Technical Report - **Product specification**

TrackTails

Course:	IES - Introdução à Engenharia de Software
Date:	Aveiro, 07-10-2024
Students:	113786: Gabriel Silva 114192: Diogo Domingues 114614: Martim Santos 114624: Sebastião Teixeira
Project abstract:	TrackTails is an innovative application that will allow pet owners to get some useful information about their pets, specifically dogs and cats. The application would make data like location, vital signs, sleep patterns and escape alerts available to pet owners easily.

Table of contents:

1 Introduction
2 Product concept
- Vision statement
- Personas and Scenarios
- Product requirements (User stories)

3 Architecture notebook
- Key requirements and constraints
- Post-requirements Observations
- Architectural view
- Module interactions

4 Information perspective
5 References and resources

1 Introduction

TrackTails is a project developed within the scope of the IES course, that would provide pet owners valuable information about their pets well-being and health. The system is tailored primarily for cats and dogs, and would provide real-time location and health tracking, which would allow the system to issue escape alerts and various reports. The data will be presented to the user in a user-friendly interface that would be easy to learn and make it easy to understand the information provided by the device in the pet. This report details the product concept, personas, scenarios, and user requirements that will help develop the TrackTails system.

2 Product concept

Vision statement

This project aims to help pet owners obtain valuable information about their pets' well-being and overall health. Unlike other products TrackTails combines GPS tracking and health monitoring in one product that provides real-time feedback. Our product also includes reports that help identify any sudden routine changes that might affect the pets' well-being or health such as sleep or activity reports, as well as escape alerts if the pet leaves a predetermined area. All of this data will be made available to the owner via an user-friendly and easy to learn graphical interface.

Personas and Scenarios

Before starting the project development, it is essential to go through a brainstorming phase where we define the personas that will represent the different types of software users. After defining the personas, we create usage scenarios for each one, which will allow us to better understand their needs and interactions with the system. From these scenarios, we can then refine a clear and detailed list of requirements that will guide the project development.

Personas

The requirements gathering for TrackTails will be supported by three personas, each representing different types of software users with unique use cases. The defined personas are Maria Costa, João Silva, and António Pereira, who will help guide the application's development according to their specific needs and distinct profiles.

Maria Costa

Maria Costa, a 20-year-old student, is in her second year of the Primary Education program at the University of Aveiro. Originally from Viseu, Maria is studying away from home and has a great love for nature. In her free time, she enjoys outdoor activities, cycling, and spending time with friends. Back in Viseu, she left her two pets: her cat Boneca and her dog Trovão, both playful and adventurous.

Living far from her pets, Maria wants to keep track of their daily activities. Therefore, she needs simple and intuitive software that allows her to easily monitor her four-legged companions, ensuring they are always well while she is away.

João Silva

João Silva is a 35-year-old worker who works as a dentist in his own clinic. He is originally from Santarém and enjoys running and traveling the world with his wife. One day, João went for a run and found a dog that was possibly lost, and at that moment, he didn’t know what to do.

To solve this problem, João thought that there could be a system that would make lost dogs easily identifiable, facilitating their return to their owner or to a place that could receive them.

António Pereira

António Pereira, a 29-year-old professional, manages a kennel. Originally from Vila Real, António has a deep passion for animal welfare and spends his free time playing video games. As a kennel manager, he has realized the need for consistent monitoring of the animals under his care, including tracking their vital signs and other key data to streamline the management of the facility.

António is searching for software that can handle monitoring for the 20 to 30 animals that come through each month. He needs the software to offer the capability to manage his professional needs efficiently at the kennel.

Luis Silva

Luis Silva, a 46-year-old veterinarian, runs his own clinic in Guarda. With a lifelong passion for animals and a love for chess, Luis has built a reputation as a caring and dedicated professional. Over time, he noticed that many pet owners struggle to bring their animals in for regular check-ups, making it harder to catch potential health issues early on.

To address this, Luis is looking for a solution that allows him to remotely monitor his patients' vital signs and behavior, such as sleep patterns and activity levels. The software needs to collect and display accurate health data for the 15-20 animals he sees each month, providing him with the insights necessary for remote diagnoses. Additionally, Luis requires the ability to receive PDF reports from pet owners, so he can review detailed health information before providing feedback or suggesting treatments remotely.

Scenarios

Scenario 1 - Maria Costa

Maria returned to college and, after a day filled with classes, she wanted to check the app to see if Thunder had been taken for his daily walk. So, she opens the app and confirms that Thunder has been walked, which gives her peace of mind.

Scenario 2 - Maria Costa

One of Maria's relatives called her and mentioned that Doll seemed to be ill and was sleeping very little. To ease Maria's worries, her relatives took Doll to the vet, who advised them to keep a close eye on her. Throughout the week, Maria was able to accurately monitor Doll's sleep patterns and vital signs through the app, which helped determine if the pet was improving.

