Tech387

tech387.com
Support@tech387.com

SpartanApps

Health & Fitness, Production-level software

Summary

SpartanApps is a production-level software which has been downloaded over five million times and is currently used by millions of people. From 2013 we have had an honour to work on this idea and later on a development of the fully functioning product, as we are the co-founders of SpartanApps.

The app has a myriad of features and some of them are: personalised AI-powered workout system, smart nutrition tracker, continuous progress tracking, intelligent data synchronisation, data analyzing / data warehousing and many many more.

Some of the challenges on such a huge and complex product were to scale the software to facilitate the number of people using it and maintain the speed of overall performance while preserving security in parallel, representing the core qualities of our product.

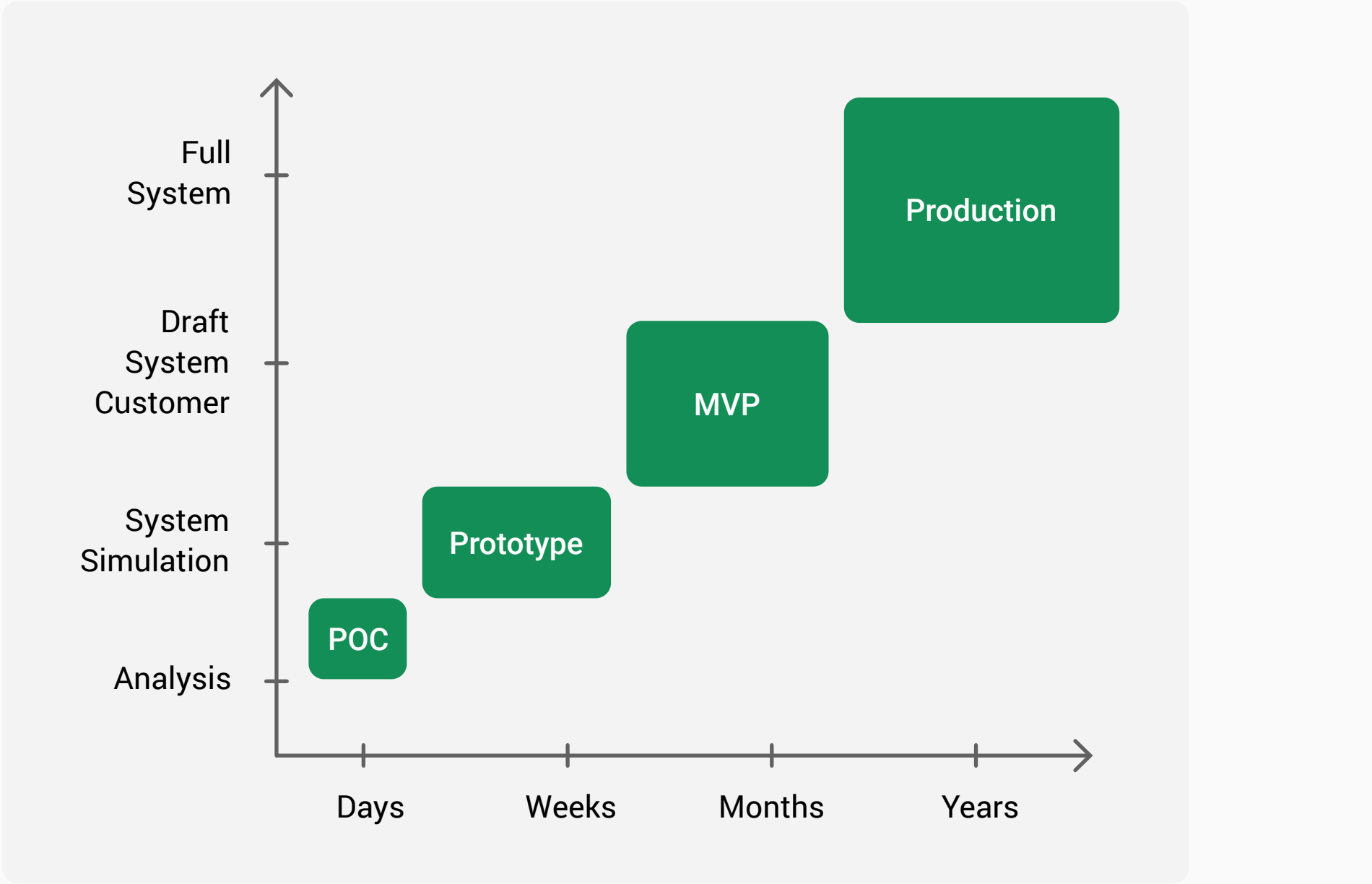
Tech challenges

