E-AGLE TRENTO RACING TEAM

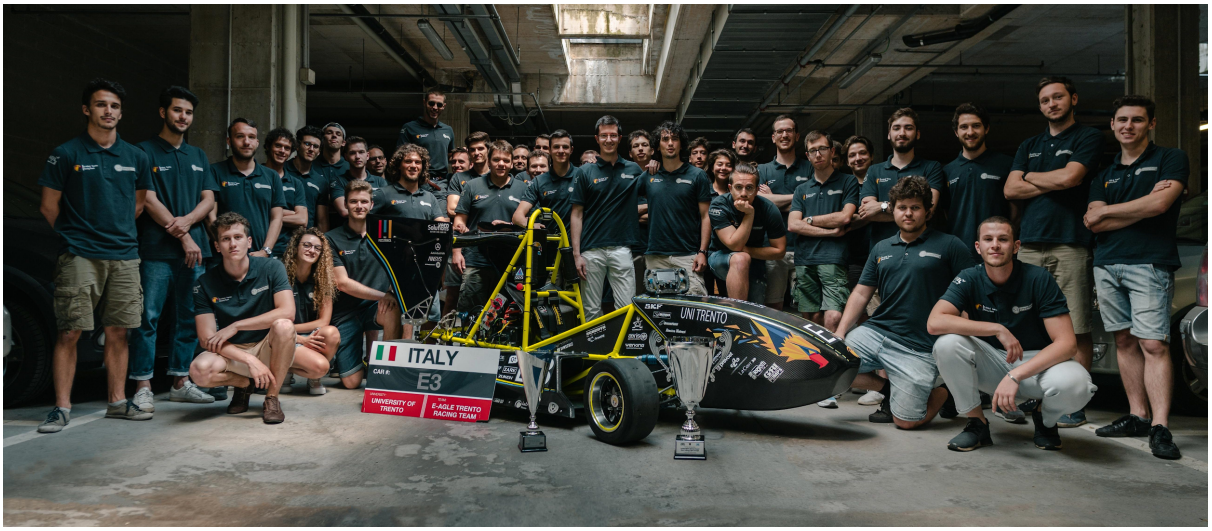
TELEMETRY WEB-APP

Luca Martinelli

2019-2021

Table of contents

Introduction



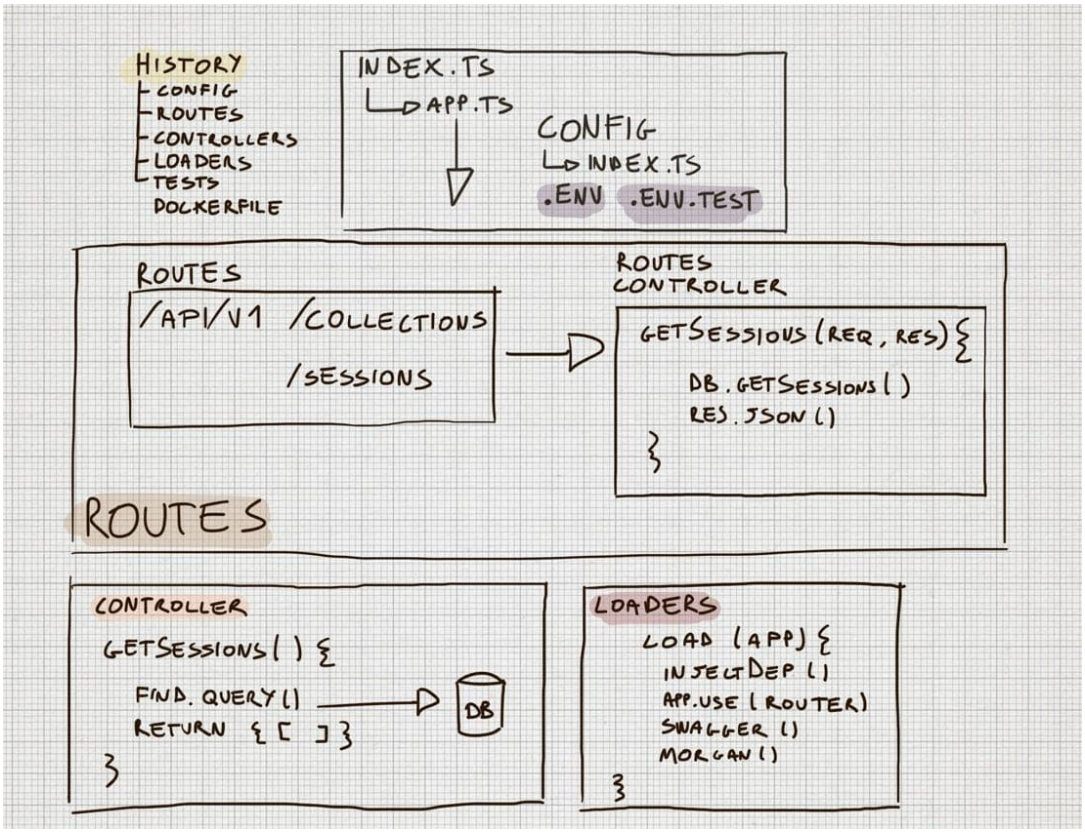
E-Agle Trento Racing Team is a group of students from the University of Trento. The team's purpose is to design, build and test an electric single-seater car. In 2018, E-Agle TRT has the ambition to develop a custom solution to improve the human-machine interface during tests and competition. The solution is a custom device named *Chimera Steering Wheel*, a customizable device never seen in the Formula Student competition.

