SZAKDOLGOZAT

Bordás Olivér Odin

2022

GÁBOR DÉNES FŐISKOLA

MÉRNÖKINFORMATIKUS SZAK

SZAKDOLGOZAT

SEEKING ALPHA ON THE CLOUD:

AN SAAS SOLUTION FOR MARKET ANALYTICS

Bordás Olivér Odin

EC33NW

053/2022

SZAKDOLGOZATI TÉMAVÁZLAT

Hallgató neve, Neptun kódja:	Bordás Olivér Odin, EC33NW
Szak/Specializáció:	Mérnökinformatikus / Szoftverfejlesztés
Telefonszáma:	0 6 70/3619916
E-mail címe:	zedasgm@gmail.com

Szakdolgozat területe:	Szoftverfejlesztés
Szakdolgozat tervezett címe:	Seeking alpha on the cloud: An SaaS solution for market analytics.

A szakdolgozat célja, rövid tartalma és vázlata (tervezett tartalomjegyzéke):

Témaválasztásomat a szoftverfejlesztés és a pénzügyi világ inspirálta melyben az évek során dolgozhattam / dolgozom.

Ezen két iparág nyelve az angol, ezért szakdolgozatomat én is angolul szeretném megírni.

Szakdolgozatom célja olyan, a DDD elvek mentén Javaban megírt SaaS-t létrehozni, mely a ma igen elterjedőben lévő microservice architecture alapokra épül, tehát egymáshoz lazán kapcsolódó, önálló, konténerizált szervizekből áll össze.

A szolgáltatást használni egy mai modern webes keretrendszerben fejlesztett UI-on keresztül lehet majd. A létrejött szoftver képes lesz arra, hogy a hozzákapcsolt adatforrásból megszerzett értékeket elemezze konvencionális piac elemzési technikákkal (melyek testre szabhatóak a felhasználó által) ezek eredményeit pedig riport/egyéb formában a felhasználó elé tárja.

A szoftver architecturájának köszönhetően szabadon bővithető lesz új szolgáltatásokkal. Eredménynek egy modern, könnyen bővithető, skálázható, bárhonnan elérhető piac elemző eszközt tekintek.

Tervezett vázlatos tartalomjegyzék:

- Témaválasztás indoklása

- Megoldásra váró pobléma kifejtése

- Felhasznált technologiák és módszerek bemutatása és indoklása

- Architectura bemutatása és indoklása

 - Szoftver tervezési és fejlesztési folyamatának bemutatása

- Szervizek tervezési és fejlesztési folyamatának bemutatása

- UI tervezési és fejlesztési folyamatának bemutatása, indoklása

- Megoldási alternativák rövid bemutatása mind Backend, Frontend, és DevOps oldalon

- Piac elemzési technikák bemutatása, indoklása

- Egyéb üzleti folyamatok összefoglaló bemutatása, indoklása - Eredmények összefoglalása, zárógondolatok

Tervezett vázlatos technologiai stack:

Spring Boot, Angular, Docker, Kubernetes, Java 8, Maven, Eclipse, Git, Sql, ...Stb

A konzulens adatai:

	Belső konzulens / Külső konzulens	
Név:		
Munkahely:		
Beosztás:		
Végzettség (min.		
Telefon:		
E-mail cím:		

(*Külső konzulens esetén végzettséget igazoló dokumentumot, és a konzulens önéletrajzát a témavázlat mellékleteként KÖTELEZŐ csatolni)

A javasolt téma konzultálását vállalom, valamint hozzájárulok, hogy a végzettségemet igazoló dokumentumomat, és az önéletrajzomat a GDF szak/záródolgozat kezelésével/ügyintézésével foglalkozók megismerjék:
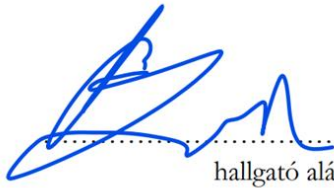
(csak külső konzulens esetén szükséges)

…………………………………….

külső konzulens aláírása

Kérem a javasolt szakdolgozati téma jóváhagyását.

Budapest, 2022.05.27.

…………………………………….
hallgató aláírása

Konzultációs lap

GÁBOR DÉNES FŐISKOLA　　　　**DENNIS GABOR COLLEGE**

KONZULENSI LAP

Hallgató neve, Neptun kódja:	Bordás Olivér Odin, EC33NW
Szak/Specializáció:	Mérnökinformatikus / Szoftverfejlesztés
Szakdolgozat sorszáma:	**053/2022**
Szakdolgozat címe:	Seeking alpha on the cloud: An SaaS solution for market analytics.

Konzulens neve:	Peck Tibor János
Konzulens munkahelye, beosztása:	Gábor Dénes Főiskola 1119 Budapest, Fejér Lipót u. 70. főiskolai tanársegéd

A konzulens véleménye a kész szakdolgozatról:

A Jelölt szoftverfejlesztő informatikusként a pénzügyi világban tevékenykedik, így a témaválasztás és a végrehajtott feladat közel áll hozzá. A felhő infrastruktúra alapjain hozott létre, egy a pénzügyi területen tevékenykedők számára hasznos elemző alkalmazást, melyhez igyekezett napjaink eszközeit figyelembe venni. Microszervíz alapokon önálló konténerizált szervizek segítségével – melyek a későbbiekben rugalmasan bővíthetőek - alkotta meg az alkalmazást.

A feladat komplexitása, valamint a felhasznált eszközök is igazolják a Jelölt szoftverfejlesztési területen meglévő jártasságát.

A munkát védésre javaslom.

A konzultációk adatai:

Sor-szám	Dátum	A konzulens által elfogadott feladatrész	Megjegyzés, javaslat	Konzulens aláírása
1.	2022.01.27	Tartalomjegyzék, főbb struktúra	Elméleti fogalmak tisztázása	*Peck Tibor*
2.	2022.04.21	Elméleti rész rendben	Eszközök, azok leírása, fejlesztési szoftverek	*Peck Tibor*
3.	2022.05.06	Választott eszközök, szoftverek	Hálózati rész, kiegészítések, irodalomjegyzék	*Peck Tibor*
4.	2022.05.11	Az elkészült fejezetek	Leadásra kész.	*Peck Tibor*
5.				
6.				

A szakdolgozatot eredetiség szempontjából átvizsgáltam, plagizált tartalommal nem találkoztam:

Peck Tibor
konzulens aláírása

A szakdolgozat beadható:

Budapest, 2022. május 26.

Peck Tibor
konzulens aláírása

GDF
GÁBOR DÉNES FŐISKOLA

Gábor Dénes Főiskola,1119, Budapest, Fejér Lipót u. 70.
web.: www.gdf.hu   E-mail: info@gdf.hu

Nyilatkozat

a szakdolgozat eredetiségéről

Alulírott <u>BORDÁS OLIVÉR ODIN</u> <u>EC33NW</u>**,** ezennel büntetőjogi felelősségem tudatában nyilatkozom és aláírásommal igazolom, hogy a *Seeking alpha on the cloud: An SaaS solution for market analytics* 053/2022 szakdolgozatom saját, önálló munkám eredménye; az abban hivatkozott nyomtatott és elektronikus szakirodalom felhasználása a szerzői jogok és a szakdolgozatírás szabályainak megfelelően készült.

Kijelentem, hogy a plágium fogalmát megismertem és ahol mások eredményeit vagy gondolatait idéztem azt minden esetben azonosítható módon feltüntettem, a dolgozatban közölt ábrák és képek közlése mások szerzői jogait nem sértik.
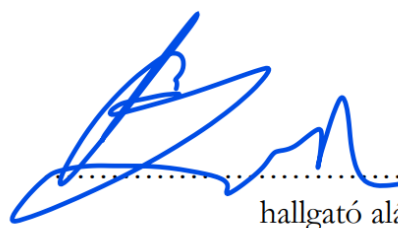
Budapest, 2022. év <u>május</u> hó <u>27.</u> nap

....
hallgató aláírása

Nyilatkozat

a szakdolgozatról

Alulírott <u>BORDÁS OLIVÉR ODIN</u> <u>EC33NW</u>**,** kijelentem, hogy **a** *Seeking alpha on the cloud: An SaaS solution for market analytics* 053/2022 szakdolgozatomat a Gábor Dénes Főiskola oktatói és hallgatói, szükség esetén más érdeklődők is felhasználhatják (pl. hivatkozás alapjául, olvasótermi használatra) későbbi munkájukhoz a szerzői jogok tiszteletben tartása mellett.