The app did not have a synching mechanism until 2017 and users were unable to backup and sync their data.

To solve this problem, we created a content management system which performs content versioning and provides support across numerous languages in our app.

Imagine it as a super highly scalable system for content management.

The problem behind this is how to run the production-level software the right and in the best possible way?



DEDICATED HOSTING

Pros of Dedicated Hosting

- High Security
- High Performance
- Some Configurability

Con of Dedicated Hosting:

- Mid-Level Scalability
- High Cost

CLOUD HOSTING

Pros of Cloud Hosting:

- High Scalability
- High Performance
- High Security
- Some Configurability

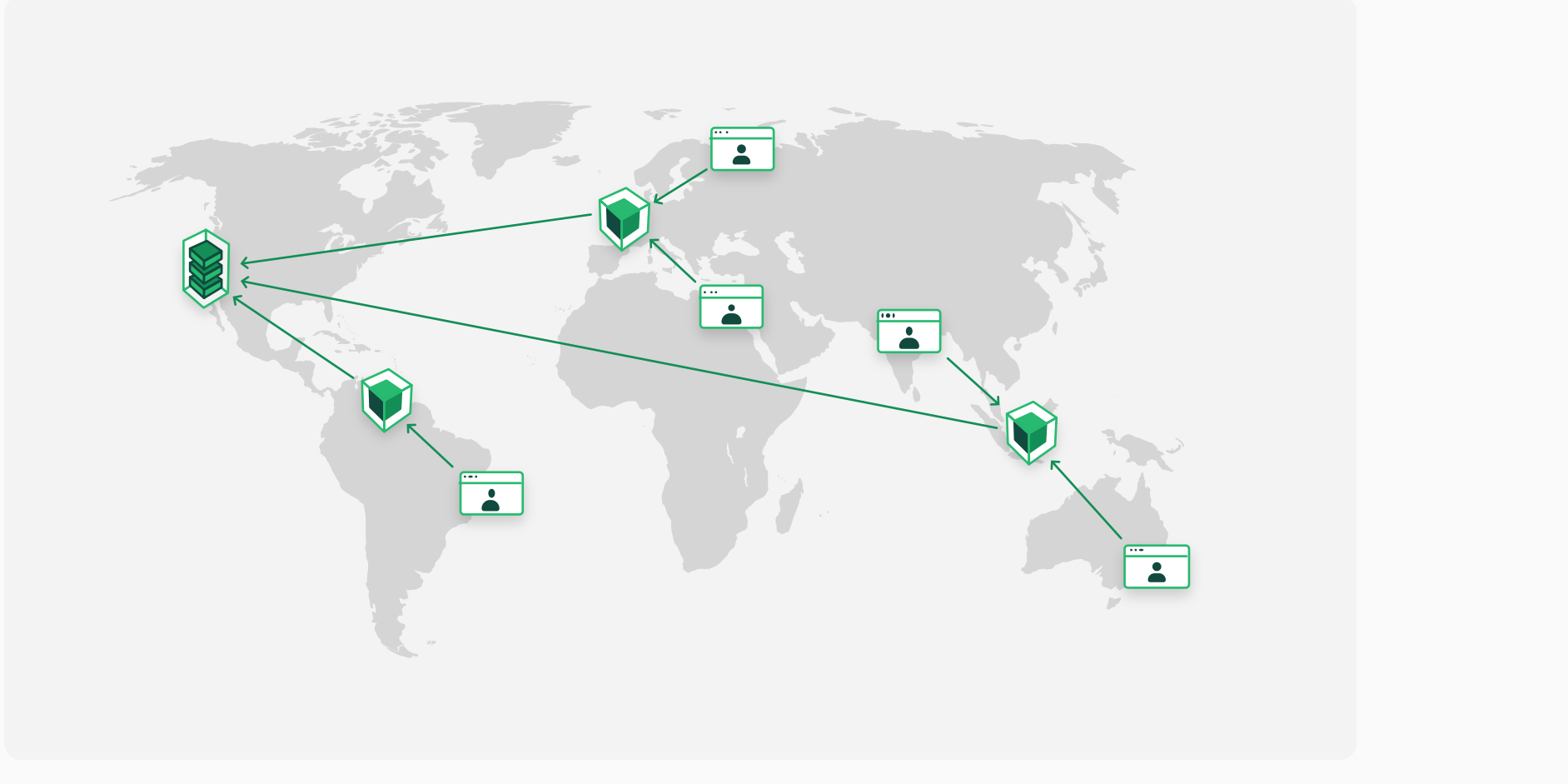
The key challenge is to create features, analyse data or make any kind of development progress when we have thousands of users accessing at an endpoint or a large number of them accessing the app from a single host.