One of the issues the team encountered while using the chimera steering wheel is the lack of a desktop solution for managing and displaying the status of the vehicle. The steering wheel does not allow to save can-bus messages of the single-seater and analyze them later except through log can, an unreliable Linux tool. With the increased awareness of the team from the dynamic point of view of the car, the group in charge of the ECU (*Electronic Control Unit*) and Control (*Tractive System*), now need data in order to validate the model of the vehicle. From this set of considerations, it was decided to start prototyping and developing an application to solve these issues.

The Telemetry application (on Github: eagletrt-api) is the result of the continuous research of the telemetry group from 2018 until 2021. The result is a maintainable application to serve the telemetry web app used by the member of the team. The main focus is to serve the web app by giving the data to render the charts of the vehicle during the tests. Each user should have the possibility to see a customizable page with charts and configuration of the vehicle.

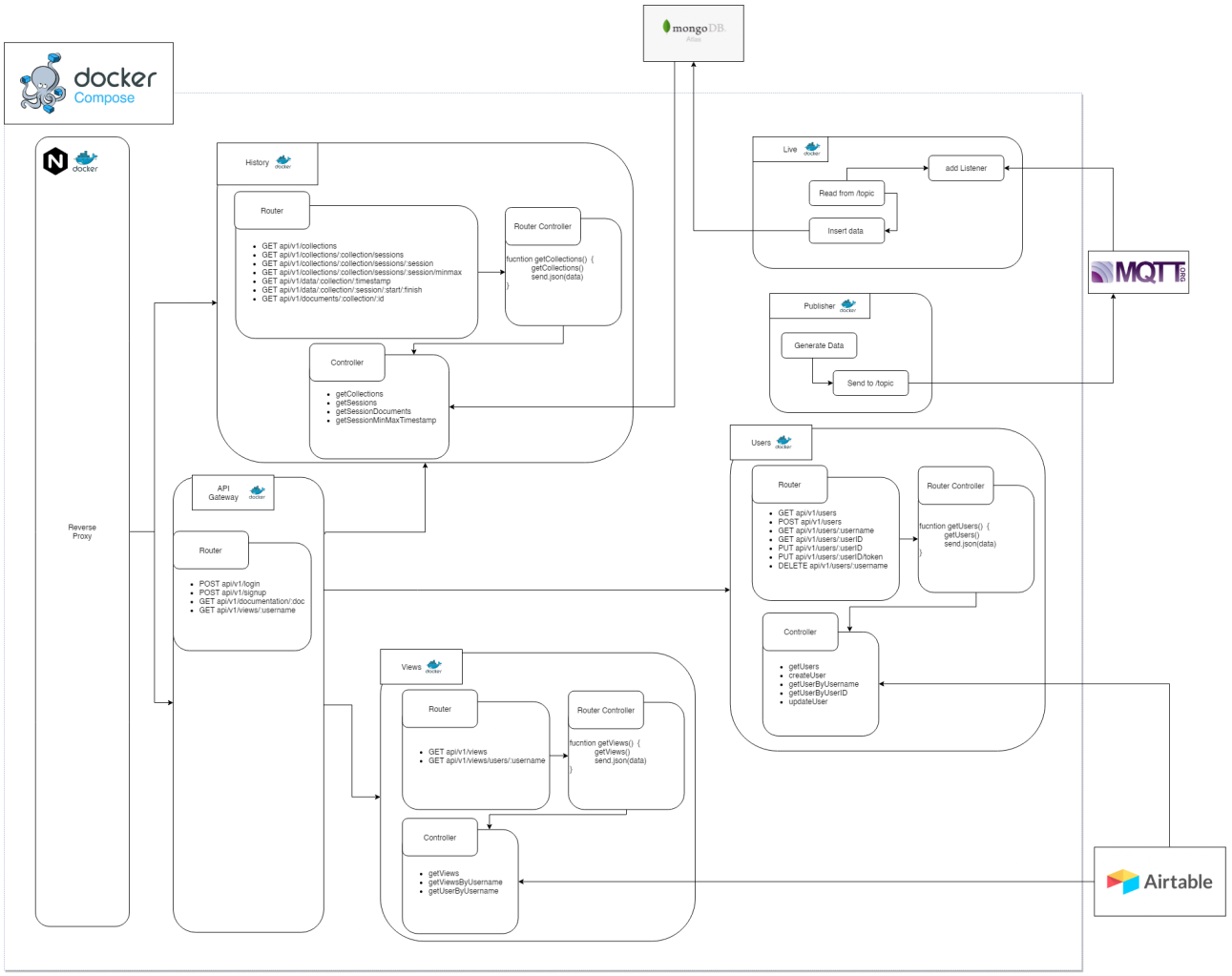
Technologies

This project consists of an express rest API server written in Typescript. It's a cloud-based application. It's built with different microservices developed with NodeJs, Docker, NGINX, MongoDB, and MQTT. The project is built starting with a template based on the different packages needed by the team members. The description of the directories is below and consists of five main folders:

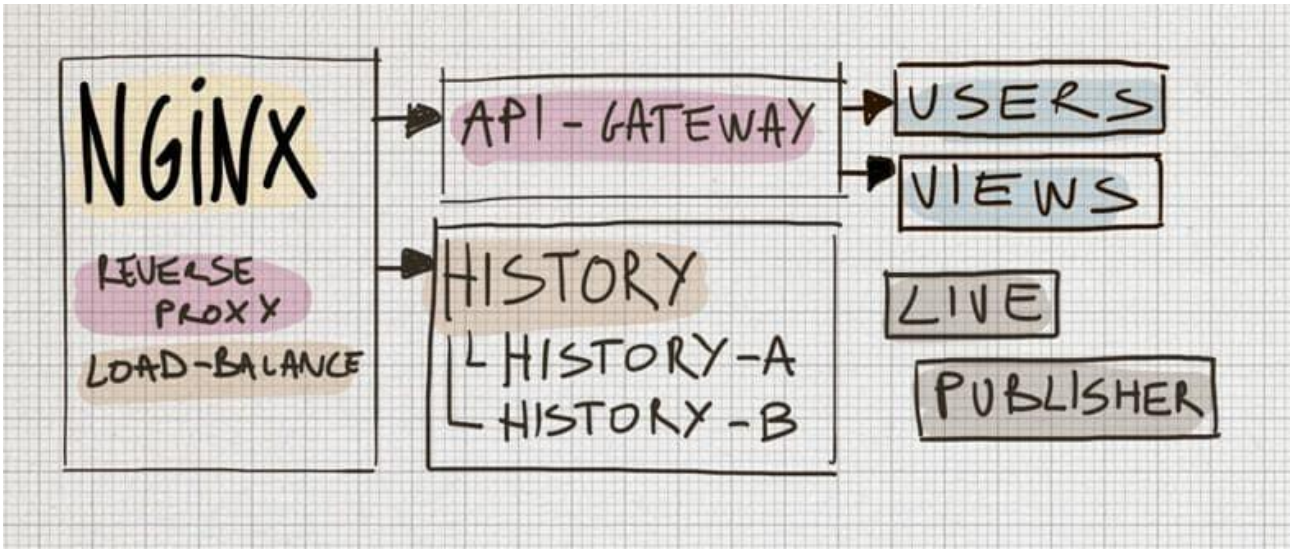


- **loaders**: to inject dependencies, routes, and all sorts of configuration.
- **routes**: where the routes of the API are described with the swagger code. It also has a *controller.ts* to handle requests.
- **controllers**: it's where the magic is, all the sauce of the project. This folder contains all the functions used to retrieve data from the database or to interact with external API.
- **tests**: it runs the application as a different process in order to test Newman's requests.
- **config**: with the index.ts it's possible to load all the *.env* variables, it's based on the environment (test or production)

