

Honours Project Report

Fitness Logger with Micro-services

Dumitru Vulpe
BSc (Hons) Applied Computing
Supervised by Andrew Colby

May 2021

Abstract

This project is a tool to let people be able to log and track workouts overtime easily from their phone in a flexible yet complete way. The initial purpose was to make a phone application which can be used across different disciplines of sport and with different workout types. This would be done by breaking up the data in different manageable units so that the use can create their own workflow for logging workouts.

However, the main appeal of this project is the backend implementation. This is because as a part of the architecture design process, it was decided that a micro-services structure would be followed. Where the backend would be split up into multiple applications which would talk to each other where needed. This was chosen for a multitude of reasons, including as a learning experience for this kind of backend architecture.

Contents

1	Introduction	3
1.1	Why micro services	3
1.2	Creating a potential product	3
1.2.1	The problem	3
1.2.2	My planned solution	3
2	Background	3
2.1	Architecture design	3
2.1.1	Helpful technologies	4
2.2	Market	4
3	Specification	4
3.1	Micro services	4
3.1.1	Business & general project management problems	4
3.1.2	The technical aspect	5
3.1.3	Solution	5
3.2	The application	5
3.2.1	Solution	6
4	Design	6

1 Introduction

TODO: Need to come back after I'm done with the entire report to summarise it

This project is split into two main objectives; first objective was to be a learning experience of creating a full stack application beginning to end and more importantly to get into micro services backend development. The second objective was to attempt to solve a problem which exists in the fitness industry by creating an application which would be useful to a different range of people and potentially taking this product to the market.

1.1 Why micro services

The basic principle of micro services as supposed to a monolith, is to split up the backend application into multiple smaller application. For example, in theory, a service such as Amazon, could have a separate micro service for the cart function, one for listings, one more user account management, one for payment and more. Micro services architectures are becoming more and more relevant simply because the industry is getting becoming more and more complex. And as size, complexity and user bases increase, so does the need for well scaled web applications. Micro services type architectures are simply one way to cleanly and easily scale web applications. So because of industry relevance and my personal enthusiasm, it was decided that this project's backend would also follow this architecture.

1.2 Creating a potential product

Creating a folly fledged product out of this project was never my main idea. However, in my personal opinion, my planned solution for the problem has enough potential that could work as a product in the market. And as long as the application works as intended, there are multiple potential monetisation routes which could be taken, such as extra features behind a subscription or just a flat rate at purchase.

1.2.1 The problem

From personal past experience I have not found a simple, fuss free, yet flexible application for logging workout sessions and related notes for said session. The applications which I have tried so far are either made for a specific workout programs or are not complete, buggy or just do not have enough flexibility in the type of logging you can achieve. It often feels like a fight to be able to

use such an app.

In the fitness world there can be a lot of different types of workouts, exercises, exercise variations, and ways to do said exercises. Because of this, coaches, personal trainers and everyday people usually would stick to using a notepad and paper simply because there are no arbitrary restrains on what you can log.

Typically a notepad and pen is not a bad solution and usually preferable to most applications, however, there are certain drawbacks to this approach. One practical drawback is highlighted when working out in a commercial gym, walking around the gym using different equipment and carrying everything one might need including the notebook and pen combo can be quite cumbersome, same goes for actually noting down the data in the notebook. When it comes to looking back on previous workouts it can also be quite cumbersome, especially when you are in the middle of a workout and do not want to loose the tempo.

1.2.2 My planned solution

The idea was that the system would be designed around the data structure which would be saved by the user. This data structure would be made as simple as possible while still allowing all the common basic exercise options (such as repetitions, resistance values, sets etc) but also others (such as the option to just write a single value). Furthermore, the system would allow the user to create their own exercise types which they would then use when logging a workout.

When using the application the user would first create a custom 'type'. This type would usually be an exercise, however, it it not constrained to be one. For example, one type would be 'exercise x' and another type could be 'caffeine intake' for tracking the intake of caffeine over different workouts. Then, the

2 Background

TODO: talk about common problems that arise in companies and how using micro services can help solve those problems. See if u can find some sources

2.1 Architecture design

TODO: see if u can find a citation for the "popular questions to be answered"

A lot of time during this project was allocated to research of the backend architecture design, specifically relating to micro services. And when it comes to it, there is not just one or few generally accepted standards and patterns that projects adopt. This is simply because each and every system architect needs to answer a lot of questions about it will all be built. For instance, the questions about how the different services should be split, some say it should be split by the feature, some say by the individual data element, by endpoint and more. Another popular questions is how the services diagram should look like, one of the popular answers are to have a separate authentication service to provide authentication for the application and proxy the requests to all the other services. However, regardless of the question, there are a lot of different ways solutions to all of these questions that need to be answered when starting a project using micro services and all of them come with their own advantages and disadvantages.