Imagine the feature which synchronizes the user's workout history. If we have one million active users who regularly update the app and sync their workout data, we end up with over 30k requests per second on top of requests from usage of regular features such as logins, nutrition scanning, image syncing, etc.

This scenario creates a significant load on your databases. If there are jumps in processings, then servers need to be scaled up which consequently increases server costs. Alternatively, we can split the user base in two halves, with one half being in Europe and another one in the USA. However, if the hosting is in the USA, there will be a delay in receiving requests from our European users.

To solve this problem, we found a solution in setting up a content delivery network (CDN), a system of geographically distributed servers that speed up the delivery of web content. Having servers closer to our users' physical locations allows them to access content more quickly as their requests travel faster and over shorter distances.

The list of challenges continued, as we had to address issues and find innovative solutions to minimise costs, while creating a fast and responsive system.



ASO Challenge

As we have mentioned before, SpartanApps is our own product family and with that in mind, we were challenged with finding innovative ways to reduce expenses. At the beginning, there was no budget set aside for marketing. However, we needed to increase app installations if we wanted to gain some traction.

Since traditional marketing methods were inaccessible due to a lack of budget, we had to find unconventional ways to solve this issue. Our salvation appeared in the form of an App Store Optimization (ASO).

In 2013, we started actively cracking the ASO algorithm, only to realise at a later stage that this was our safest and the most economic path forward. Since then, hundreds of hours of tireless work were invested into getting this right. Nonetheless, we now pride ourselves with extensive ASO experience and results we achieved with this method. From being banned by the UFC for using their name in keyword optimization, to building backlinks on Reddit and optimizing relevant keywords, we finally reached over 1 million installs in only one year.

Under normal circumstances, this would have been worth more than $2 million in advertising spend as most of our installs came from Tier 1 countries. However, thanks to our technical expertise and creativity, we managed to reach this achievement with zero investment.

" *A framework does not define a project architecture* "

Mārtiņš Tereško

Challenges we addressed in the development process

The following is a list of challenges we addressed one by one:

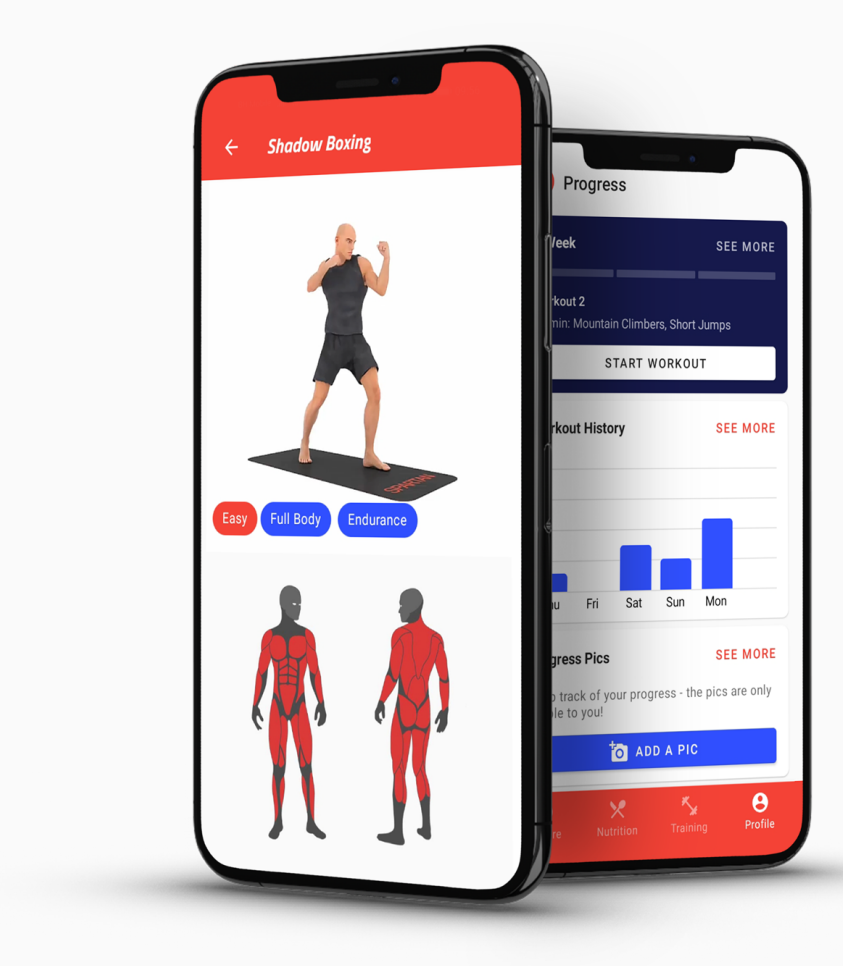
1. Security
2. Scalability
3. Speed
4. Flexibility
5. Maintainability
6. Stability
7. Insights on user and event data
8. Personalization

Considering our objectives, we had to create an economically efficient infrastructure which had all these features running optimally for a production-level software. It wasn't an easy task at all. In fact, it took us four times from the start to get it done right. To find out more about our efforts in this stage, check out our articles on how NOT to build a production-level software.

As you may notice, the tech and innovations we came up with to solve these challenges had an unconventional approach, as our motto in approaching software development is
"go hard or go home"!

On the backend we used PHP with C. Yes, you've heard well, PHP!

Some of you will ask why PHP? It's simply because we are pushing the boundaries and using the best possible existing solutions in our approaches. Anyone who disagrees can go and use the JS-based framework and take an easy, but not so good, road.



The way we used PHP with C was by writing our C scripts and compiling them in PHP. Once you call a function in PHP, it just increases the execution time to insanity! On top of that, if the execution is slow on PHP, it means that the code is not good enough. In essence, this combination acted as an adequate benchmark for writing quality code.

After we optimized the PHP code with C to be fast enough, we developed some microservices in GoLang and Python.

In honesty, GoLang was not our priority since Google made it. However, the combination of Python with C level libraries produced processing at the speed of light. If one connects the same C script to PHP, the results would end up the same. For anyone who disagrees, we are more than pleased to discuss and benchmark their code versus ours.

Finally, we focus on our solutions:

1. Security

This one is always difficult as you never know what will happen and how someone may attempt to damage your product. We solved this by implementing authentication using Oauth2, access control for our admin consoles, encryption of sensitive data and secondary approval for any data deletion or manipulation. Moreover, we connected only a few peripheral microservices to the public internet while most of them remained in a private network.