Budapest, 2022. év <u>május</u> hó <u>27.</u> nap

....
hallgató aláírása

Kivonat / Abstract

In the recent years, due to technical innovations, access to the stockmarket is easier than ever. The new tools of trade are created by fast growing fintech companies and startups, whose main focus is on user-growth and application experience, that not neccessarily benefits the end user on the long term.

Such platforms, combined with previously unseen quantitative easing, low interest rates, and soaring stock market will not motivate end users to educate themselves in investing, prepare for rainy days, and look for real value. The thesis and the software, called Market Analytics Platform (MAP), attemps to give a technical demo counter example to these trends.

With the use of modern technologies like Docker, microservices on a Kubernates cluster, Angular, gRPC, Swagger, Spring Boot it offers an SaaS solution that lets the users search for companies, analyse and compare them, and generate a report.

The thesis will also elaborate on development methodologies required for any large scale project, as well as dwell into the details of microservice architecture, provide a technical glossary to describe the used technologies in MAP and explain its development in detail.

Tartalomjegyzék

Introduction

In the recent years, we can see a significant increase of trading volumes on the most liquid market, the US stock market, generated by retail investors [1]. There are multiple definitions for a retail investor, according to the Cambridge Dictionary retail investors are:

„A member of the public who makes investments, not a large organization or business that makes investments." [3]

I am going to expand this definition with the following:

Retail investors are non-professional members of the public who make investments, with emphasis on the non-professional part.

Numerous studies [4][5][6] have looked into the overall performance of this type of investors and found that their vast majority is losing on the long term. Not surprisingly, considering their characteristics which was described as inexperienced, aggressive in terms of trading, speculative, their outperformance would be expected by sheer luck, and overconfident.

This segment is also more likely to engage in trading activity on the basis of non-fundamental information. During the dotcom bubble sometimes they responded to important news regarding certain companies by quickly buying similarly named but distinct companies. Such events are still not uncommon, it has happened recently when a misinterpreted social media posts caused a company value skyrocketing [1]. No wonder that the financial media coined this group as "dumb money", in comparison with the big institutional investors and mutual fund companies, named as "smart money". [7]

Their composition however, has changed over time. While in the late 1990s the mean retail investor was 50 years old and invested around $47,000, nowadays they are younger, around 31, and has less to invest, between $1,000 and $5,000.

With the crowd, the platform and communication landscape has changed as well. While in the 1990s, the online chatrooms, newsletters and forums were the main ways of communication and slow internet or expensive telephone brokerages were the common tools of trading, these days the social media platforms serve as the main way of communication, and the newly surfaced trading or financial technology (fintech) firms

such as Robinhood, Revolut or Wise are the ones who are channeling the huge amount of retail volumes into the market [2].

Yet the question arises, if retail investing holds such risks and underperformance, why is it on the rise?

Part of the answer lies in the new platforms, accompanied with the herd like behaviour generated by the fast information exchange on social media sites, however I rather intend to focus on the platforms now.

The 2010s brought huge technical innovations. In this decade smartphones and fast mobile internet speed became common in the developed world, which has also influenced -among others- the financial services industry. With the advent of fast and constantly innovating mobile operating systems it is also served as the breeding ground for new mobile application development technologies that allowed to create services that previously was existed only in brick and mortar form. Nowadays, we can open a bank account in some countries without ever stepping into a bank building, if such building exists at all besides the development office of the fintech company that has acquired a banking licence.

The same change has happened with brokerages. Fintech companies have challenged the "old world" brokerage structures; the ones that stopped innovating in the IT or weren't fast enough, have risked their own existence. The new challengers brought everything that this new generation of retail investors needed. With the new technologies they made access to the stock market as easy as never before, the registration process was quick, interactive and fun as new IT specialisations emerged such as UX Design that made the user engaged with the application as much as possible. The easy to use interface design, the commission-free execution - that forced the whole brokerage market to follow - became possible with the established new alternative income streams such as selling customer data, providing order flow/liquidity to exchanges for incentives. Even provided out of reach shares to the public via the opportunity to trade fractional shares, as well as simplifying options trading, which was due to its complex and abstract derivative nature, previously utilised much less by this segment.

Given all of these new and innovative features, no wonder that commercial success and user growth has followed.

"A Deutsche Bank survey found that almost half of US retail investors were completely new to the markets in the past year. They are young, mostly under 34. And they are aggressive: much more willing than those more experienced in stock markets to borrow to fund their bets, to make heavy use of options to fire up wagers on stocks, and to use social media as a research tool to find trading ideas."[2]

Retail investors can now invest without all the fuss, at the dormitory, on the tram, or over university lunch breaks on cleverly-designed smartphone apps. Easy access, however, comes with appalling consequences.

"Access to leverage — in the form of "margin" loans from brokerages or financial derivatives like options — is also freer than ever before. US margin debt soared to a record $799bn in January. "[2]

It seems that despite the losses, such features accompanied with government stimulus in the US, (from what - a recent survey responders tell - 37% will go into the markets [2]) are hard to beat, and the year 2021 showed that the synergy between these participants are working quite well.

The question is: what can we do if we have an aspiration for investing in the long term on our own but do not want to participate in the flow of the "pump and dump" herd?

Turns out several things, which I will explain in detail during later chapters, but all of these methods have the same foundation, namely a Market Analytics Platform (**MAP**), which is going to be the topic of this thesis. As a Software as a Service (SaaS), it can be accessible from anywhere, and due to its flexible microservice nature, it can easily be expanded with new features to help us generate alpha over the long term and hopefully raise us above the bar of dumb money.

MAP will be contained on its own cluster, (as the configuration of the infrastructure is also part of the thesis besides the softwares) and will be able to act just like a SaaS on the host machine. It will offer some simple financial analysis option and demonstrate a complex, scalable software solution and infrastructure with the use of many current modern technologies.

I will use a bottom-up approach in the development and consequently in my thesis.

First, I will introduce the possible development methodologies for collaborative development and the software's architecture, followed by a short presentation of the used technologies and the specification.

In the middle of the thesis, I will dive into the explanation of the software's development, then attempt to elaborate on the alternative solutions regarding the used technologies.

At the end, I am going to conclude my findings, and present a use case for the completed software.

Development Methodology

Among the very first steps of an application design, there is the need to choose a Software Development Methodology (SDM) framework, as choosing a proper methodology can increase the project success rate [8]. There are several frameworks, methods, to choose from, and it very much depends on the project itself, which framework would be best to use, or worst.

A development methodology helps the team working on a project by providing a collection of processes, techniques and toolkit that helps, organises and standardise the processes throughout the project lifecycle [8]. Such methods can be related to workflow management like Scrum, or can be a directly applicable development procedure like the Test Driven Development (TDD).

As I mentioned earlier, there are several frameworks, methods, to choose from. To be able to see the complete picture, I am providing the following image.
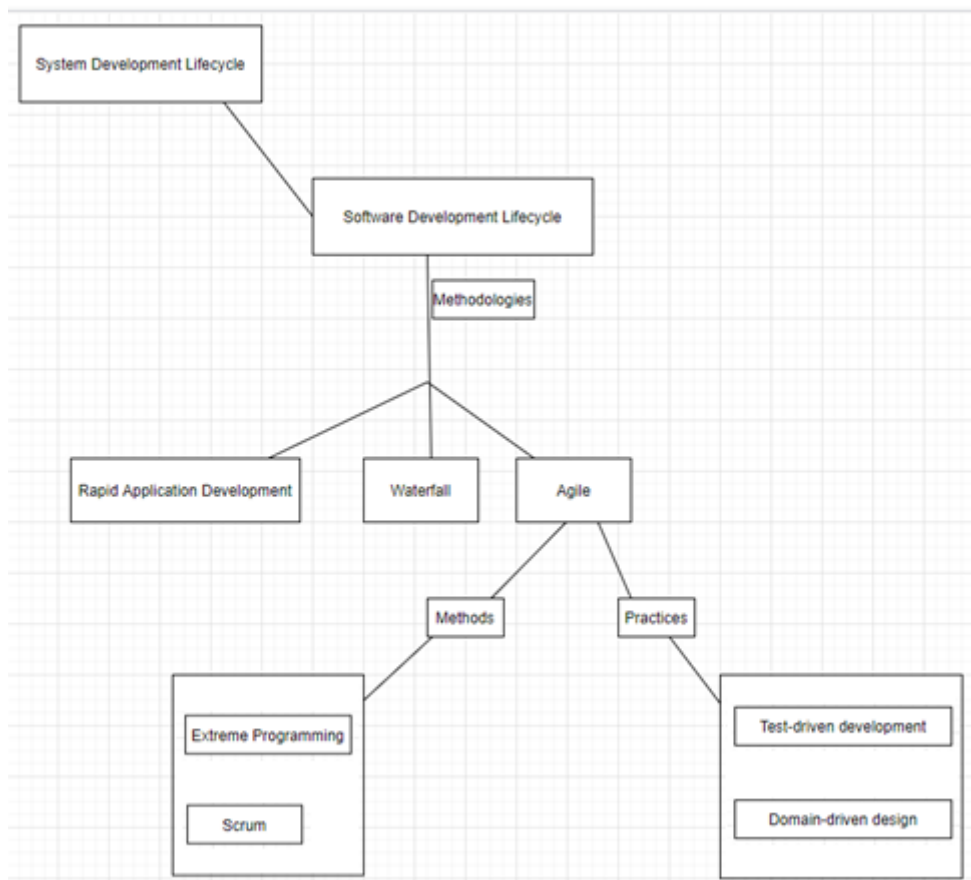
1. ábra. Development Methodologies
(source: self-edited illustration)

