Technical Report: Beyond Strava, Custom Analytics for Personal Fitness Data

Alessandro Mecchia and Paolo Nicolet
Bachelor in Data Science and Artificial Intelligence
Software Engineering, Modeling and Applications

June 3, 2025

Abstract

This project involves the development of a web service application and a client library designed to manage and analyze data from the Strava API stored in a database. The primary motivation is to offer athletes an efficient tool to handle detailed athlete profiles and activity data, as well as to enable comparisons between them. The core problem addressed includes managing a large volume of activity data and providing meaningful analytics to users. State-of-the-art technologies such as Flask and a modern database system (MongoDB Atlas) are employed. The approach includes building a robust backend to facilitate data interaction, alongside a client library that simplifies data retrieval and visualization tasks. Key results include successful integration of Strava data, comprehensive athlete management capabilities, and insightful analytics features. The project concludes by highlighting significant achievements and outlining potential enhancements, such as more advanced analytics, real-time data processing, and improved user interaction mechanisms.

Contents

1 Introduction

This report presents the design and development of a web service application and a Python client library for managing and analyzing fitness data retrieved from the Strava API. The project was developed as part of the course *Software Engineering, Modeling and Applications* in the Bachelor's program in Data Science and Artificial Intelligence. The goal of the project was to create a modular and extensible system capable of overcoming some of the limitations found in existing platforms like Strava and Garmin, particularly in terms of data accessibility, customization, and analytics.

The report is structured as follows:

- **Section 2 – Motivation and Context:** describes the personal and technical motivations behind the project, and the broader context of fitness data analysis.
- **Section 3 – Problem:** outlines the specific limitations in current tools that this project aims to address.
- **Section 4 – State of the Art:** reviews the most relevant existing platforms, technologies, and tools used in the field of fitness tracking and analytics.
- **Section 5 – Approach to the Problem:** details the system architecture, backend and client implementation, testing approach, and overall development strategy.
- **Section 6 – Results:** presents the outcomes of the project, including implemented features and outputs such as analytics and visualizations.
- **Section 7 – Conclusion:** summarizes the project's contributions and discusses possible future improvements and extensions.

2 Motivation and Context

Both members of the team are passionate about sports and regularly engage in various physical activities. One aspect we particularly enjoy is sharing our workouts with friends, challenging ourselves by comparing our performance and exploring the data behind our performances. To do this, we have long relied on **Strava**, a popular platform among athletes that combines social features with activity tracking and performance analytics.
At the beginning of the project, we considered using data from **Garmin**. Unlike Strava, Garmin tracks a broader range of physiological data such as sleep, stress levels, and body battery, which would have allowed us to offer a more holistic analysis of an athlete's condition. We found an *unofficial API* on GitHub that enables access to this data, but integrating it would have introduced additional complexity in terms of authentication, data structure handling, and long-term reliability. Given the scope and limited duration of the project, we concluded that using Garmin data was not practical.
Instead, we chose to work with Strava, which offers a well-documented and stable API. Although it focuses mainly on activity tracking, it still provides valuable data for performance analysis. In recent years, however, Strava has moved many of its advanced analytics features behind a *premium subscription*. This change has limited free users' ability to access detailed statistics like heart rate zones, pace trends, or elevation gain summaries.
This motivated us to build our own solution: a web service and client library that can

fetch and store Strava data locally, provide key athlete and activity management features, and generate custom analytics and visualizations—free from external restrictions.
Through this project, we aim to combine our interests in sports, data analysis, and software development to deliver a tool that reflects real user needs and opens up new opportunities for personalized activity tracking.

3 Problem

The primary issue that motivated this project is the increasing restriction of advanced analytics features on the Strava platform. Over the years, several data insights that were once freely available—such as:

- heart rate zone breakdowns
- detailed pace distribution
- training load analysis—

have been moved behind a **premium paywall**. This creates a barrier for users who want to understand and track their performance in more detail but do not wish to pay for a subscription.
Additionally, Strava offers limited flexibility in terms of customization. Users are confined to the default analytics provided by the platform, with little to no control over how their data is processed, filtered, or visualized. There is no support for creating custom queries or performing exploratory data analysis beyond the predefined charts and metrics.
Our project addresses these limitations by:

- enabling users to locally store and access their activity data
- replicating key metrics available on the Strava platform
- developing new types of analysis currently not supported by Strava

Furthermore, by building the system with extensibility in mind, the project opens the door to integrating advanced features in the future, such as machine learning-based insights. These could include performance trend prediction, anomaly detection, or personalized training recommendations—features that go beyond what is currently offered by mainstream platforms.
In short, our solution aims to give users **full control** over their fitness data and the freedom to explore it in ways that existing tools do not allow.