Scenario 3 - João Silva

João went for a run to the city center and came across a dog that appeared to be lost. He approached the dog and noticed that its collar had a device with a QR code that said, "If I'm lost, scan me." He scanned the code, and an interface opened, displaying all the information he needed to follow in order to return the dog to its owner.

Scenario 4 - Luis Silva

Luis Silva is at the veterinary clinic where he will treat Dona Lucinda's cat who has been eating little. To be able to make the diagnosis, Luis Silva needs to ask the owner a set of questions, leaving him dependent on the uncertain precision that his answers may present.

Through the proposed system, Luis Silva is able to quickly access a vast set of data, more or less precise, which allows him to better carry out the diagnosis.

Scenario 5 - António Pereira

António Pereira, manager of a kennel, is reviewing the list of animals that recently arrived. With 20 to 30 animals passing through the kennel each month, he is starting to have difficulty keeping track of them.

By finding a suitable tool, António will be able to improve animal care and focus on ensuring that they have a comfortable and safe stay in the kennel.

Product requirements (User stories)

Epics	**User Stories**
Epic #1 Animal-system association	As a pet owner, I want to be able to access data about my pet remotely so I can track them wherever I am.
	As a pet owner, I want to be able to track multiple animals in a single interface, to make management easier when the number of animals is high.
Epic #2 Track pet states in real-time	As a pet owner, I want to be able to locate my pet so that I can easily find it if it gets lost.
	As a pet owner, I want to track the physical activity of my pet, so I can better understand my pet's habits and be able to notice changes in his behavior.
	As a pet owner, I want to monitor my pet's sleep

	patterns and vital signs so I can see if the pet is showing any symptoms indicative of health problems or improvements in them
	As a pet owner, I want to be alerted whenever something abnormal happens (vital signs are very altered or abrupt movement indicates a fall or being run over), so I can help you as quickly as possible.
Epic #3 Share basic owner information	As someone who found a lost animal, I want to be able to obtain information about the animal and contact details about the owner, in order to help it return to its owner.
Epic #4 Analyze historical data	As a pet owner, I want to be able to consult the route taken by the pet over a certain period of time so I can have some control over the places it travels.
	As a pet owner, I want to be able to consult historical data on physical activity, sleep patterns and vital signs of the animal, in order to detect anomalies over time.
Epic #5 Share Pet Information with Veterinarian	As a veterinarian, I want to be able to have access to the animal's data in order to help me carry out diagnosis.

3 Architecture notebook

Key requirements and constraints

To begin with, the system must be scalable as the number of registered users and animals increases. To make an estimation of the volume of messages sent by animals' devices, we will start by identifying which metrics will be collected:

By the pets' devices:

- Geographic Location (coordinates)
- Heart Rate (beats per minute)
- Respiratory Rate (breaths per minute)
- Movement (m/s)
- Device Battery (%)

By user input:

- Weight (kg)
- Height (cm)

Computed:

- Sleep patterns (hrs)

In order for it to be possible to issue alerts based on the measurements taken by the devices, they must be sent at a high frequency (at least one measurement every 3 seconds). To minimize the number of messages sent, making messages more efficient, measured values can be sent in batches, in a single message, instead of sending one message after each measurement. This leads us to conclude that each device will send a message at least once every 3 seconds. Thus, the minimum number of messages per second will be directly proportional to the number of devices in operation. For example, if there are 100.000 devices in operation, the system must be capable of processing at least 35.000 messages per second. To further reduce this value, we could set up devices not to send messages if there is not relevant variation in any of the metrics. However, this strategy would imply a series of precautions, such as the need to guarantee a minimum frequency of messages so that it is possible to detect the unavailability of a device, as well as the need to interpolate data during their processing to avoid errors due to their temporal inconsistency.

To be able to deal with high traffic, we can adopt a diverse set of strategies:

1. The system must be able to buffer messages, so that it can handle load spikes;
2. The system must be able to start new instances on demand to collect data,

without interrupting/disturbing the operation of the remaining system components.

To meet these 2 requirements, a Message Queue can be used (see Architectural Overview), which not only works as a buffer, but also removes the coupling between the devices and the *Data Collectors*.

Since the main feature of the system is, precisely, the location of pets when they are lost, it is essential that the system can be used while the user is on the move. Therefore support for mobile platforms is a priority. Taking into account that there is no specific requirement that requires the use of the desktop format and given the limited time to complete the project, there will be no guarantee of support for desktop formats. Therefore, building a *Progressive Web APP* (PWA) becomes an excellent option, meeting all the requirements mentioned above. Furthermore, in a *PWA* the client only needs to communicate with the web server on the first access, with the respective browser being responsible for *caching* the interface. In this way, it is estimated that the load on the web server will be reduced enough so that more than one running instance is not necessary.