Microservices Overview

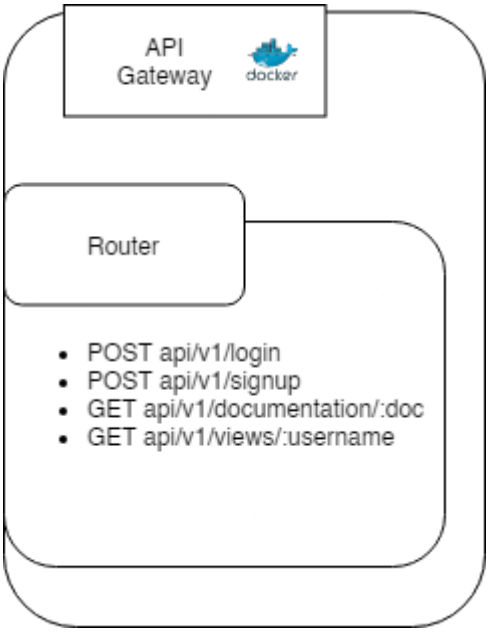


Nginx



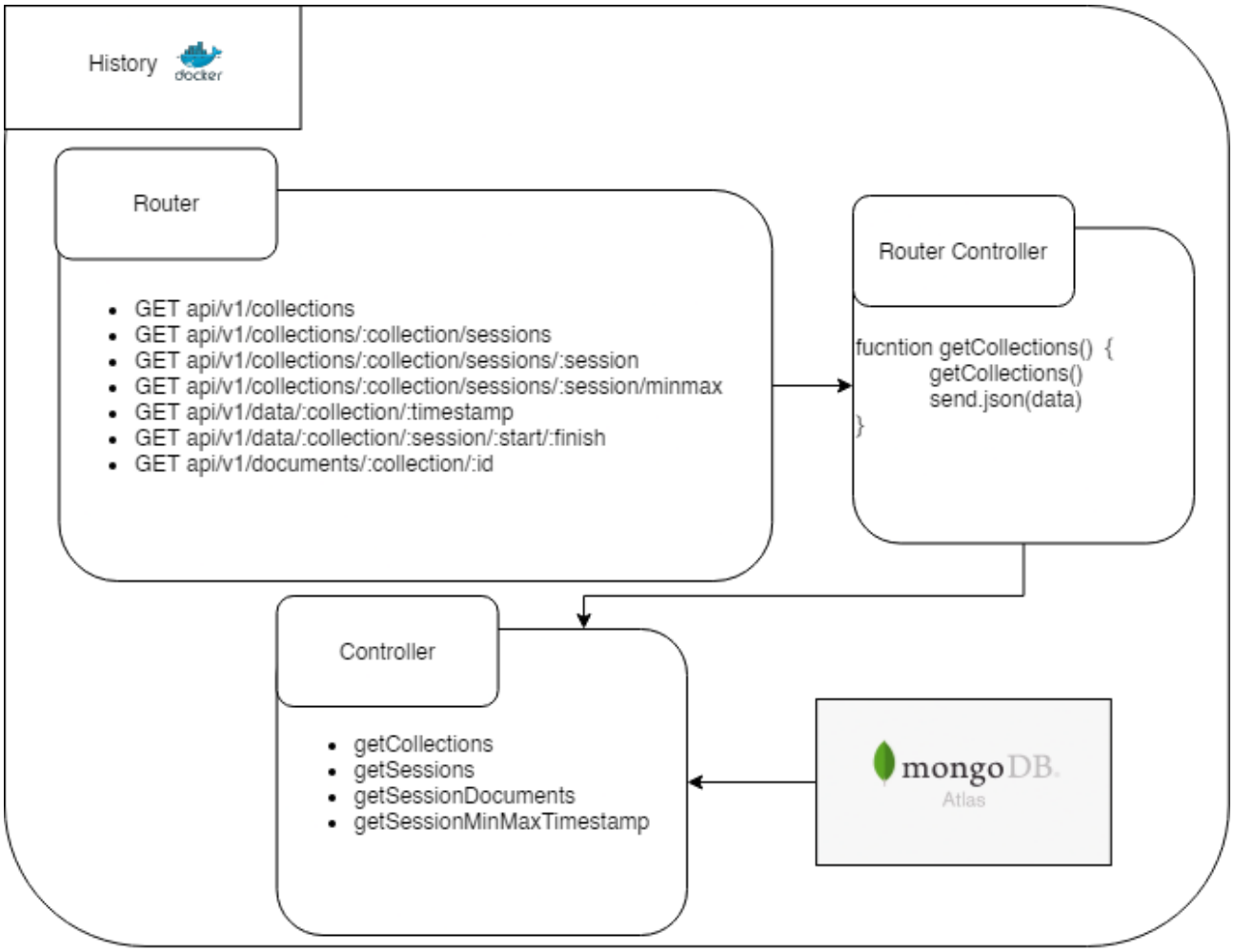
Nginx is an open-source web server that also serves as a reverse proxy and HTTP load balancer. As a reverse proxy, it sits in front of our microservices and API Gateway. When a browser makes an HTTP request, the request first goes to the reverse proxy, by checking the API Key for the user, which then sends the request to the appropriate microservice. It is straightforward and lightweight, but in the future, the authentication process will be handled differently.