4 State of the Art

There are several platforms widely used by athletes to track and analyze their training data. Among the most popular are:

- **Strava**
- **Garmin Connect**
- **TrainingPeaks**

- **Polar Flow**

These platforms collect detailed activity data through GPS-enabled devices and offer various levels of analysis ranging from basic statistics to personalized training recommendations.
Strava is especially known for its strong social features and intuitive interface. It provides core metrics such as distance, pace, elevation, and heart rate analysis. However, many of its advanced analytics—like training load, performance trends, and heart rate zone analysis—are restricted to premium users. Moreover, its analytics are static and generic, offering little room for user customization or exploration beyond the platform's built-in dashboards.
Garmin Connect collects a broader set of health and physiological data, including sleep tracking, stress levels, and body battery. While this would allow for a more complete view of an athlete's condition, the absence of an official public API limits its integration potential. Although unofficial APIs exist, they bring complexity and reliability concerns, especially in the context of short-term academic development.
Strava, in contrast, offers a stable and well-documented official API that grants access to most core activity data, making it a more practical and reliable choice for integration. However, it provides only raw data, leaving developers responsible for implementing analytics and visualization logic.
To build these analytics, developers typically rely on general-purpose Python libraries:

- `pandas` for data manipulation
- `matplotlib` and `seaborn` for plotting
- optionally `Plotly` for interactive visualizations.

While these libraries are powerful and widely used in data science, they are not specifically tailored for sports analytics. As a result, developers must define the logic for extracting meaningful insights from activity data.
Our project leverages some of these tools to bridge that gap—creating a focused, reusable client library that handles Strava data and provides flexible analysis and visualization features out of the box. This approach allows users to generate insights similar to (or beyond) what is offered by premium services, while retaining full control over the data and the ability to expand the system in future iterations.

5 Approach to the Problem

To address the limitations of existing fitness platforms and provide personalized data analytics, we designed a system composed of three primary components: a MongoDB Atlas database for persistent data storage, a Flask-based backend web service and a class-based Python client library. Our goal was to create a modular, testable, and extensible architecture capable of retrieving, storing, and analyzing Strava data independently of the official platform's constraints.

Database

We chose **MongoDB Atlas**, a modern NoSQL document-oriented database, for its flexibility and natural fit with the data structure of athlete profiles and activities. Since

both athletes and their activities have variable fields and hierarchical data, MongoDB's document model allows efficient and intuitive storage.

Our database, named `strava-db`, is hosted online and contains two main collections:

- **athletes** - storing individual athlete information such as ID, name, gender, and weight, etc.
- **activities** - storing activity records (type, distance, duration, timestamps, streams, etc.), each linked to an athlete via the `athlete_id` foreign key.

Backend Web Service

The backend is built with the **Flask** web framework and structured using **Blueprints** to separate different route groups (e.g., athlete routes, activity routes) into logical modules. This design choice ensures maintainability and scalability of the codebase as new features are added. The service interacts with the MongoDB database.
We used **MongoEngine** as an Object-Document Mapper (ODM) to simplify the interaction between Python classes and MongoDB documents. For lower-level operations and performance tuning, we also incorporated **PyMongo** in specific service layers.
The backend exposes RESTful endpoints to:

- Retrieve all athletes or activities, or specific items by ID
- Filter activities by parameters (e.g., type, time range, duration)
- Aggregate activity data (e.g., total distance per week)
- Return data summaries useful for analysis and plotting

All logic is separated into *services* that handle business rules and *routes* that manage HTTP request-response cycles. This separation of concerns, combined with Flask's Blueprint support, helps keep the architecture clean and extensible.

Client Library

The client library is implemented in Python, following an object-oriented approach as practiced in class. It encapsulates API communication logic using the `requests` library, abstracting away HTTP complexity behind intuitive class methods. For example, developers can easily fetch all activities in a time range or retrieve athlete metadata with a few lines of code.

The library includes:

- A `Client` class for API interaction
- Config and exception handling modules
- Service modules for managing athletes, activities, and data streams.

This makes the library both a convenient developer tool and a useful analytics utility for end users.

Testing and Packaging

To ensure software reliability, we implemented **unit tests** across both the backend and client components using **pytest**. Backend tests validate the service logic and data retrieval endpoints, while client tests simulate interactions by **mocking the backend API**. This allows isolated testing of client behavior under various conditions without requiring live server access. Tests are organized by module (e.g., `test_client.py`, `test_athlete_services.py`, `test_activity_model.py`) and can be executed independently or in batch.
From a packaging perspective, the project is organized into two main Python packages: `web service` for the backend, and `client lib` for the client library. The `client lib` package includes a sub-package `services`, which encapsulates logic related to athletes, activities, and streams.
To facilitate reuse, the `client lib` is packaged with a `setup.py` file, allowing it to be built as a distributable wheel. This enables easy installation and importation into other modules, such as the utility scripts used by the demo application The packaging process uses standard Python tools to build and distribute the library, ensuring seamless integration into various environments via `pip`.