SDMs are all models that exist to support the Software Development Lifecycle (SDLC), while SDLC itself is a subset of the System Development Lifecycle. [9] SDLC can be described as:

"... process of building or maintaining software systems[7]. Typically, it includes various phases from preliminary development analysis to post-development software testing and evaluation. It also consists of the models and methodologies that development teams use to develop the software systems, which the methodologies form the framework for planning and controlling the entire development process." [10]

There are several SDLC models [9][11], Image I. shows the following:

- Rapid Application Development (RAD)

    The RAD method enables information systems to rapidly transition from design phase to completion, while keeping costs relatively low. The system is split up to smaller packages, which helps to make changes while the project is still in development. For each package, there are defined deadlines for delivery that should not be exceeded. The project's requirements can be reduced to match the deadlines.[8]

- Waterfall

    In Waterfall model, the software is split up into multiple, consecutive parts, where some overlap is tolerable between parts. The focus is on design, schedules, deadlines, budgets, along with the overall development of the whole software at the same time. With extensive documentation process, reviews and approval from the customer and the management are all needed at the end of most parts, therefore there is strict governance built into this model. [11]

- Agile

    Agile comprises practices and frameworks/methods that include requirements discovery and solutions improvement through the collaborative effort of self-organizing and cross-functional teams with their customer(s)/end user(s), adaptive planning, evolutionary development, early delivery, continual improvement, and flexible responses to changes in requirements, capacity, and understanding of the problems to be solved.[12] [13]

Based on my previous work experience, in case of a collaborative development, I would have chosen Agile as my preferred SDLC model. As shown, Agile consists of Frameworks/Methods and Practices. [14] [15]

The following are displayed on Image I above.

Methods

- Extreme Programming

    Extreme Programming (XP) is an agile software development model with the goal to improve the software's quality and its developer's life quality. In its core, it defines interconnected engineering practices for software development. These practices are advised to be done in conjunction as they can reinforce each other. Extreme Programming practices are like: The Planning Game, Small Releases, Metaphor, Simple Design Testing, Refactoring, Pair Programming, Collective Ownership, Continuous Integration, 40-hour week, On-site Customer Coding Standard. [20]

- Scrum

    Scrum is a model used to handle software development projects and other knowledge work.

    Scrum is practical in that it provides a frame for the development team to set up their own theory of how something works, test it, and then make a suitable adaptation according to the experience.

    Due to Scrum's structure, it allows other practices from different frameworks to be used within its context. [21]

Practices

- Test-driven development

    Test-driven development or TDD for short, is a style of programming where the following 3 actions are closely tied: Design (refactoring), Testing as unit tests, and Programming.

    It can be summarised as: [22]

    1. Create a unit test related to an aspect of the program.
    2. Run it and see it fail as that aspect has not yet been implemented.

3. Make the minimum amount of code that would make the test pass.

4. Refactor the code until it contains no duplication, pass the tests, has clear boundaries between its parts according to each part's responsibility, and uses the minimum required number of methods, classes, etc.

5. Repeat [22]

- Domain-driven design

    Domain-Driven Design or DDD for short, is a software development style that focuses on creating a domain model that has clear yet deep concepts about the rules and methods within the domain.

    A 2003 book by Eric Evans described the name of DDD and its approach with a list of patterns.

    Since then it has been developed further, and it works particularly well in complicated domains where a lot of logic needs to be organised. [23]

Based on my previous work experience, I have chosen to base the application design on the Domain-driven design principles. The DDD will provide a clear connection between the business domain and technical concepts.

Architecture

By deciding on an architecture, we are defining the skeleton of the application. It will set the boundaries and structure of the development. No complex application can be built without a clear architecture, just like no house can be built without a clear blueprint.

There is a vast number of architectural patterns to choose from. [16] The choice depends on the type of the application in development [16], and the type should be based on business understanding and the software's use cases.

Here are a few application types and their related architectural patterns:
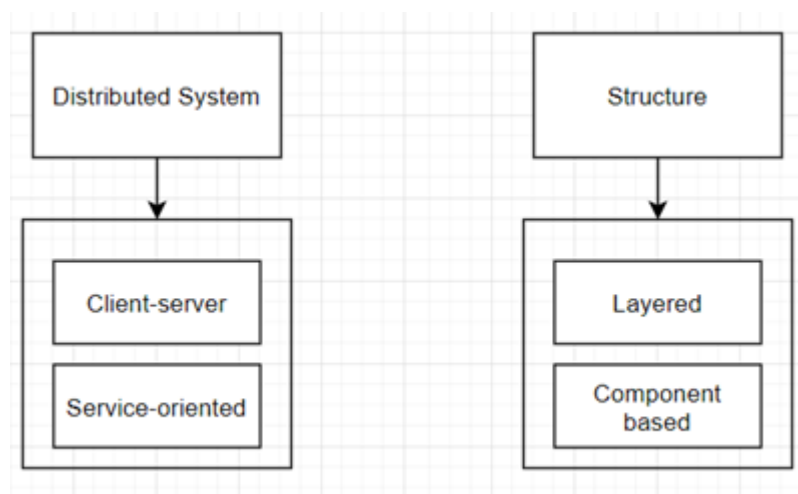
2. ábra. Application types and patterns
(source: self-edited illustration)

Distributed System type

Example: web applications, email

- Client-server

  Client/server architectural style details a distributed system that require an independent client and server, with a network between them. The most basic form of client/server architecture imply a software on the server that is reached directly by numerous clients. It is often called as a 2-Tier architecture. [16]

- Service-oriented

  In Service-oriented architectured applications, the functionality is delivered as a set of services, and the architecture allows the creation of applications that make

use of software services. The services are loosely coupled as they use interfaces that can be called, published, and discovered. [16]

Structured type

Example: Pluggable applications, UNIX, TCP/IP protocol suite

- Layered

   When in an application architecture we focus on grouping related functionality into separate layers that are built on top of each other, we have a so called Layered architecture. Each layer's logic has a common role/responsibility. The layers communicate with each other in a loosely coupled manner. The layers help to achieve a strong separation of concerns while providing maintainability and flexibility. [16]

- Component based

   In Systems Design, the Component-based architecture is describing a software engineering approach. With well-designed interfaces containing functions, variables, it comprises separate logical components. The emphasis is on achieving a higher level abstraction (compared to OO design) without spending too much time on state management and communication protocol issues. [16]

The recent trend of the past few decades is a shift towards modularization, loose coupling, and distribution. [17] This trend became more prevalent, as digitalization appeared throughout various industries and with that, there is an increasing need towards software quality and maintainability. [17] While softwares are growing more and more complex, reaching enormous amount of users, one step toward this trend is the use of Service-oriented Architecture (SOA).[17]

There are multiple subsets, approaches to a Service-oriented Architecture [18], here I am going to talk about one of them, which is going to be the choice of my architecture: Microservices.