API Gateway



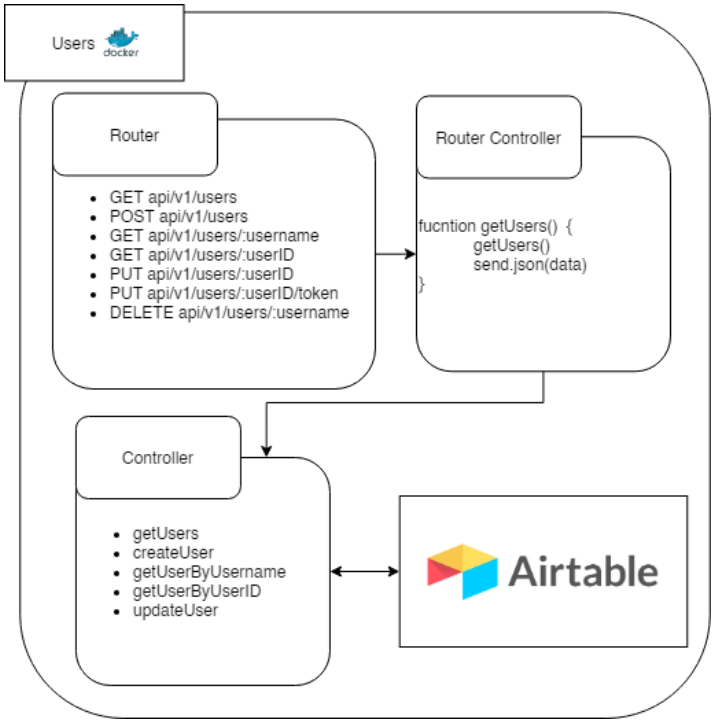
The ***API Gateway*** is responsible for request routing, composition, and protocol translation. Requests from clients first go through the API Gateway. It then routes requests to the appropriate microservice. The API Gateway will often handle a request by invoking multiple microservices and aggregating the results.

History



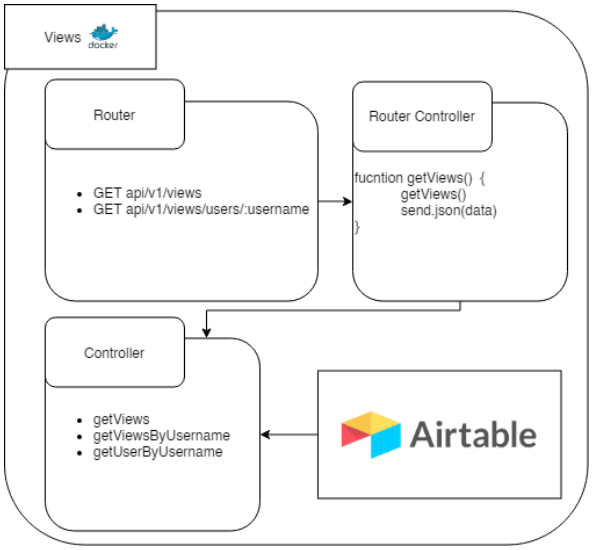
The ***History*** microservice is used to serve the web app with all the documents, data of the vehicle. This microservice consists of an adapter layer to get the information from a schema-less database (MongoDB on Atlas and inside the University of Trento).