2. Scalability

We've mentioned that we do things the hard way. Well, as we already had a CD ready here and didn't want to use a third-party software, we ended up configuring Jenkins on Kubernetes from scratch. With load balancers in place, we scale our microservices according to the load. What's more, we have regular health checks to make sure that all of our pods are functioning seamlessly.

3. Speed

In order to achieve a high speed of processings around the globe, your queuing and caching performance has to be excellent. Here is the example: a few years ago we were working with YouTube API. It took them one hour to process the queued data for a like on a video which was requested over their API. This is a great example that prompts you to address queuing and caching performance in order to achieve speed. It also goes without saying that a certain budget has to be invested to optimize speed. Therefore, we hosted our code in multiple regions using CDNs, resulting in a product that works like a charm.

4. Flexibility

What if the design team creates crazy features which our users like? Also, did we mention that we had to redo the infrastructure multiple times? Well, this was one of the reasons why. If you require a new feature, and if you are like some of our developers who like SOAP and REST, you will push these communication protocols. However, the reality is that you need something more flexible - something that allows mobile developers to develop features which do not need direct access to a database and where you can remove additional requests for syncing. To solve this, we found an answer in GraphQL and it is truly delivering needed flexibility on a large scale.

5. Maintainability

Several experiences guide our suggestions for easing app maintainability: do not use third party dependencies, write tests, use a lot of automation, have a CI pipeline and monitor that closely.

6. Stability

You can have all of the above functioning great, but what if some of your pods fail, or what if for some reason a service is failing by your cloud provider (a reason we switched from GCP to AWS)? Have in mind that you might need to have a failover system for a failover system. Thanks to the Kubernetes team, we had a lot of flexibility in creating a structure which is functioning superbly.

7. Insights on user and event data

As famous author Peter Drucker says, "what's measured can be managed". To gain insights on user and event data, we could have gone with options such as Firebase Analytics to gather event data and Mixpanel for user data. However, this wasn't enough for us as we tend to do things the 'hard' way. Effectively, we set up a data warehouse in the background to collect the data, on top of which we applied our data magic to maximize user experience. One great thing that we did here was live streaming data from workouts to our warehouse. After the whole experience is recorded, it is then analysed to optimize user experience. While it might be lightweight, it is a very powerful and effective feature.

8. Personalization

This is linked to the previous data part, however here we played with machine learning, a field where we initially lacked expertise. At the beginning we used the AWS ML services which are great for the things we had to solve but recently we turned to our own autonomous solutions. So, stay tuned to see what's coming!

How do we know that our approach was right?

We will mention just two statements that demonstrate our expertise in this approach:

1. We managed to find a bug in GCP MySQL, consequently switching to AWS and,
2. The AWS architecture engineers viewed our code and stated that our architecture is perfect to the point where they would not change anything on it.

Results

We are very proud to have built a cost-effective product that is efficient and is every day exceeding our own expectations. We had started the SpartanApps as an idea that is today a globally recognised product. To date, millions of people have used our app and it continues to gain new users and features. The emphasis remains on the impact the app has produced in easing and changing the lives of our users, which constantly motivates us to release new features. A US veteran who had lost lower limbs in a war once reached out to us with an emotional feedback and stated: *"Thanks to SpartanApps, I have been able to create custom workouts that helped me regain physical as well as mental strength. This app has really brought me back to life."*

Our team successfully accomplished combining the most innovative solutions with focus on high performance, AI integration, and real-time data and operational analytics. Stay tuned for more insights on our journeys where we share lessons learned and solutions we applied.

Get in touch if we can be of assistance to you and let's turn your idea into reality!

Tech stack

Backend

aws, RabbitMQ, Jenkins, GO, Microservice, Google Cloud Platform, RPC, PostgreSQL, { REST:API }, node, python, MySQL, MEMCACHED, mongoDB, php

Frontend

ANGULAR, React, Ruby, TypeScript

Mobile

Kotlin, Swift, Java

Other

Firebase, mixpanel