Microservices

In the literature, there are several definitions available for Microservices:

"A microservice is a cohesive, independent process interacting via messages." [18]

"A microservice architecture is a distributed application, where all its modules are microservices." [18]

"Loosely coupled service in a bounded context " [18]

"Small autonomous services that work together, modelled around a business domain." [18]

"A small application that can be deployed independently, scaled independently, and tested independently and that has a single responsibility."[18]

Among these I am going to apply the last definition, when designing the architecture in my thesis. It contains all the -few, but important- basic principles of a microservice architecture:

- Bounded Context

  Related functions are combined into a single business process, and each microservice implements one such ability. In this way there is a perfect symmetry between business abilities and system composition, making it easy, for example to know where a function is, in order to update or fix it. [17]

- Size

  The focus on small size is a key component of microservices compared to the previous SOAs. Idiomatic use of microservice architectures suggests that if a service is too large, it should be broken into two or more services, thus preserving granularity and maintaining focus on providing a single business process only. The smaller size brings crucial benefits in terms of service maintainability and extendability: a small service can be simply changed, and if needed, rebuilt from the start with limited resources and in short time. [17]

- Independency

  Microservice encourages loose coupling and high cohesion by expressing that every microservice in microservice architectures is operationally free from others,

and the only form of communication between services is through their interfaces. This is foundational since this allows one to change, fix or upgrade a microservice without compromising the system correctness, provided that the interfaces are kept. [17]

In essence, microservices are small applications -independent softwares- that are interacting with each other.

Communication

When talking about communication in the Microservice context, first, we have to distinguish two types:

In-Process and *Inter-Process communication*. [24]

Communication between different services across the network are called *Inter-Process communication*, while communication within a service is called *In-Process communication*, and they are very different in nature. Among these differences I would like to highlight three, namely performance, handling change and errors. [24]

Performance

The main difference between the *In-Process* and *Inter-Process communication* performance is that while during *In-Process* calls, the compiler and runtime may be able to utilise several optimization techniques to make the call the most efficient, in *Inter-Process* calls such optimizations are limited, as here packets have to be sent. An illustrative example would be the following.

In an *In-Process* method call, when I pass a parameter to this method, this data structure doesn't move anywhere, as I am likely to pass just a pointer in a memory location.

In an *Inter-Process* call, like making calls among microservices on the network, the previously mentioned data structure would be needed to be serialized to be transmittable, and on the receiver side, it is needed to be deserialized, both which take time. Therefore we really need to think about the payload size. [24]

Compared to *In-Process* communication, utilising services that communicate in such a distributed environment in a demanding manner - meaning making several requests to each other across the network - are needed to be carefully created with efficient serialization mechanism and reduced amount of data usage. Even changing the whole data handling functionality might make sense performance wise. Like instead of taking the data as a parameter, it can also be offloaded into a file system for example. [24]

Handling changes

Changing an interface inside a single process -normally- has no additional burden on the whole rollout procedure. The interface is packed together with the caller, and usually a modern IDE even supports built-in refactoring abilities for changing method signatures. Such deployment can be done as one atomic step.

When such interface is used among several other independent services the solution is not so simple once it comes to changes. For example, a backward-incompatible change to a microservice interface would force us to use new techniques, such as lockstep deployment to handle the new contract rollout. [24]

Handling errors

The nature of errors is usually different in a distributed and in a single process environment.

In the latter, the errors are likely to be deterministic. Either they are expected and can be easily handled, or disastrous, in which case we just push the error up to the call stack.

Within a distributed environment there is a serious exposure of errors that are outside of our scope. For example downstream service outages, network timeouts, container crashes due to excessive memory consumption. [24]

In the book, Distributed Systems by Andrew Tanenbaum and Maarten Steen [25] distinguishes 5 types of failure modes within Inter-Process communication: Crash failure, Omission failure, Timing failure, Response failure, and Arbitrary failure. [24]

Below are the short descriptions of these modes:

- Crash failure

  Server is suddenly crashed.

- Omission failure

  Response is missing, or a downstream service for example stops firing events.

- Timing failure

  Something has happened too early or too late.

- Response failure:

  Incorrect response received.

- Arbitrary failure

  Situation when something went wrong but the participants cannot agree if the failure has occurred or why. [24]

Some of these errors might go away easily, but some might be persistent. Therefore in distributed systems we need a richer set of returned error descriptions, for example HTTP is a protocol that well understands this requirement. [24]

Inter-process communication technology overview

There are several styles of inter-microservice communication, with different technologies.
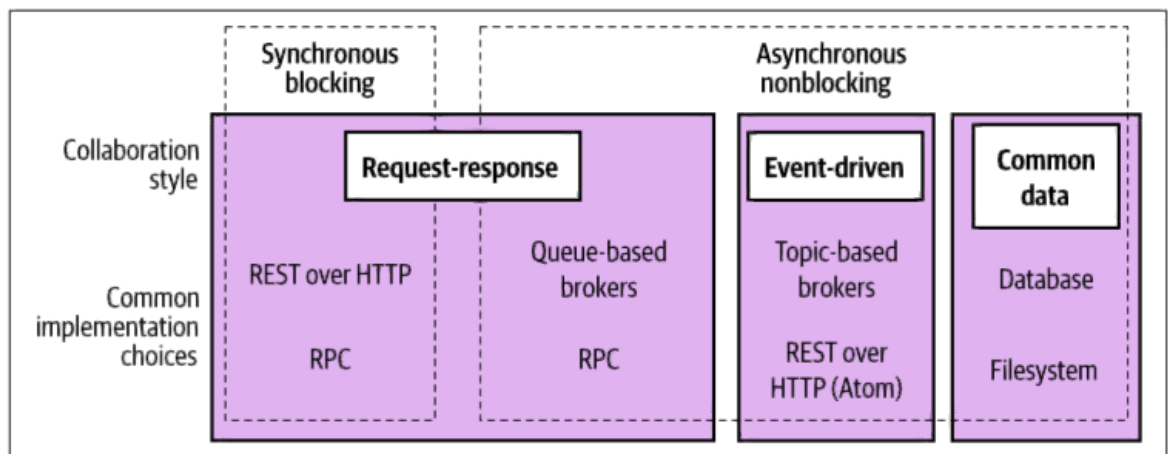
3. ábra. Inter-process communication styles
(source: [24] page 94.)

- Synchronous blocking

    The operations are blocked while waiting for a response.

- Asynchronous nonblocking

    The operations are continuing after the call has been made without waiting for a response.

- Request-response

    After the call, we are expecting the result in a response.

- Event-driven

    Events that might or might not be consumed by other services are emitted. The event emitter has no knowledge of the consumers.

- Common data

    A shared data source used among services.

There are several things to consider which style is preferable for a given project. Choices like: preferred technologies such as message brokers, or whether there are specific latency targets to achieve, or the need for ability to scale, even security related aspects can influence the decision.

For the Market Analytics Platform project, I will utilise synchronous communication, in two different ways:

I will use REST over HTTP technology which is a request-response collaboration style, to communicate with the frontend microservice that will essentially run outside the microservice cluster, in the browser as it will be an Angular application.

For inter-process communication I will use gRPC, a high performance Remote Procedure Call (RPC) framework that uses HTTP/2 for transport, and Protocol Buffers as the interface description language.

Before I introduce the Market Analytics Platform project's architecture, I would like to write about two common technologies associated with the microservice world, namely:

Service Mesh, API Gateway and an implementation technique named Saga for business processes spanning through multiple services. [24]

Service Mesh

Simply put, a service mesh holds the common functionalities of inter-service communication.

This approach comes with two benefits:

- Reduces the code needed for microservices to implement themselves.
- Consistency of common functionalities.

For example, service discovery or load balancing are such functionalities that are common within a service mesh. Alternatively, these logics can be used within a library, however, the disadvantage would be the increased maintenance time as each microservice must be updated with each new version of the library, as well as issues arising in the meantime when separate microservices are using different versions of the library. [24]

A service mesh is utilising proxies (so-called sidecar proxy or data plane) next to each service. A single business process can trigger multiple inter-process (or "east-west" in data center lingo) calls. Such amount of calls can place a heavy load on the proxies, and service mesh implementations race to limit the impact while seamlessly providing the above stated functionalities. A control plane is also part of the service mesh, it acts as a place where the proxies are managed, where their behaviour can be changed, as well as collecting information about them. [24]

API Gateway

API Gateway focuses on handling requests coming from external sources into the microservice environment, in other words dealing with "north-south" traffic.