Users



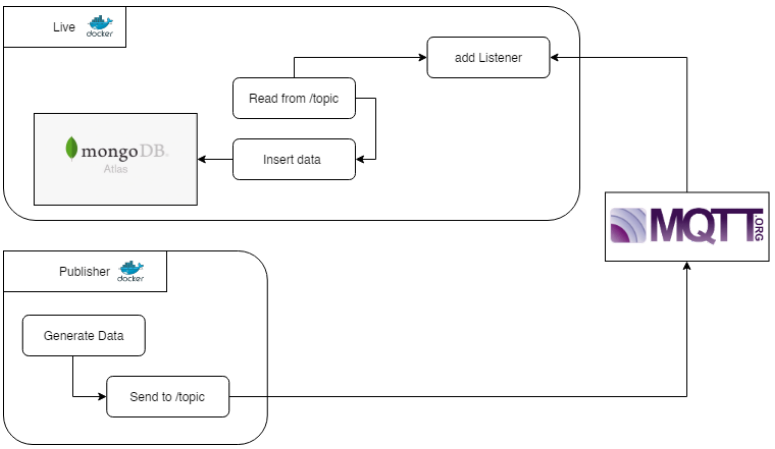
The ***Airtable*** microservice is used to serve the web app with all the information about the user and their personalized views based on their role. This microservice consists of an adapter layer to get the information from Airtable.

Views



The ***Views*** microservice is used to serve the web app with all the information about the personalized views based on the user role. This microservice consists of an adapter layer to get the information from Airtable.