Regarding data processing and ensuring user access to the information, we realized that there are different components of the system that have specific well defined functions and each with its own requirements. For example, report generation is not a vital project functionality, unlike, for example, animal registration and management. While generating reports is a process in which *delays* of minutes or even a few hours are tolerable, when accessing animal data and histories a delay of even 5 seconds is not tolerable. Furthermore, it is not acceptable for a problem in a non-core functionality to compromise the system as a whole. Thus, we concluded that the adoption of an architecture based in microservices is very advantageous, as it provides the necessary level of decoupling to allow scaling different services at varying rates while also isolating errors to the specific component affected. In order for the client to be able to access the various services, it makes sense to use a *Reverse Proxy*, which can also perform *load balancing*, thus promoting greater scalability.

As for the data persistence layer, we could just use a relational database. However, an approach that utilizes two databases can be advantageous as the relational database is well-suited for tasks like storing user information and their associated animals, while a time-series database is particularly useful to store the data collected from the animals' devices. Each of these records should be associated with a *timestamp*, and the system must be capable of maintaining the entire history of the data. By using a dedicated time-series database, it is also estimated that write and read will be faster and more efficient, especially when dealing with real-time data. With that said, it is still necessary to estimate the storage capacity to estimate the storage capacity required to keep the system fully operational. The size of the user and animal records will be more or less proportional to the number of registered users. However, the volume of data collected by the devices will grow not only in proportion to the number of registered users but also over time. If we have 1000 registered animals collecting the aforementioned metrics every 3 seconds, we will see an increase of more than 1GB. As more animals are registered, the need for storage will grow even faster.

In short, these are some key requirements and system constraints that have a significant bearing on the architecture:

- The ability to quickly scale the real-time data collection capacity.
- Failures in non-critical components must not impact critical components.
- While certain functionalities, such as report generation, do not require significant concern regarding response times and processing capacity, others, such as accessing animal data via API, do not tolerate delays, even if they are not very large.
- The system must provide prioritized support for mobile devices.
- It is necessary to restrict unauthorized access to the data, and user identification must always be required.
- The system operation requires an Internet connection with a bandwidth that allows receiving at least *1Mbps* of data for every 1000 registered animals.
- The storage capacity must be easily scalable due to the continuous growth of the stored data volume.
- To prevent data loss, redundancy and data recovery mechanisms must be implemented.

Post-requirements Observations

While the diagrams below have been updated in every iteration whenever there was a need to adapt or modify the project architecture, the points above remained static, so that we can make a comparative analysis of what were the initially defined requirements with what is actually being implemented.

So, we reserve this section for some final considerations regarding the needs that were discovered throughout the development of the project, leading to several adaptations to the architecture.

Taking into account the type of architecture chosen for the project, a set of problems and challenges associated with the microservices paradigm emerged, which we have resolved with different types of solutions.

1. Regarding authentication, as previously decided, it was implemented by bringing JWT token validation to the gateway side. This made it possible to decouple the microservices from the user service (service responsible for authentication). The

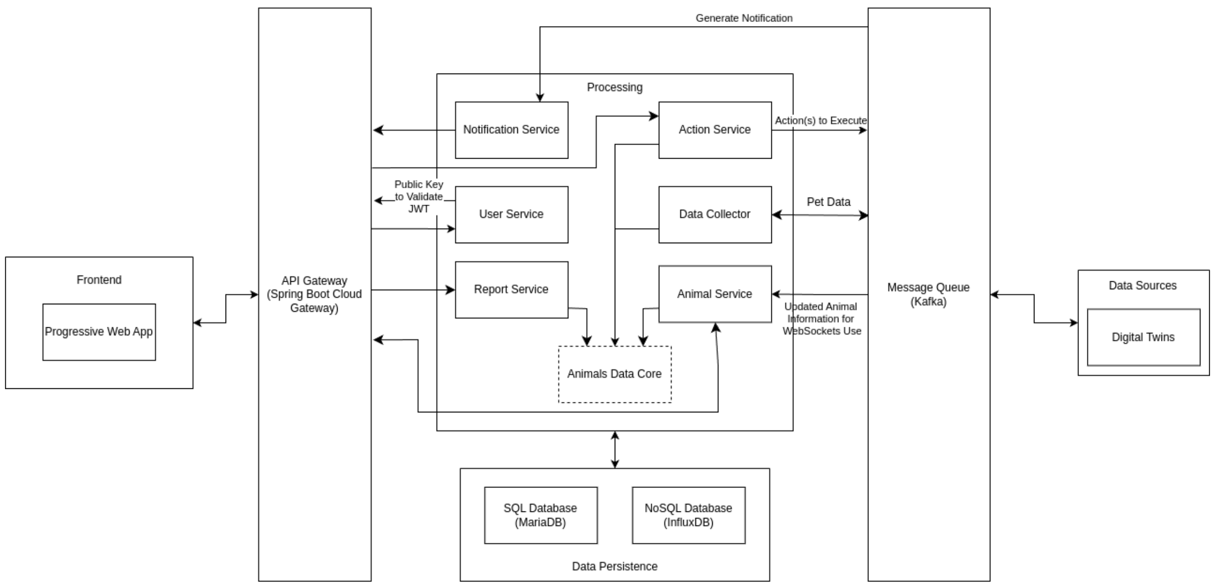
solution found to implement this architecture was to pass the JWT via JWKS to transmit the public keys corresponding to those used to sign the jwt, from the user service to the gateway.