They mainly act as a reverse proxy, in addition to providing for example, logging services, rate limiting, or servicing API keys to external parties.

API gateway is for projects where a huge number of external parties is expected to use our API directly, for example a data provider software or a headless CMS platform. [24]

Using an API Gateway for softwares that are not part of this "API Economy" (or have their own GUI clients to access their microservices) would be just an increase to their complexity and making them more vulnerable and error prone by implementing this feature.

Besides some technical reasons (for example in case of Kubernetes, a simplified API Gateway implementation is needed as it cannot handle the communication to and from its cluster by itself) there have to be a careful analysis before using it. [24]

Distributed transactions

Implementing business logic that spreads through multiple services can be tricky by itself and even more in a distributed environment.

In such cases we would like to achieve that all changes occur as one single unit, meaning either all changes complete successfully or no changes occur at all. Most of the time this requirement points us towards the use of a database transaction. [24]

In a database transaction we can be sure that part of the CRUD (specifically the **c**reate, **u**pdate and **d**elete) operations are completed successfully, on multiple tables as well. The key properties for a database transaction can be summarized in the ACID acronym. [24]

In short, ACID stands for:

- Atomicity

  Means that all operations within a transaction either all complete or fail, leaving no interim states in the database.

- Consistency

  All changes in the database keep the data's integrity, or it doesn't happen at all.

- Isolation

  Multiple transactions can run at the same time without any issue as each transaction's non-committed changes are not invisible for the others.

- Durability

  In case of a system failure we can be sure that committed data remains in the database, intact.

Business processes that occur within a single microservice environment involving only the microservice's own database (or if multiple microservices are working on the same database at once) can easily achieve ACID properties. [24]

The outlook is a bit different if the business process has to be implemented across microservices that each has its own database. In this case, we have to split up an otherwise

single transaction into two or multiple ones. In such multiple depending transactions treated as one single business operation, we have lost the atomicity property on the whole operation. One common solution that is trying to solve this problem is the use of a distributed transaction, more specifically a two-phase commit implementation. [24]

Two-phase commit - 2PC

Two-phase commit is a distributed transaction algorithm that helps in case multiple separate processes need to make transactional changes in a distributed system. A two-phase commit -as its name says- consists of two phases:

- Voting phase
- Commit phase

There is a so called central coordinator mechanism involved which is handling these stages. In the voting phase, the central coordinator asks the workers involved with the transaction whether or not the state changes can be made. If all workers confirm that the state changes can occur, the commit phase comes next. If any worker says that there is a problem, for example a constraint violation, and the change cannot occur, then the entire operation aborts and no changes are made. [24]

The only way a worker can safely say that the change can occur in the database, is through the use of a locks. In a small scale operation, that involves a few tables and happening rarely, this cause no harm. However, in a more resource intensive scenario, meaning a business transaction that involves many tables and spans through several minutes, or even hours, days, or just simply occurring multiple times in a very short timeframe, it can cause vast amount of errors that are hard to take care of. [24]

For example, as there can be latency between the participants of the transaction, meaning part of the workers had already committed their transaction yet another part of the workers not, that means on the whole single business operation we have lost the ACID's isolation property. As workers need to acquire and hold locks until the second phase, it can lead to several issues regarding lock management, and deadlocks. Problems related to workers, such as a worker is not responding to the commit request from the central coordinator, can lead to troubles that need to be handled manually by an engineer. [24]

For these reasons, it is generally not advised to implement two-phase commits, or use it just for rarely occurring, short living operations. There are some well implemented distributed transactional algorithm implementations that can be used successfully on large-scale databases, for example Google's Spanner. However, for Google to succeed in this area it had to, for example, invest in very expensive datacenters and use satellite-based atomic clocks, which on its own shows that to achieve such a feat correctly is not a simple business. [24]

However, if we still have the need to manage large business transactions spanning through multiple microservices and their databases, there is a better alternative: Sagas. [24]

Saga

At its core, a Saga is no more than an algorithm that is coordinating multiple state changes while avoiding the need to lock resources for longer than needed periods of times. Designing a Saga involves modelling the whole business process as discrete steps that are executed one by one. Effectively we are creating a sequence of transactions. Each of these transactions by itself complies the ACID properties, as it happens within the respective microservice's database, while the Saga itself does not. [24]

In case a failure happens, there are three common recovery modes.
- Backward recovery
- Forward recovery
- Mixed recovery

Backward recovery is basically the standard rollback function steps, such as reverting and cleaning up all the failed states. Here, we create compensating transactions that help to revert previously committed processes. In case of a forward recovery, the algorithm starts over at the point where the failure has occurred, and keeps executing from there. As its name suggests, the third option involves both backward recovery and forward recovery processes. A valid scenario could be that after a certain number of tries of the forward recovery algorithm, if the an error keeps occurring, it switches to the backward recovery algorithm to revert all previously committed states.[24]

It is important to point out that Saga can only recover from business failures and not technical issues. A business failure is for example: at a merchant's system the reserved stock's product id in the warehouse database does not exist in the accounting database.

However, if the accounting service throws an Internal Server Error because the accounting service bean was not populated by the dependency injection framework due to missing default constructor signature in its class, that's an issue that needs to be handled on its own, a Saga cannot recover from that. In that sense, the Saga is expecting to work with proper components, but in case we would like to handle the technical errors as well, there are some solutions that builds on a Saga, effectively creating a layer above it to handle such technical failures. [24]

Saga implementations

There are two types of Saga implementations:
- Orchestrated Saga
- Choreographed Saga

Orchestrated Saga is using a central mechanism (== orchestrator) to conduct the business transactions and issue a compensating order if required. This central orchestrator can become an issue later on, as it introduces domain coupling as well as it might take on logic that otherwise should have been placed inside a service, basically centralizing the logic. [24]

Choreographed Saga on the other hand tries to distribute the responsibility between the affected services, effectively crating a reactive environment. The events are circulating in such a system, and only affected services reacting to them, allowing more room for parallel processing and this way might make the business transaction even faster. Usually a message broker handling the events around the system for example RabbitMQ, or Apache Kafka. Due to the event-driven nature of this type of Saga implementation, the services don't know about each other, therefore the domain coupling is greatly reduced, compared to Orchestrated Saga implementation. [24]

At the same time, in such a decentralized system, it can be hard to know in case of an error what has happened and how it happened. A simpler solution for that is the use of a unique correlation id for the Saga that is passed around for that Saga's events. This way, with logging process and a specifically designed service with compensating transaction algorithms can then put together the business transaction, then detect and correct from the point where it has failed by issuing the appropriate reverting transaction, in case the originally involved services were not able to do it. One way or another, in case of a choreographed Saga, the ability to trace the Saga transaction events is imperative. [24]

