs, Java Streams etc. Add more explanation to structure here

Testing
We will be testing the DAO, service layer and endpoints using JUnit, Hamcrest, Rest Assured and test containers.

Deployment
We will use the deployment pipeline to deploy the API to a Digital Ocean Droplet. So GitHub Actions, Docker Hub, Docker Compose files and Watchtower to keep the API Container up to date. We use Caddy to serve the API over HTTPS on a sub-domain.

Link: Add link here

REST API ENDPOINT DOCUMENTATION

ANIMALS

Method	URL	Request Body (JSON)	Response (JSON)	Error (e)	Notes
GET	api/animals		[animal, animal, …] (1)	(e1) (e4)	Retrieves all animals
GET	api/animals/{id}		animal (1)	e1), (e2), (e4)	Retrieves a specific animal by id
GET	api/animals/populate		{ "msg": "Data populated" }	(e4)	Populates animals into DB
POST	api/animals	{ "name": "", "species": "", "age": "", "userId": "" }	{ "msg": "Animal created", "animal": {...}}	(e2), (e4)	Creates a new animal
PUT	api/animals/{id}	{ "name": "", "species": "", "age": "" }	{ "msg": "Animal updated", "animal": {...}}	(e1), (e2), (e4)	Updates animal details
DELETE	api/animals/{id}		{ "msg": "Animal deleted" }	(e1), (e4)	Deletes a specific animal

USER

Method	URL	Request Body (JSON)	Response (JSON)	Error (e)	Notes
GET	api/users		[user, user, …]	(e4)	Retrieves all users
GET	api/users/{id}		user	(e1), (e4)	Retrieves a specific user by id
GET	api/users/{id}/animals		[animal, animal, …] (1)	(e1), (e4)	Retrieves animals of a user
POST	api/users	{ "fullName": "", "email": "", "phone": "" }	{ "msg": "User created", "user": {...}}	(e2), (e4)	Creates a new user
PUT	api/users/{id}	{ "fullName": "", "email": "", "phone": "" }	{ "msg": "User updated", "user": {...}}	(e1), (e2), (e4)	Updates user details
DELETE	api/users/{id}		{ "msg": "User deleted" }	(e1), (e4)	Deletes a specific user and their associated animals

ADMIN

Method	URL	Request Body (JSON)	Response (JSON)	Error (e)	Notes
GET	api/user		[user, user, …]	(e1) (e2)	Retrieves user information for admin use

VETERINARY CLINICS

Method	URL	Request Body (JSON)	Response (JSON)	Error (e)	Notes
GET	/api/clinics		[clinic, clinic, …] (1)	(e4)	Retrieves all clinics
GET	/api/clinics/veterinarians		[veterinarian, veterinarian, …] (2)	(e4)	Retrieves all veterinarians
GET	/api/clinics/{id}		clinic	(e1), (e4)	Retrieves a specific clinic by id
GET	/api/clinics/openinghours		[openingHour, openingHour, …]	(e4)	Retrieves clinic opening hours
GET	/api/clinics/populate		{ "msg": "Clinics populated successfully" }	(e4)	Populates clinic data into DB
POST	/api/clinics/veterinarians/{id}	veterinarian (2) without id	{ "msg": "Veterinarian created", "veterinarian": { ... }}	(e2), (e4)	Creates a new veterinarian
POST	/api/clinics/{id}	clinic (1) without id	{ "msg": "Clinic created", "clinic": { ... }}	(e2), (e4)	Creates a new clinic
POST	/api/clinics/{id}/openinghours		{ "msg": "Opening hours for clinic added successfully"}	(e2), (e4)	Adds opening hours to a clinic
PUT (update)	/api/clinics/{id}	{ "name": "", "city": "" }	{ "msg": "Clinic details updated successfully" }	(e1), (e2), (e4)	Updates clinic details
DELETE	/api/clinics/veterinarians/{id}		{ "msg": "Veterinarian deleted successfully"}	(e1), (e4)	Deletes a specific veterinarian
DELETE	/api/clinics/{id}		{ "msg": "Clinic deleted successfully"}	(e1), (e4)	Deletes a specific clinic