2. One of the points that had not been planned at the beginning of the construction of the architecture requirements was the transmission of animal data in real time to the client interface. This does not constitute any of the user stories colors, however, we consider it an important feature to implement to improve the user experience. Here some problems arose. It is necessary to preserve the system's scalability capacity and, to this end, the data collector cannot be responsible for websockets because, with multiple data collector instances, not all of them will receive all messages. Therefore, it would not be possible to establish many-to-many communication between data collectors and websocket clients. Regardless of which data sink consumes a given message, all websockets clients that have subscribed to that information must receive that information in real time. So, the solution involves bringing websockets connections to the animal service side, making it possible to create a many-to-many means of communication. To achieve this, the solution is to use a centralized communication system between the data collector and the animal. We chose to use Kafka to communicate between these microservices. We recognize that this situation is not perfect, as we are creating asynchronism, which contradicts the concept of real-time data. Using http requests would not work as communication in this protocol is one-to-one and not many-to-many.

3. As in the previous point, there was a need for the data collector to be able to communicate with the notification service, so that a notification is triggered whenever the collected data justifies the creation of a certain event. So again, kafka which was used to pass the information between the data collector and the notification service

4. In relation to the action service, we define two different types of actions of different natures. The first consists of the device generating a sound. This is an event that happens over a short period. In this case, we do not need to store the state of this action. The other consists of flashing a light on the device. The light will continue flashing until the user explicitly indicates that they want to stop. In this case, we have to save the state of the action so that the client can know whether the action is being executed or not.

Architectural view



As mentioned above our project is based on a microservice architecture, this way we can achieve low coupling, high cohesion which helps us fix bugs and implement new features faster and independently from other services or layers, on the diagram we simply group components in layers in order to make it simpler to understand which components are responsible for what.

We have 6 main layers that are:

- Frontend -> This layer is responsible for all the user interaction.
- API Gateway -> This layer is responsible for handling all the interactions from the frontend and the backend, including verifying authentication through JWT (Java Web Tokens).
- Processing -> This layer is responsible for receiving and processing all the data received either from the user or the pet, it is also responsible for storing it as well as aggregating and feeding it to the API Gateway if necessary.
- Data Persistence -> This layer is responsible for storing all the data from the users and pets.
- Message Queue -> This layer is responsible for streaming the data from the pet device to the processing layer, as well as some inter-services communication.
- Data Sources -> This layer is responsible for generating the data to be processed and stored by the system.

In the following table we will take a deeper dive into each component;

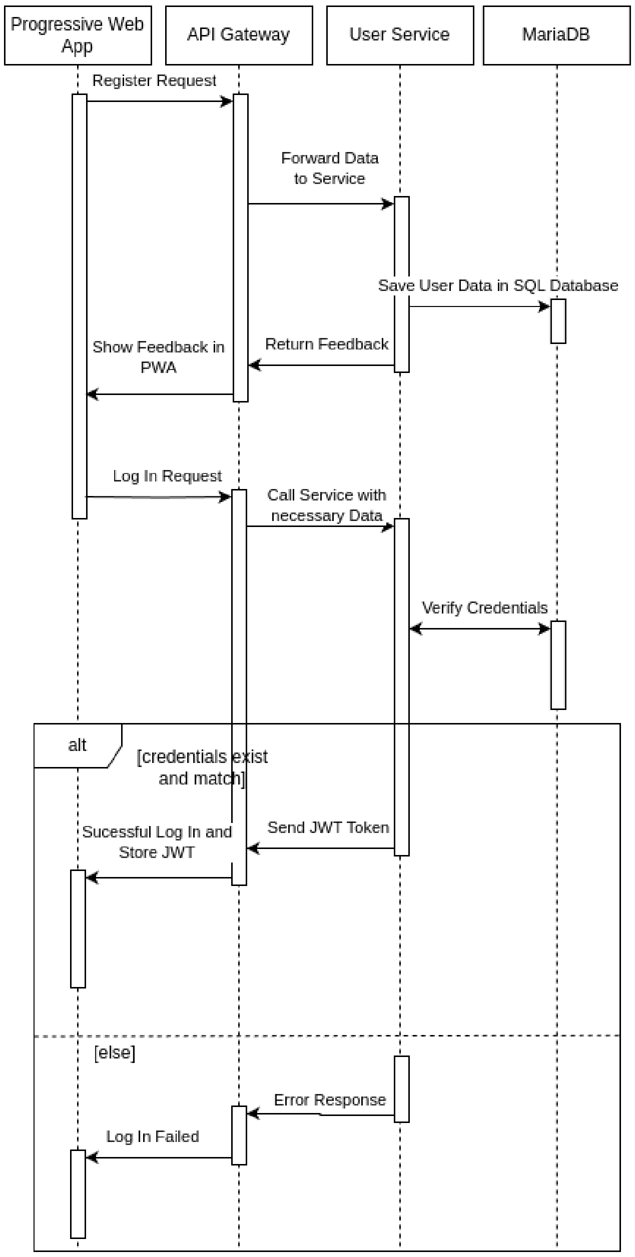
Layer	Component	Description
Frontend	Progressive Web App	Allows users to interact with our system, this includes both regular users and pet finders.
API Gateway	-	This layer will handle all the interactions between the frontend and the processing or data layers, and authentication verification.
Processing	User Service	This component is responsible for verifying credentials and creating new users.
	Animal Service	This service is responsible for interacting with the databases as well as gathering and processing all information about a user and his pets to present to the frontend.
	Report Service	This service is responsible for gathering information about a pet in a certain timeline and making a PDF file with the information gathered.
	Action Service	This component is responsible for receiving actions to execute on the device present in the pet.
	Data Collector	This component is responsible for receiving data from the pet device as well as processing and storing that data.
	Notification Service	If the component responsible for processing real-time data detects an anomaly it will call this service to send a notification to the user.
	Animals Data Core	This component is a shared library with structures used by services that need to query data related to animals (static or dynamic, i.e, user or device provided).
Data Persistence	SQL Database (MariaDB)	This database will be responsible for keeping the users data and pets associated with them, hence why we chose a relational database.
	NOSQL Database (InfluxDB)	This database will be responsible for keeping all the pet data collected by the device in the pet.
Message Queue	-	This layer is responsible for streaming the data collected by the pet device to the real-time processing component or some