Market Analytics Platform Architecture and Technology

The following picture shows the architecture of MAP:
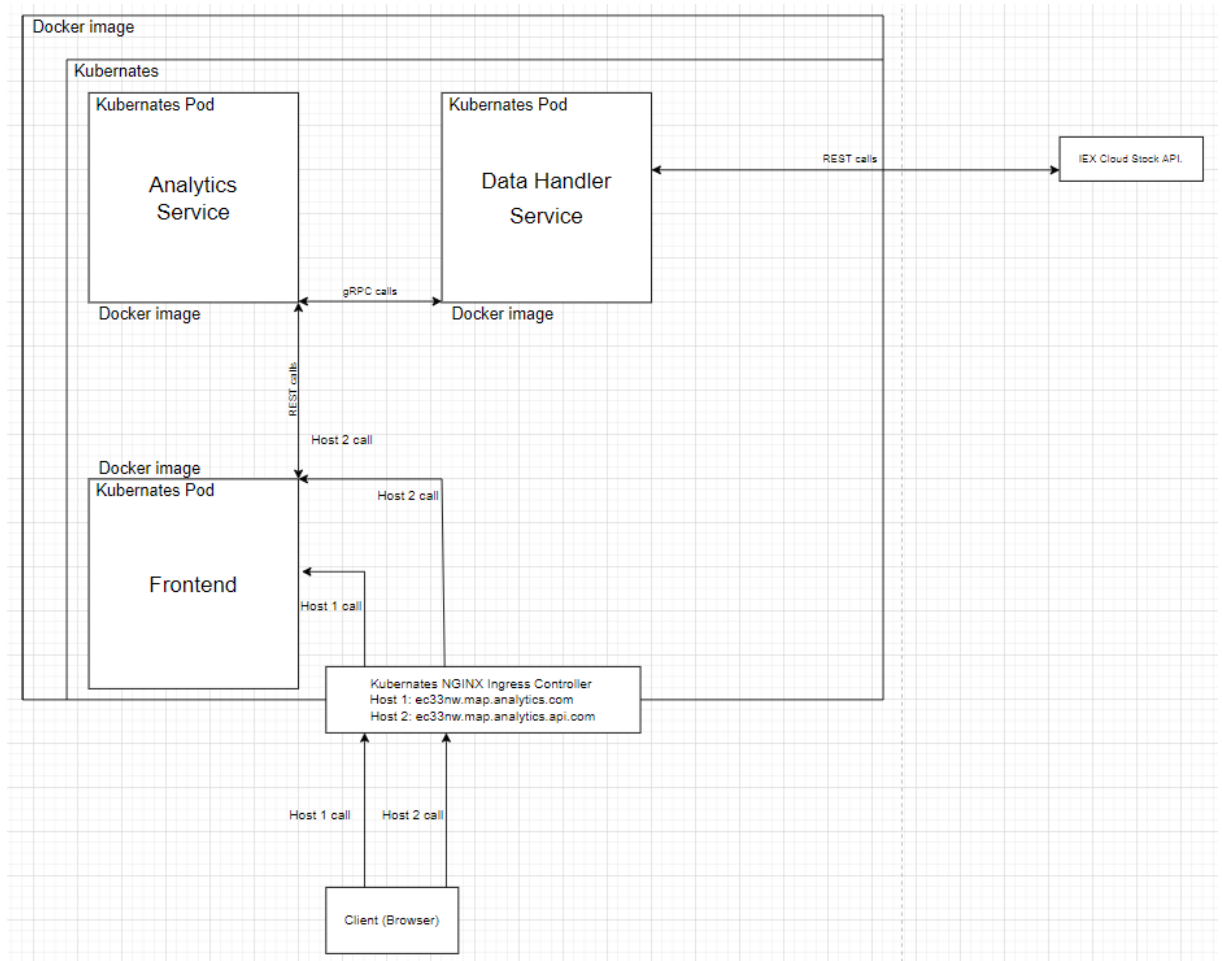
4. ábra. MAP architecture
(source: self-edited illustration)

Specification - Functional description

The Market Analytics Platform will consist of 3 microservices:
- Datahandler
- Analytics
- Frontend or MAP-GUI

Datahandler

It is responsible for all tasks related to data integrations. Currently, it will integrate only with the IEX Cloud API, through a 3rd party client library:

https://github.com/WojciechZankowski/iextrading4j

Datahandler, and so the MAP will support 5 methods:

- getSymbols
  This method is responsible for getting the traded companies ticker symbols from IEX Cloud API. Due to the large amount of data returned, the result list is limited to the first 101 symbols.
  Input: None
  Output: List of Symbols

- getBalanceSheets
  This method is responsible for getting the last 4 balance sheets of the requested company from IEX Cloud API. As currently we only connect to the free test data service of IEX Cloud, only a list of 1 item will be returned.

  Input: Symbol of the requested company
  Output: List of Balance sheets, wrapped in an object.

- getIncomeStatements
  This method is responsible for getting the last 4 Income statements of the requested company from IEX Cloud API. As currently we only connect to the free test data service of IEX Cloud, only a list of 1 item will be returned.

  Input: Symbol of the requested company
  Output: List of Income statements, wrapped in an object.

- getCashflowStatements

  This method is responsible for getting the last 4 Cash flow statements of the requested company from IEX Cloud API. As currently we only connect to the free test data service of IEX Cloud, only a list of 1 item will be returned.

  Input: Symbol of the requested company

  Output: List of Cash flow statements, wrapped in an object.

- getPeers

  This method is responsible for getting the tickers of the requested company's competitors.

  Input: Symbol of the requested company

  Output: List of company symbols that are the input company's competitors.

The Datahandler acts as the datasource of the MAP (from MAP point of view), so it should also be the host of gRPC server in the cluster.

It must provide the above stated functions inside the cluster as gRPC methods, and also handle the data conversions from the POJOs coming from the IEX Cloud API to gRPC objects for other services inside the cluster via custom mapping.

As a standalone microservice, it should have its own Dockerfile, as well as the relevant Deployment and Service configuration .yaml files for the cluster.

Analytics

It is responsible for data analytics, and the integration with the Frontend. Currently it will only support Frontend integration. It should provide a REST API towards the Frontend, with the same methods as described in Datahandler service, and it must be able to provide an OpenAPI specification of these services to make the client side code generation possible on the Frontend.

On request, the dataflow should go from the REST API to the gRPC client stubs to request data from the Datahandler service, and in turn from the IEX Cloud API.

Upon the data return form the gRPC client stubs, the Analytics service must use Mapstruct to map the gRPC data to POJOs, and return these pojos via the REST API to the Frontend.

As a standalone microservice, it should have its own Dockerfile, as well as the relevant Deployment and Service configuration .yaml files for the cluster.

Frontend

It is responsible for requesting and displaying data, and providing an interface to the user.

Design description:
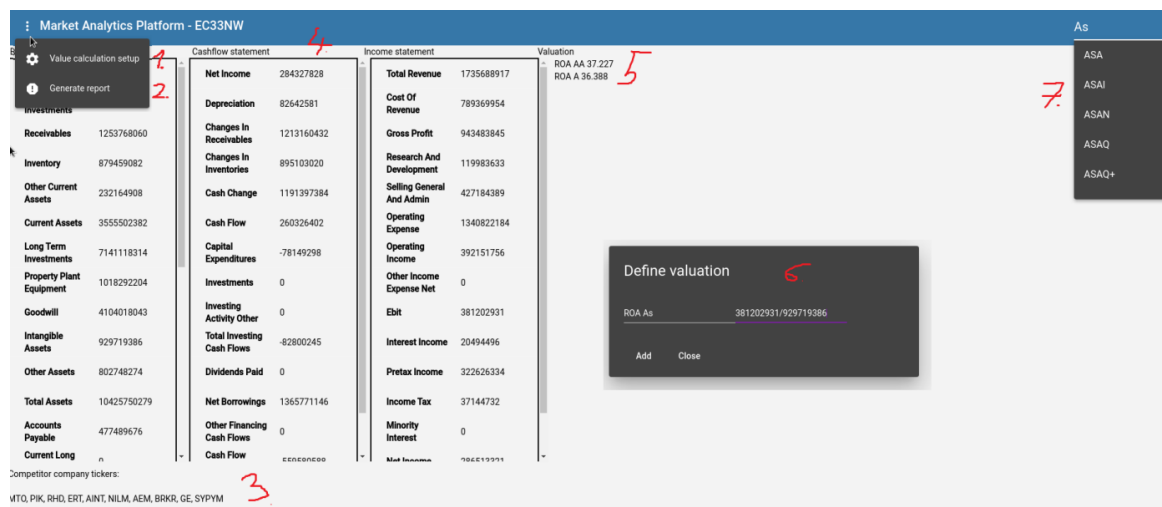
5. ábra. Design description
(source: self-edited illustration)

Frontend should provide the following 7 marked functionalities:

1. Have a toolbar with 2 menu items, 1. To open valuation definition, 2. To generate report.
2. Generate a report that shows all information presented on the MAP screen.
3. Display competitor company symbols.
4. Display the searched company's Cash flow, Income, and Balance statements.
5. Have a place where the recorded valuation metrics are stored.
6. By opening the valuation definition dialog, it should allow edit the background without closing itself, should not close itself unless explicitly clicked on its Close button. In this way, after multiple company searches are run, the dialog box will remain open, and it doesn't need to be opened over and over again. It makes entering different company valuations into the screen easier. By clicking the Add

button it should execute the basic mathematical operations displayed in its right field. Should be freely draggable across the screen.

7. Search field with autocomplete, should provide all (101) searchable symbols. By writing a symbol letter into it, it should only display relevant symbols, such as the ones that are containing the supplied letters in them. Should execute search when selecting a company ticker and hitting enter.

Frontend's API clients must be generated with Swagger from the Analytics service. The Frontend must use Angular Material components, and resemble to the design plan.

As a standalone microservice, it should have its own Dockerfile, as well as the relevant Deployment, Service and Ingress configuration .yaml files for the cluster.