2.1.1 Helpful technologies

When creating micro services systems, there are a lot of technologies which could and would be used to make the development of these services easier. Probably one of the first pieces of technologies to ease the development and deployment of micro services a lot easier was containers. Containers allow you to package applications with all of the needed dependencies and environments, they virtualise the user-land applications and share the hosts kernel, and on a Linux system that means that containers can virtually run anywhere. Furthermore, we have also have container management and orchestration solutions such as Docker[1], docker-compose[1], Kubernetes[2] and more. These help with running the actual containers, and in the case of Kubernetes, it also allows us to easily define configurations and scale the containers either on a definition or on demand. Scaling here would be done by running more instances of that container across a cluster of machines running the Kubernetes and load balancing the traffic amongst them. This is known as horizontal scaling.

Micro services would often also need to talk to each other and as before, depending on requirements and design, there are different ways to do it. Overall there are two options many systems using both synchronous and asynchronous calls. Synchronous calls are usually straight forward and done over HTTP and the only thing that would need to be solved is proper authentication. Asynchronous calls are often made with either RPC calls (commonly gRPC[3] would be used) or a message/event bus system such as Kafka[4].

Another great series of helpful technologies are service meshes, they provide great insight into your cluster of micro services. It is a dedicated infrastructure layer that all the different micro services would proxy their communication in-between. This can provide a multitude of benefits including communication observability, call chain traceability, secure connections and more. Examples of such services are Linkerd[5] and Istio[6]. If this project would be using Kubernetes, a service mesh would definitely be very useful and worth while to integrate into the system.

2.2 Market

TODO: Need to find and list of competitor apps
- the one I have tried was zero to hero

3 Specification

3.1 Micro services

The proposed project is tackling two main problems, one being a whole set conceptual problem which are very common with increased project sizes, weather that is in terms of complexity, code base, team size or general scope of the project. So it is generally welcome to try and apprehend for these problems ahead of time by careful planning and designing of the system. In this case, the careful planning and designing would be done on the backend architecture. Here, using micro services would solve a lot of problems, both technical and business problems.

3.1.1 Business & general project management problems

When a project grows in size to the point where there is a need for multiple teams it can become quite challenging to have all of the teams work on the same code base. Different tactics such as smart management of source management system could help with this issue. One such example would be to have different development branches for each team, however, this could introduce more problems such as needing to re-base the branch mid development. Here a much better way would be to simply have the teams work on different codebases, each making up a micro service. Depending on how the teams are organised, or on how the features are broken down in each micro service, each team would be solely responsible of one or more micro services, making them

the de facto experts on that specific micro a service and its inner workings.

What about when new individuals or teams enter the workforce? There is usually a fair amount of time taken just simply getting acclimated with the codebase and the whole product. Now, this would be true weather the project is or is not following a micro services architecture, however, if the project is just a monolith, usually the individual or team would need to get familiar with the while codebase. This would not need to be true in case of micro services.

With micro services the individual or team would just need to look at the API documentation of the project, then only get accustomed with the micro service(s) that they would be working on. This way collaboration between teams also will become easier due to each team maintaining their own small codebases, and documentation for how to interact with their services.

3.1.2 The technical aspect

Originally projects would just be one monolithic application and if we needed more performance we would optimise the code and add more power through better server hardware (or add more resources to the virtual machine or container if it was applicable). This is called vertical scaling. However, what happens when we need even more scaling? There can only be so much power we can add to one single server and only so much optimisation we can do to a codebase. Projects then started to scale their applications horizontally by deploying more instances of it, then adding some form of a load balancer which would balance the requests across all of their different deployments. These deployments could be individual computers, running the application directly on bare metal¹. This, however, could be quite inefficient because it would scale the whole application and this might not be needed.

For example, assume *Application x* has a user profile endpoint which is just a simple CRUD² operation and a pictures endpoint which is responsible for handling picture uploads, picture compression, storage, serving, etc. The part which will need the most processing power will be the pictures endpoint, but when vertically scaling a monolith, all of the endpoints get deployed again regardless if they needed the extra resources or not. This in principle is wasted resources and by extension wasted budget for a project.

¹Bare metal meaning that there is no other layer of management such as virtualisation or containerization and is running directly on the host OS.

²Referring to data operations of Create Read Update Delete.

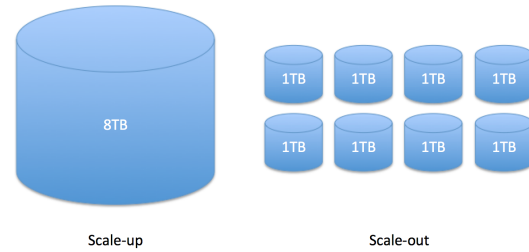


Figure 1: Scaling out vs scaling in

Note figure 1, this is an example of vertical scaling (or scaling up) versus horizontal scaling (scaling out) for a file system.