		inter-service messages.
Data Source	Digital Twins	This component will simulate the pet device, feeding data to the message queue layer.

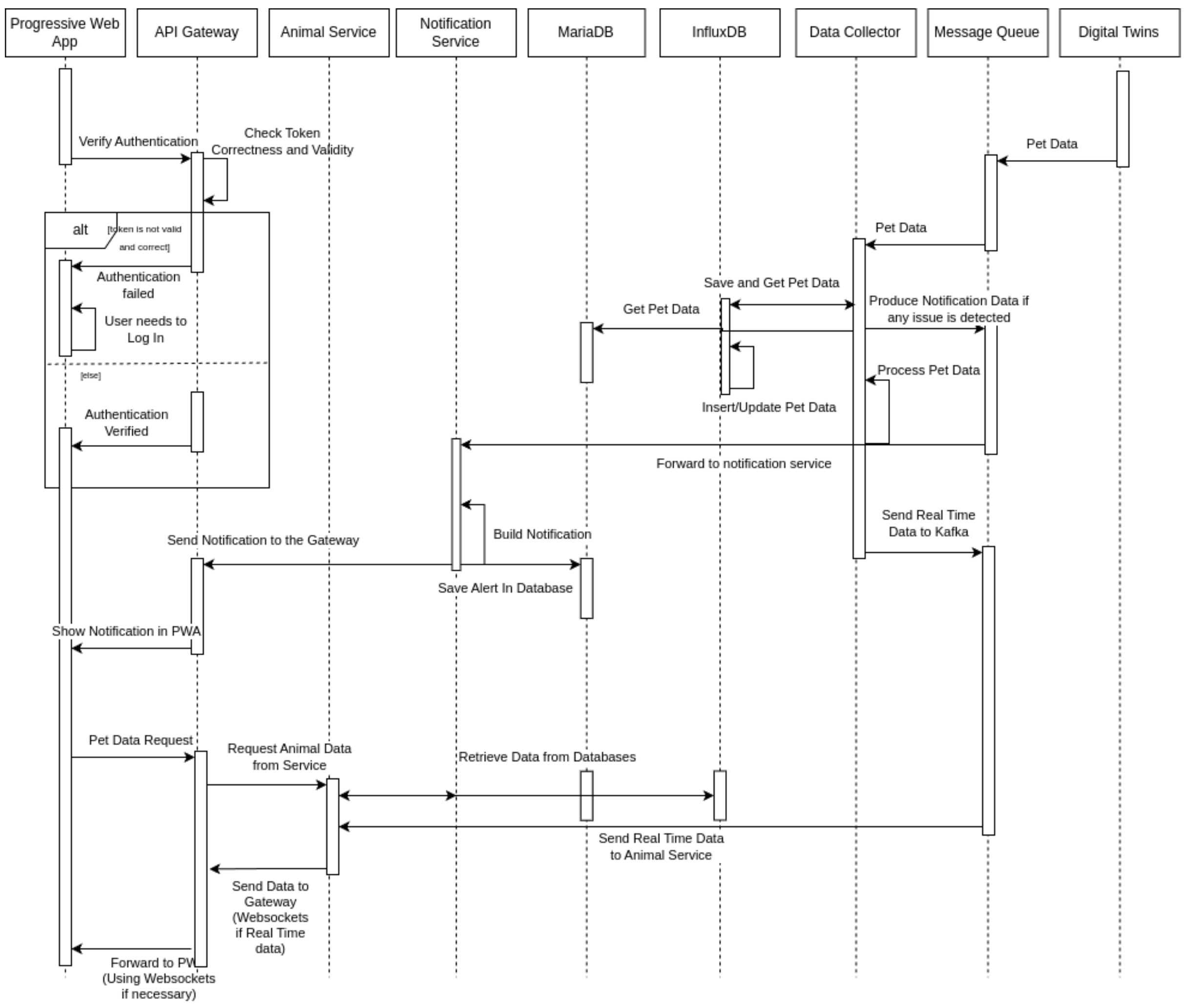
Module interactions

Most of the interactions between the modules are represented by the following sequence diagrams:

Below each diagram there will be a brief explanation of the flow:

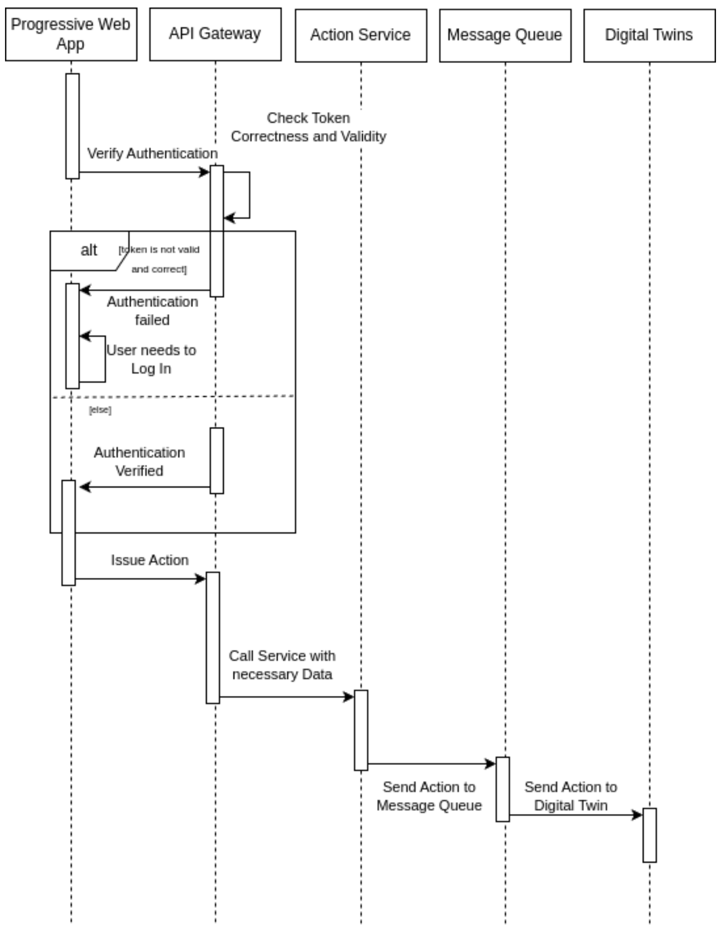


- Register and Log In -> Any user interaction either starts with a Log In if an account already exists or with a register if it doesn't. Registering starts with the *PWA* sending a *Register Request* to the Gateway that will forward the data to the correct service (User Service) and then if everything is correct the credentials will be saved in the database, and a feedback will be returned. When a user is already registered the *PWA* will send a *Login Request* instead simply verifying the credentials and if they are correct it will generate a JWT that will be stored in *Local Storage* and used by the *Gateway* to control access to all the features in the *PWA*, if they are incorrect it will simply fail a *Log In* and the user must try again or *Register*.


Progressive Web App
API Gateway
Animal Service
Notification Service
MariaDB
InfluxDB
Data Collector
Message Queue
Digital Twins
Verify Authentication
Check Token Correctness and Validity
Pet Data
alt
[token is not valid and correct]
Pet Data
Authentication failed
Save and Get Pet Data
Get Pet Data
Produce Notification Data if any issue is detected
User needs to Log In
[else]
Process Pet Data
Insert/Update Pet Data
Authentication Verified
Forward to notification service
Send Real Time Data to Kafka
Build Notification
Send Notification to the Gateway
Save Alert In Database
Show Notification in PWA
Pet Data Request
Request Animal Data from Service
Retrieve Data from Databases
Send Real Time Data to Animal Service
Send Data to Gateway (Websockets if Real Time data)
Forward to PWA (Using Websockets if necessary)