Technologies

The whole MAP will be contained in a dockerized Kubernates cluster called Minikube. Each service will have its own Docker image, available on my Dockerhub:

https://hub.docker.com/u/zedas

The Analytics and Datahandler services will run on an openjdk:8 base image, while the Frontend will be based on nginx:alpine image.

The Analytics and Datahandler services will be microservice implementations in Java 8, and Spring Boot will be used as a DI framework. Both services will be built by Maven and they will use its dependency management feature to resolve the required gRPC, Sprint Boot, Swagger, Mapstruct, and IEX Cloud Client API dependencies and plugins. Eclipse will be used as an IDE.

Frontend will be written in Angular. I used Angular Material as the component library. Angular Routing will be enabled to ease page navigation development in the future, and Swagger is utilised to generate the REST API clients from the Analytics service. The services handling the REST API clients will use RxJS to manage asynchronous request/responses. Visual Studio Code will be used as an IDE for the development of Frontend, with npm as the package manager.

Each microservice will have their own Dockerfile and.yaml Kubernetes descriptor files. To interact with the cluster, pods, and the images, kubectl and Docker commands will be used. The whole development will be done on an Amazon Linux 2 machine hosted by

AWS, as a Desktop as a Service, since my own PC is not strong enough to accommodate these technologies together.

In the following segment I will provide a short explanation of the above mentioned technologies, then I will proceed to explain the MAP development steps.

Technical Glossary

Kubernetes

"Kubernetes is a portable, extensible, open source platform for managing containerized workloads and services, that facilitates both declarative configuration and automation."

[26]

The need for a platform to manage multiple independent applications appeared as a direct result of technological innovation and the rapid growth of distributed applications.

First, organizations ran their software directly on physical servers, however this often led to a point where one application was taking up most of the resources, while the rest were starving. There was no way to define resource boundaries, and the only solution was to run each application on its own server, which was very expensive, and the machines were vastly underutilized. [26]

Then came the virtualized deployment era, where each physical server had multiple Virtual Machines (VM). The VM provided high level of security, as each VM had their own defined resource boundary, and information cannot be accessed from the host machine or other VMs freely. However, each VM is a fully equipped machine with its own operating system, running all components, on top of a virtualized hardware, and all of this were still too resource intensive. [26]

So we arrive at the latest "era" of containerized deployment, where instead of a fully fledged VM, we use lightweight containers to run our applications. Containers are considered lightweight as they have relaxed isolation properties to share the Operating System, compared to VMs, but they still have their own separate resources such as share of CPU, memory, etc. Since they are decoupled from the underlying platform, they are easily portable. [26]

For complex environments, where a physical server can host hundreds or thousands of containers, there will be a need for a platform to manage them, that includes but not limited to: automatic failover management, scaling, service discovery and load balancing, storage orchestration, automated rollouts and rollbacks, automatic bin packing, self-healing, secret and configuration management.

Kubernates provides these services and more on its platform. [26]

kubectl

kubectl is a command line interface that uses the Kubernates API server.

Kubernates API server exposes an HTTP API that lets users, different parts of the cluster, and external components to communicate with each other, as well as lets the user to manipulate the state of the objects within the cluster, such as Pods, Namespaces, etc. [27]

Kubernates Nodes and Pods

A node on a Kubernates cluster behaves just like a physical or virtual server. A node can contain one or many pods, which in turn can contain one or many containers. A pod is the smallest unit that can be deployed on a Kubernates cluster. [28]
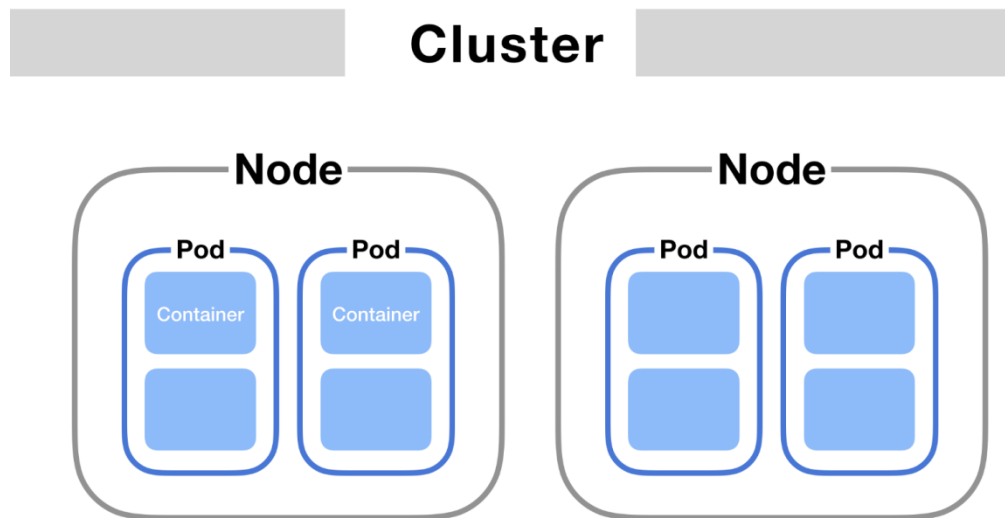
6. ábra. Cluster
(source: https://matthewpalmer.net/kubernetes-app-developer/articles/networking-overview.png)
(download date: 2022.05.24.)

Minikube

Minikube is a local Kubernates software, running on a Docker image. [29]

Istio

A service mesh implementation.

For more information, visit: https://istio.io/

API

An **A**pplication **P**rogramming **I**nterface provides a standardised (by its author) connection between softwares to interact with each other.

Docker

Docker defines itself as a platform to develop, run and ship applications. It allows the users to package and run an application in a loosely isolated environment, known as containers. [30]

Further information on containers can be found above, under the Kubernates section.

Via a file, called Dockerfile, Docker lets us build Docker Images, which is a standalone, executable software package that can be used to start-up a container.

A Dockerfile basically describes the start-up procedure (commands) of a container, as well as the Base Image. (All Docker Images must be based on another Image, called Base image. That also implies that Docker Images can be built on top of each other, constructing complex Docker builds.)

Out of the Docker Image, containers can be started within the Docker environment.

The containers can be entered from the host machine via command line (bash/sh), and as they are separate, standalone environments, they need to be configured for example to have network access inside the container.

To communicate with the application inside the container, we can map ports between it and the host, as well as mount shared volumes to the container from the host, so we can have a dedicated space where files can be shared between them.

The containers are managed by a persistent process called dockerd.

MAP's microservice Docker Images named as:
  Frontend: zedas/ec33nw-map-gui
  Analytics service: zedas/ec33nw-map-analytics
  Datahandler service: zedas/ec33nw-map-datahandler

Dockerhub

Dockerhub is a repository service hosted and provided by Docker. We can create our own private or public repositories, and download (pull) and upload (push) our and other's Docker Images.

All 3 service images of the MAP are stored here, publicly:

https://hub.docker.com/u/zedas

Nginx

At its core, it is a web server but has many more use cases, such as reverse proxying, caching, load balancing, etc.[31]

For MAP, it serves as the Base Image of the Frontend, as it hosts the Angular build files, and act as a standard webserver.

Java 8

The 8th version of Java, a high-level, class-based, object-oriented programming language.

MAP has been developed in this version, as the 8th version -despite it is 8 years old- is still dominating the corporate landscape [32].

Spring Boot / Spring

Part of the Spring ecosystem which provides various solutions for enterprise Java application development. The core concept of Spring is the convention over configuration model, the dependency injection (DI), and for Spring Boot the possibility of quickly bootstrap an application via Spring Initializr, together with the preconfigured, built in Tomcat server provides a way to quickly deliver a MVP, focusing on the actual problem to solve not the surrounding issues.

MAP's Analytics and Datahandler are utilising the Spring Boot features, such as DI, or the Tomcat server.

For more information, visit: https://spring.io/projects/spring-boot

Maven

Apache Maven is used as a build and plugin/dependency management tool for MAP. Its project object model file (pom.xml) can act as the central descriptor for the project and the software development processes involved with it.