SECURITY

Method	URL	Request Body (JSON)	Response (JSON)	Error (e)	Notes

GET	/api/routes		[route, route, …]	(e4)	Retrieves available routes
GET	/api/protected/user		user	(e3), (e4)	Requires user-level authentication
GET	/api/protected/admin		admin	(e3), (e4)	Requires admin-level authentication
GET	/api/protected/vet		vet	(e3), (e4)	Requires veterinarian-level authentication
GET	/api/auth/test		{ "msg": "Test successful" }	(e4)	To test the endpoint, login not needed
GET	/api/auth/healthcheck		{ "msg": "API is up and running" }	(e4)	Check if API is live
POST	/api/auth/login	{ "email": "", "password": "" }	{ "msg": "Login successful", "token": "..."}	(e2), (e3), (e4)	Adds JWT token logic on successful login
POST	/api/auth/register	{ "fullName": "", "email": "", "password": "" }	{ "msg": "User registered" }	(e2), (e3), (e4)	Registers a new user
POST	/api/auth/user/addrole	{ "userId": "", "role": "" }	{ "msg": "Role added" }	(e2), (e3), (e4)	Adds a role to a user, requires token auth

Request Body and Response Formats

(1) Clinic
{
“Id” : Number
“Name” : String
“Specialization” : [“dog” | ”cat” | “horses” ..]
“Opening Hours” : String
“City” : String
“Veterinarians” : [v1, v2, v3 ..]
}

(2) Veterinarian
{
"id": Number,
"age": Number,
"name": String,
"gender": String [“male” | “Female”],
"email": String (email)
}

(3) Animal
{
“id”: Number
name” : String
species : String [“dog” | “Cat”],
age: String
}

(4) Login
{
"username": "user",
"password": "1234”
}

Errors

These are standard response codes used in web communication to indicate the result of a client's request to a server.

4xx (Client Errors)

- *400 Bad Request*
- *401 Unauthorized*
- *403 Forbidden*
- *404 Not Found*
- *408 Request Timeout*
- *429 Too Many Requests*

5xx (Server Errors)

- *500 Internal Server Error*
- *502 Bad Gateway*
- *503 Service Unavailable*
- *504 Gateway Timeout*

In case of errors, the API returns a JSON response with an appropriate HTTP status code and error message. We use the following error codes in our documentation to secure the correct error response output:

Error Codes

(e1): { status: 404, "msg": "The requested resource could not be found" }

(e2): { status: 400, "msg": "Invalid or missing parameters in the request." }

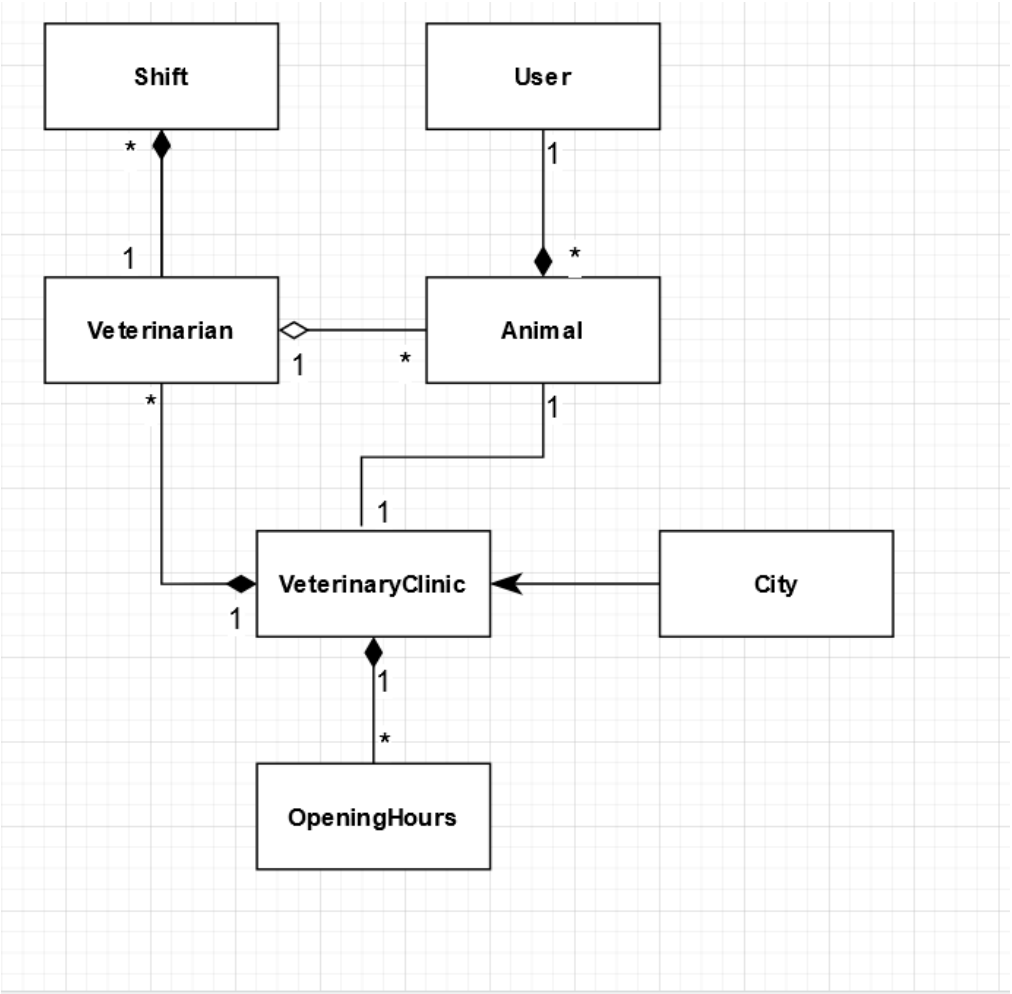
(e3): { status: 401, “msg”: “Authentication failure, typically due to a missing or invalid API key.” }

(e4): { status: 500, “msg”: “An unexpected error occurred on the server.” }

(e5): { status: 502, “msg”: “” } // add more here

DOMAIN MODEL

The domain model describes the relationships between users, animals, and veterinary clinics in the system. The User entity can have multiple Animals, and each Animal is linked to a User through a composition relationship, meaning the Animal is deleted if the User is deleted. Animals are associated with a Veterinary Clinic who can treat them based on specialization. The Veterinary Clinics have opening hours and veterinarians. The veterinarians have shifts which are deleted if the associated veterinarian is deleted. The model ensures that the system can effectively manage emergency veterinary visits and the administration of users and animals.



Entities

User

Attributes: Id, fullName, email, phone, *Set<Animal> animals;*

Relationships: **Composition relation with Animals**: A user can have multiple animals.. If the user is deleted, all associated animals are deleted as well.

Animal

Attributes: id, name, Species (dog, cat, etc.), age, User

Relationships:

Composition relation with User: A animal is always associated with one user.

Association with Veterinarian: An animal can be treated by a veterinarian, but it is not dependent on one veterinarian (as pets may switch veterinarians based on need).

Veterinary Clinic

Attributes: id, name, List<veterinarians>specilizations (which animals they operate on), openinghours, city, veterinarians

Relationships: **Association with Animals**: Veterinary clinics treat animals, but they are not owned or composed by animals.

Veterinarian

Attributes: id, name, specialization (which animals they operate on), clinic

Composition relation with Shift: A veterinarian has an associated on-duty schedule (Shift). If the veterinarian is deleted, their associated shifts are also deleted.

City

Attributes: id, cityName, address, postalCode

Opening Hours

Attributes: id, weekday (enum), localTimeStartTime, localTimeEndTime

Relationships: **Composition relation with Veterinarian**: An opening hour is tied to a specific veterinary clinic.

Shift

Attributes: id, weekday (enum), localTimeStartTime, localTimeEndTime

Relationships: **Composition relation with Veterinarian**: A shift is tied to a specific veterinarian. *(future feature)*