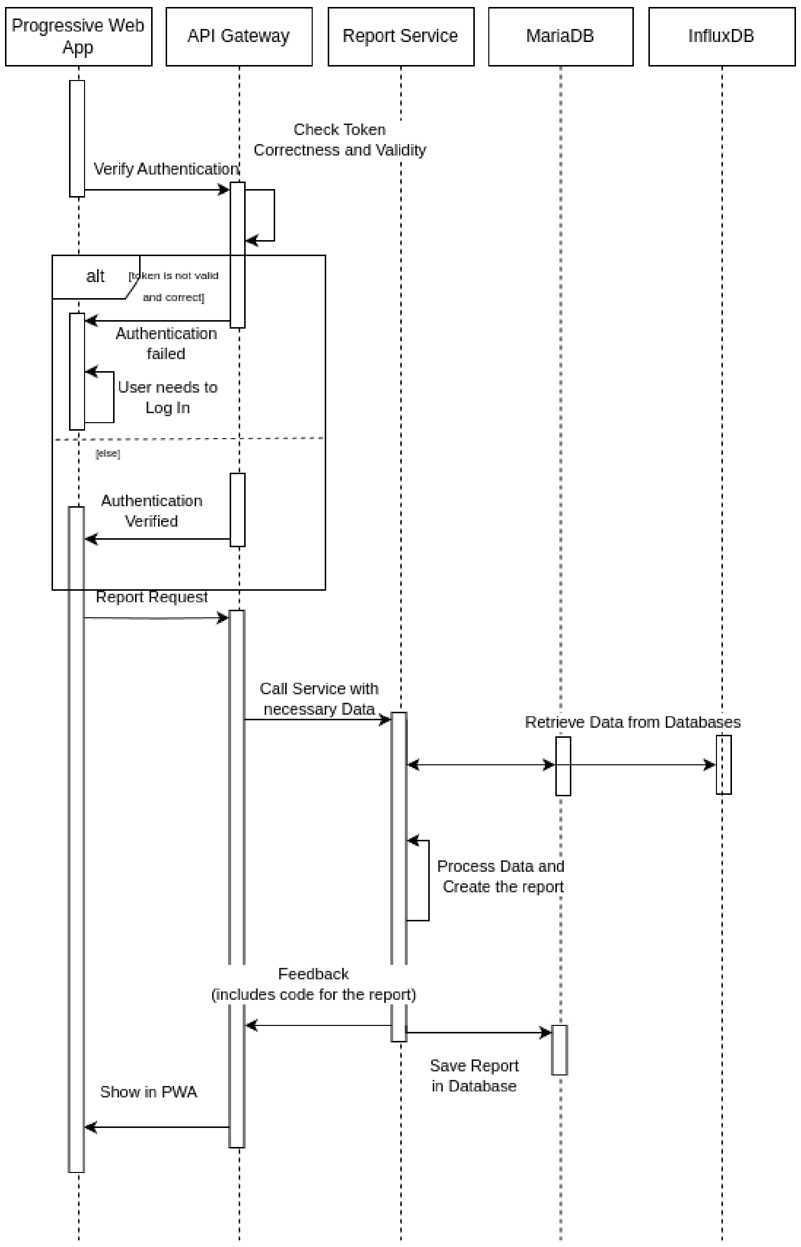
- Pet Data and Notifications -> This diagram can be subdivided into 2 fluxes, the first one is getting pet data and storing it, this starts with the pet device sending all collected data to the message queueing service which will stream the messages containing said data to the real-time data processing service (*Data Collector*) which will process and store the data in the appropriate database (*InfluxDB*), however if a value collected is outside the defined safe range (bpm, too much speed, outside desired location, etc) a

String containing all necessary information to create a notification will be send to the message queueing service which will forward it to a consumer in the *Notification Service* where a notification will be created and sent to the user, this can be seen mostly in the right and middle of the diagram. The second flux is obtaining pet data, this will start with verifying user authentication (i.e, checking token correctness and validity) and either log the user in or ask for the user to log himself back in. Once we have a valid session the *PWA* will request Pet Data, only for the logged in user, and then we will ask the appropriate service for this data (Animal Service) which will retrieve pet data from the SQL database (name, age, etc) and data either from the NoSQL database (heart bpm, location, speed, etc) or real-time data received from *WebSockets*. That data will then be sent to the *Gateway* which will then forward it to the *PWA* to be shown. This is represented on the lower half of the diagram.