6 Results

The project successfully delivered a fully functional command-line interface (CLI) application that allows users to manage athletes and their activities, perform data analysis, and visualize routes and streams. The implemented menus provide a comprehensive set of features, including:

- Athlete Management: viewing, adding, editing, and deleting athlete profiles, as well as summarizing athlete data.
- Activity Management: viewing all activities, showing detailed information, adding, editing, and deleting activities.
- Analysis and Advanced Functions: comparing athletes, listing athlete activities, providing statistical overviews, and visualizing route and stream data.

Below are some images of the advanced features of our project, such as specific graphical analysis for each activity (obviously if the data is available) on heart rate, elevation difference, average pace, and the map plot with all runs. There is also an image representing one of our favourite features and the reason we wanted to develop this project: the comparison between two athletes in a sport chosen from running, swimming, and cycling.

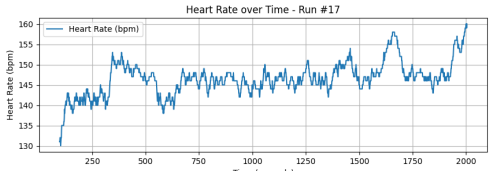

Figure 1: Heart rate stream visualization for a selected activity.

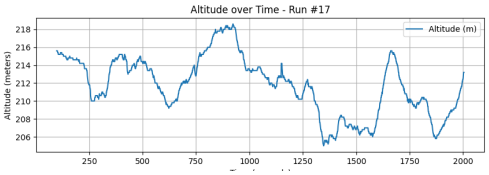

Figure 2: Altitude profile over time for a running activity.

Figure 3: Pace evolution during a run.

```
=== Analysis & Advanced Functions Menu ===
1. Compare athletes
2. List athlete's activities
3. Athlete statistics overview
4. Visualize routes and streams
5. Back to main menu
Select an option: 1

>> Compare Two Athletes in a Sport
Enter first athlete ID (integer): 134836178
Enter second athlete ID (integer): 25454948
Choose sport from: ['Run', 'Swim', 'Ride']
Enter sport: Run

Comparison in Run between Athlete 134836178 and Athlete 25454948:
Athlete 134836178: 72.08 km, 1131.00 m elevation, 15 activities
Athlete 25454948: 37.02 km, 249.00 m elevation, 8 activities
```

Figure 4: Athlete comparison based on total, total elevation, total activities for that sport.

Figure 5: Map showing all running routes of an athlete based on GPS data.

While the core functionalities were implemented correctly and tested, some advanced analysis features initially planned were not completed due to time constraints, like specific routes in the `web service` to retrieve specific data for extra analysis.
During development, we encountered issues related to environment management, particularly with `pipenv` configurations for the `demo` folder. To resolve this, we consolidated the environment setup by creating a single `pipenv` at the project root level. This unified environment allows seamless running of the demo, utility scripts, and the `run.py` script that launches the web service backend. At the same time, we successfully implemented separate environments for the other project packages, maintaining modularity and isolation where needed.
Overall, the project met its main objectives, providing a robust backend and client library, along with a user-friendly CLI for interaction and analysis. The unresolved advanced analysis features remain areas for future improvement.

7 Conclusion

This project presented the design and implementation of a client-server application for managing and analyzing fitness data retrieved from the Strava API. By developing a robust Flask-based backend connected to a MongoDB Atlas database, alongside a Python

client library and a command-line interface, we provided users with flexible tools to store, query, and visualize personal activity data independently from platform restrictions.
Working with this type of data proved particularly interesting due to its complexity and structure, especially considering the hierarchical nature of athlete profiles and activity documents. This experience highlighted the potential benefits of adopting specialized databases for certain data types, such as time-series databases for the streams data, which could improve performance and analysis capabilities in future developments.
Despite some advanced analytics features remaining to be implemented, we successfully developed functions that were of particular interest to our team, including detailed data analyses and direct comparison tools for athlete performances across different sports.
Environment management challenges were addressed by consolidating the demo environment while maintaining separate environments for other project components, ensuring modularity and ease of development.
Overall, his project shows that it's possible and useful to build custom fitness data tools without depending on commercial premium platform features. It creates a strong base for future improvements, such as adding real-time features, using machine learning, and including more types of data