3.1.3 Solution

As it turns out, micro services architecture, plus technologies such as Docker and Kubernetes solve this exact problem by having each feature/endpoint/verb/structure on a different service then vertically scaling each service when needed with a load balancer re-directing the traffic to instances.

In the long term this will also save on resources because developers will not need to spend as much time on writing code which can will take complete advantage of all of the computing resources available to it.

3.2 The application

When it comes to logging any sort of fitness activities, there is no standard, and there are a lot of different ways an individual or a trainer might log these activities; there simply does not exist a *one size fits all* solution to this. It all just depends on the goals of said person or trainer. Because of this it is gets fairly complex when it comes to trying to make a digital form which could be filled out and be able to process and display all the data from it in any meaningful way. Usually different applications would create a rigid structure for the data then there would be forms to input the data in. For some people that would work perfectly, but not for all especially across athletes from different disciples of sport.

#	Exercise	Sets
1	Squat	10x40kg, 10x60kg, 8x70kg, 8x80kg
2	Dead lift	12x50kg, 12x70kg, 10x80kg, 8x100kg
3	Hack squat	15x50kg, 15x60kg, 15x70kg, 12x70kg
4	Leg press	15x40kg, 15x60kg, 15x70kg, 15x80kg
5	Ham string curl	15x26kg, 15x32kg, 15x36kg, 15x32kg

Table 1: Simple workout log example

For instance, one of the simplest logging types would be just performing straight sets of different exercises, some exercises being done with weights, or some other form of resistance which would also be noted down with every set. Table 1 shows one such example. However, as mentioned just now, it is a very simple form of logging and there are a lot more details that could be added, or a lot more variations that people could be doing which would need to be noted. Some examples are the following: *super-setting*³ and *drop-setting*⁴. These are just some of the more common ones, however, not everyone will use these exercise techniques so not everyone will need to be able to log them. All of this adds extra complexity, which means an application for logging will either be very complex, or only useful to a subgroup of people with a specific workout style.

3.2.1 Solution

The premise to the solution in this project is quite simple. *make it simple and stupid*. The solution which is being proposed it to simplify the data structure as much as possible while still including

NOTE: - the flexibility part of the workout logging
- explain how each user can create their own types which could be anything not just exercises with their own defined measurement units and more.
- could even talk about my initial idea of creating a highly flexible tagging system which can be searched

4 Design

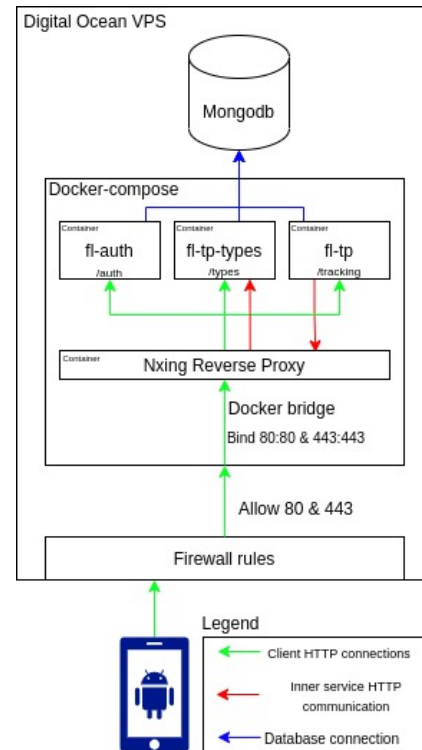


Figure 2: Fitness Logger Architecture

- Overall arch -

³Super-setting is performing a set of one exercise and another set of another exercise straight away without resting, then repeating the cycle.
⁴Drop-setting is when the athlete performs another set without set at a smaller resistance, usually -20%

References

- [1] Docker (Software) Wikipedia (2021).
[`https://en.wikipedia.org/wiki/Docker_\(software\)`](https://en.wikipedia.org/wiki/Docker_(software))
- [2] Kubernetes website (2021).
[`https://kubernetes.io/`](https://kubernetes.io/)
- [3] gRPC website (2021).
[`https://www.grpc.io/`](https://www.grpc.io/)
- [4] Kafka website (2021).
[`https://kafka.apache.org/`](https://kafka.apache.org/)
- [5] Linkerd website (2021).
[`https://linkerd.io/`](https://linkerd.io/)
- [6] Istio website (2021).
[`https://istio.io/`](https://istio.io/)

Appendices