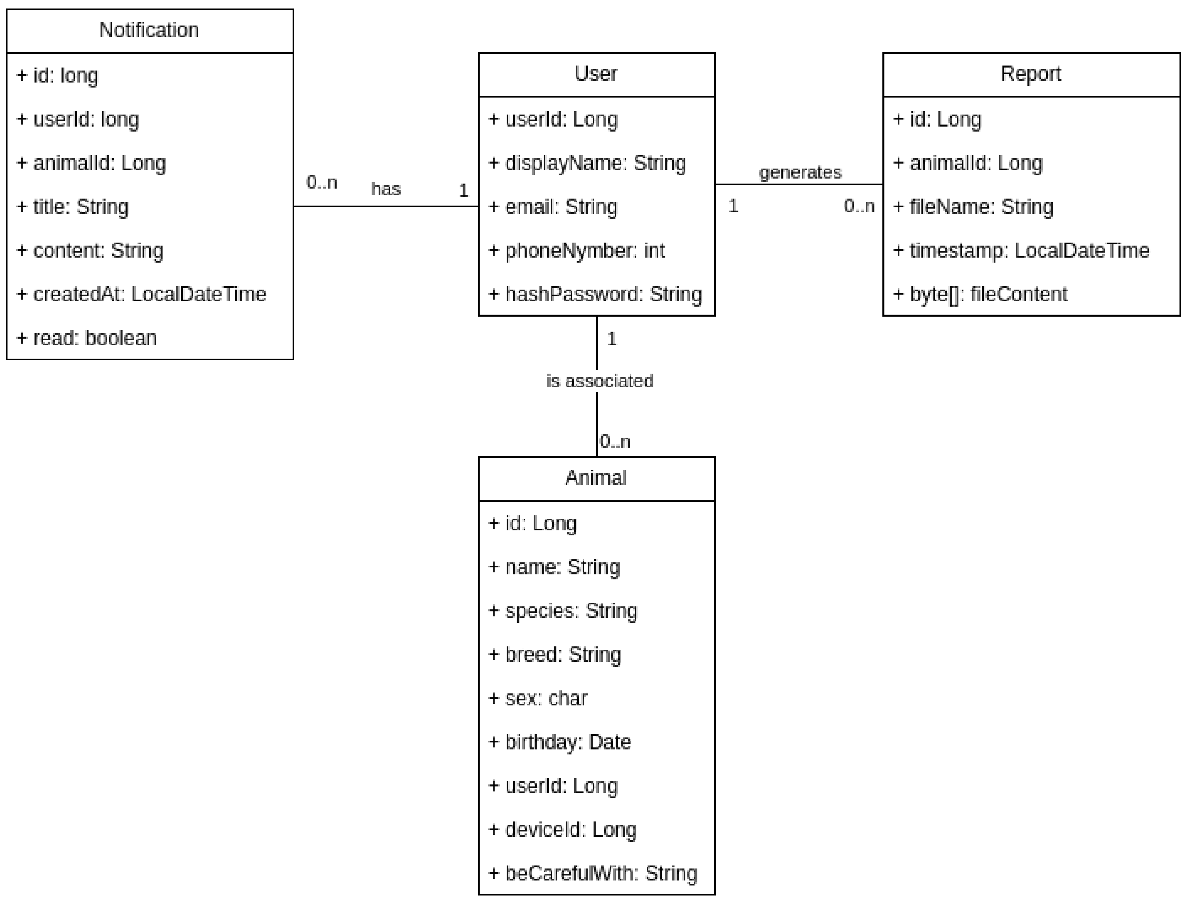
- Issue an Action -> Similarly to the *Pet Data and Notifications* flow this sequence will start by the authentication verification with similar consequences if failed. If successful though the *PWA* will send an action request to the *Gateway* which will forward it to the correct service (*Action*

Service) which will then send the action to be made to the *Message Queue* that will finally send the *Action* to the Digital Twin (representation of real device).


Progressive Web App
API Gateway
Report Service
MariaDB
InfluxDB
Check Token Correctness and Validity
Verify Authentication
alt
[token is not valid and correct]
Authentication failed
User needs to Log In
[else]
Authentication Verified
Report Request
Call Service with necessary Data
Retrieve Data from Databases
Process Data and Create the report
Feedback (includes code for the report)
Save Report in Database
Show in PWA

- Request a Report -> As always this flow will start with authentication validation. Once authentication is verified *PWA* will make a request to the *Gateway* which will call the adequate service with the necessary data (animal id, time period, etc), then the service will start to gather the data and join it in a report. Once the report is ready it will be stored in the *SQL Database.* Finally, we just have to send feedback to the *Gateway* and then the finished report.

4 Information perspective



The **'User'** class is linked with zero or more **'Animal'** entities, indicating a one-to-many relationship where one user can be associated with several pets, on the PWA the user will be able to see all the data associated with the pet. A similar situation will also happen for **'Notification'** and **'Report'**, the **'User'** will have zero or more notifications and zero or more generated reports both of which can be seen and consulted in the *PWA*, also indicating a one-to-many relationship.

5 References and resources

- https://kafka.apache.org/
- https://www.influxdata.com/
- https://mariadb.org/
- https://spring.io/projects/spring-cloud-gateway
- https://spring.io/
- https://aws.amazon.com/pt/compare/the-difference-between-rabbitmq-and-kafka/
- https://tractive.com/pt/