Live

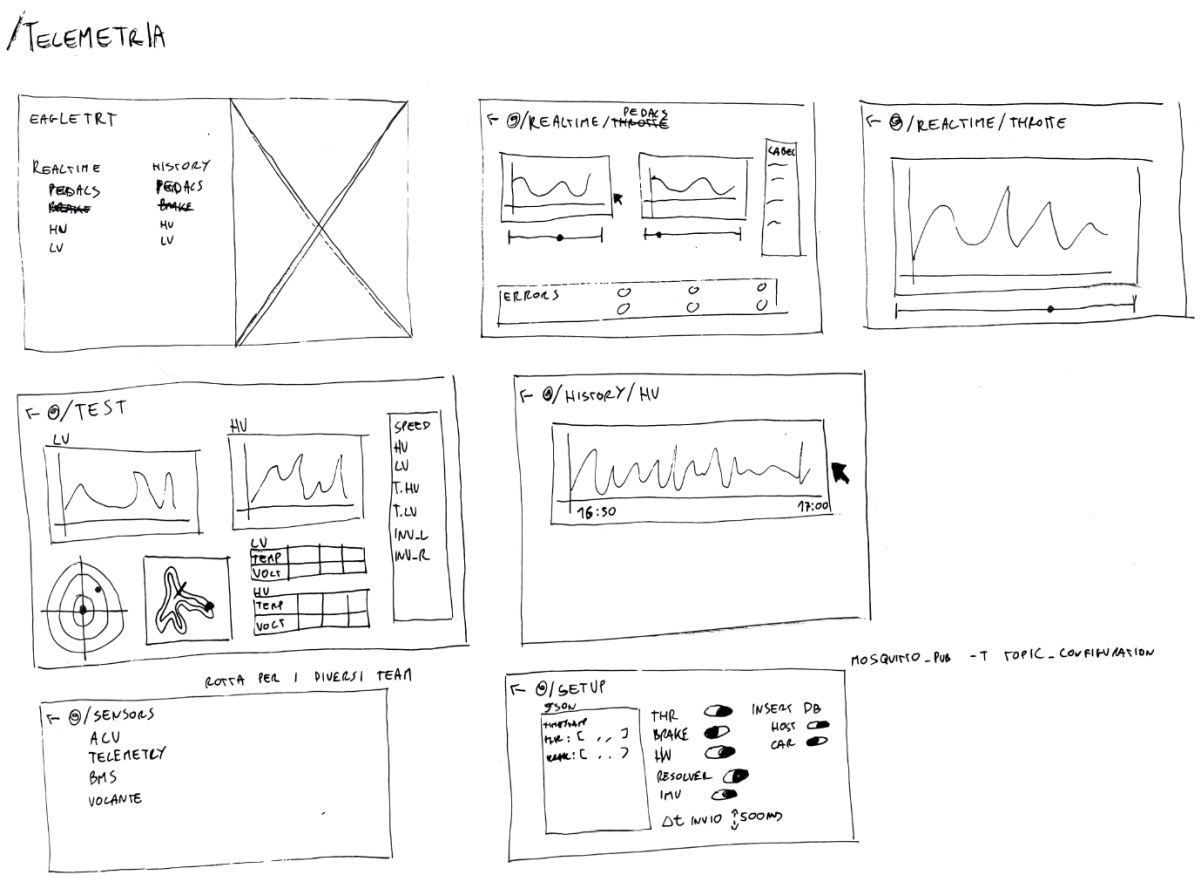


The ***Live*** microservice is used to insert the documents/data sent from the vehicle. This microservice consists of a proxy for the messages received on the vehicle topic. The broker is *broker.mqttdashboard.com*, but it will probably be changed from team members to provide better performance.

MQTT Publisher

The MQTT Publisher microservice is intended as a test service to emulate the condition of the car which is sending data to the online application.

Front-end



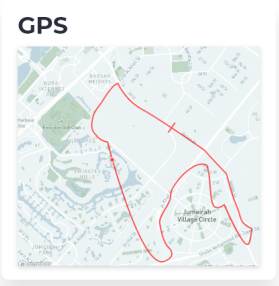
The first low-fidelity prototype is designed according to the feedback gathered via a questionnaire in 2019. Other interactions are required to validate the usability and the complete set of functionalities needed for tests and race events. This first mockup has as requirements:

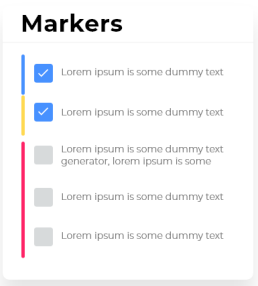
- Login/Signup for the user
- Group-based views
- Division of charts into cards for more customization
- The division into Live and History for single-seat data

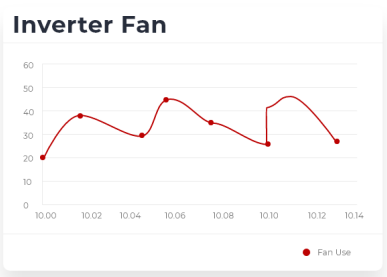
Live
- Pedals
- Steering
- Telemetry
- Low Voltage
- High Voltage

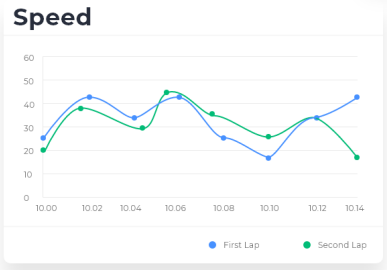
History
- Pedals
- Steering
- Telemetry
- Low Voltage
- High Voltage

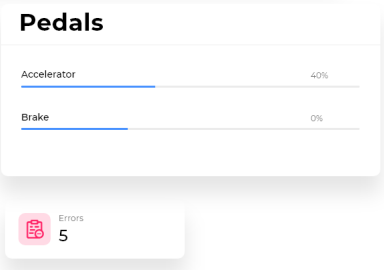
Live | Pedals | Steering | **Telemetry** | Low Voltage | High Voltage











Further researches and development

The project can be considered a good starting point to finish the team's telemetry solution. The skeleton and core functionality of the web app are now implemented and are available. The use of *docker-compose* allowed to deliver more instances of the most expensive service, *history*, which is now handled with a simple round-robin. The *load balancing* distributes client requests across a group of servers. A client request is forwarded to each server in turn. The algorithm instructs the load balancer to go back to the top of the list and repeats again.

Additional steps reported by the new project manager and myself are:

- Authentication system via OAuth.
- Microservice integration for downloading sessions from prototypes previously developed on Chimera.
- Distributed database management.
- Session payload sizing via paging methods.