For more information, visit: https://maven.apache.org/what-is-maven.html

gRPC

Stands for **g**RPC **R**emote **P**rocedure **C**all, it is a modern open source high performance RPC framework that can run in anywhere. It can efficiently connect services, and has pluggable support for load balancing, tracing, authentication, etc. [33]

A client application can call a function directly on a different machine wherethe server application is running, just as it would call on a local object. It is especially helpful in case of distributed applications, such as microservices. [34]

gRPC uses Protocol Buffers by default as the Interface Definition Language. It is describing both the service interface and the structure of the messages. [35]

Protocol buffers are defined as:

> *„Google's language-neutral, platform-neutral, extensible mechanism for serializing structured data – think XML, but smaller, faster, and simpler."[36]*

OpenAPI / Swagger

Open API is an API description format for REST APIs. It is essentially a specification written in YAML or JSON that can describe the API's endpoints, I/O parameters, authentication methods, etc.

Swagger is a toolset built for OpenAPI that helps to build or consume such REST APIs, for example it can generate the API's client side code. [37]

MAP's Frontend uses such generated client code from the Analytics service swagger-enabled REST APIs.

Mapstruct

Mapstruct is a code generator tool that helps the mapping implementation between Java bean types [38]. In the MAP, the Analytics service uses it to convert between gRPC generated objects coming from the Datahandler, to POJOs moving towards the Frontend.

On the other hand in Datahandler, I wrote a custom converter between POJOs coming from the IEX Cloud API, to gRPC objects moving towards the Analytics service.

IEX Cloud Client API

IEX Cloud is a financial information provider that MAP utilises as the source of its data [39].

As the access of the real data is only free for up to a certain limit, MAP uses a special token that allows the access to all REST endpoints for testing purposesbut the data returned aretest/dummy data. This test-mod is available to all registered free-tier users on IEX Cloud's platform.

For the client API to IEX Cloud, MAP's Datahandler service using a third party implementation from GitHub: https://github.com/WojciechZankowski/iextrading4j

Angular

Angular is a platform based on Typescript, for developing single page applications, and also an application design framework [40]. MAP's Frontend is built on Angular via Typescript, and the standard HTML/JavaScript/CSS.

Typescript

Strongly typed superset of JavaScript.

For more info, visit: https://www.typescriptlang.org/

Angular Material

Design components for Angular, used by MAP's Frontend.

For more info, visit: https://material.angular.io/

npm

npm (in all small cases) is a JavaScript Package Manager that allows to download a vast amount of software packages via its Command -Line Interface. Used by MAP's Frontend.

For more info, visit: https://www.npmjs.com/package/npm

RxJS

RxJS stands for Reactive Extensions Library for JavaScript, to develop asynchronous and event-based programs.

Used by MAP's Frontend.

For more info, visit: https://rxjs.dev/guide/overview

Git

Software for tracking changes in any file.

Used by the entire MAP project.

For more info, visit: https://git-scm.com/doc
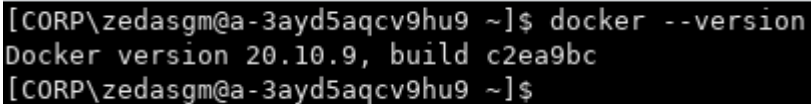
MAP Implementation, issues and future development goals

The base operation system is an Amazon Linux 2, hosted by AWS. First, the environment needs to be set up, in the following order:

Docker install & verify

```
sudo yum install docker
```
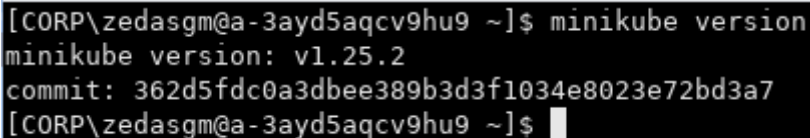

7. ábra. Docker verify
(source: self-edited illustration)

Minikube install& verify

```
curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-latest.x86_64.rpm
sudo rpm -Uvh minikube-latest.x86_64.rpm
```
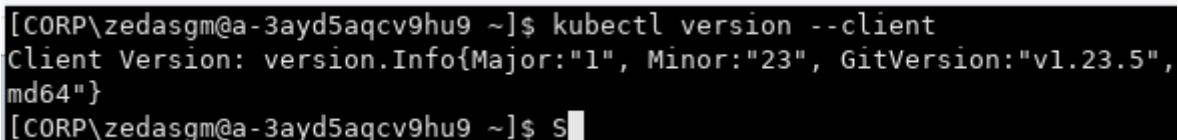

8. ábra. Minikube verify
(source: self-edited illustration)

kubectl install & verify

```
curl -LO "https://dl.k8s.io/release/$(curl -L -s
https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"
```

```
sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl
```


9. ábra. kubectl verify
(source: self-edited illustration)

Java JDK 8, NodeJS, Git, Maven, Eclipse, Visual Studio Code download and install is straightforward, therefore I am omitting them. The npm CLI is automatically installed with NodeJS by default.

For Java and Maven it was important to set up the required environment variables, and I also added a simplified "k" alias for kubectl, as it will be used frequently.

All set in the /home/zedasgm/.bashrc file.

10. ábra. .bashrc file
(source: self-edited illustration)

Angular install & verify

```
npm install
```
```
npm install -g @angular/cli
```


11. ábra. Angular version
(source: self-edited illustration)

OpenAPI Generator install

```
npm install @openapitools/openapi-generator-cli -g
```

Docker setup, and issues

With Docker I have encountered 2 main issues,

- After starting dockerd as a normal user, and trying to call commands, I got:

  docker: Got permission denied while trying to connect to the Docker daemon socket.

  The solution was to give permission to the docker.sock:

  ```
  sudo chmod 666 /var/run/docker.sock
  ```

- After starting a container, the container had no access to the internet.

  I have entered into the container via:
  ```
  docker exec -it <containername> bash
  ```
  ```
  ping  google.com
  ```
  and received timeout.

  The solution was to start the dockerd with the following command:

  ```
  sudo dockerd --dns 8.8.8.8
  ```

  This way, the containers have access to the internet which is especially important for deployments on Minikube, as it was setup to pull the images from a container repository. Without internet access, upon creating my pods I received ImagePullBackOff error.

After this setup, Minikube can be started with the command:
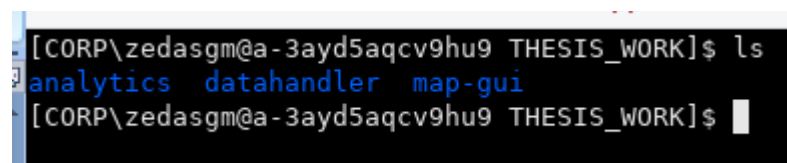
```
minikube start
```
Pods can be viewed with the command:

```
k get pods
```

The development will be done in the /home/zedasgm/THESIS_WORK folder.

3 project folders are created, each named according to the service it is going to hold.

12. ábra. Work directories
(source: self-edited illustration)

45

For Analytics and Datahandler, the project is a Maven project type in Eclipse, while with MAP-GUI, after I created the folder and opened it in Visual Studio Code, I had to issue the command:

```
ng new map-gui
```

to create the Angular project.

Analytics and Datahandler initial pom.xml config created with Spring Initializr (https://start.spring.io/) which allows to create Spring Boot projects quickly.

Datahandler development, issues, and areas to improve

Datahandler pom.xml excerpt I./II., only showing the relevant dependencies, with comments describing their use:

```xml
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <!-- IEX Cloud API wrapper START -->
    <dependency>
        <groupId>pl.zankowski</groupId>
        <artifactId>iextrading4j-all</artifactId>
        <version>3.4.5</version>
    </dependency>
    <!-- IEX Cloud API wrapper END -->
    <!-- gRPC START -->
    <dependency>
        <groupId>io.grpc</groupId>
        <artifactId>grpc-netty-shaded</artifactId>
        <version>1.45.1</version>
        <scope>runtime</scope>
    </dependency>
    <dependency>
        <groupId>io.grpc</groupId>
        <artifactId>grpc-protobuf</artifactId>
        <version>1.45.1</version>
    </dependency>
    <dependency>
        <groupId>io.grpc</groupId>
        <artifactId>grpc-stub</artifactId>
        <version>1.45.1</version>
    </dependency>
    <!-- gRPC END -->
</dependencies>
```

13. ábra. Datahandler pom.xml excerpt I/II.
(source: self-edited illustration)

These dependencies are important for the gRPC configurations as well as for providing the required IEX Cloud API client code.

Datahandler pom.xml excerpt II./II., only showing the relevant plugins, with comments describing their use:

```xml
<build>
    <!-- gRPC START -->
    <plugins>
        <plugin>
            <groupId>kr.motd.maven</groupId>
            <artifactId>os-maven-plugin</artifactId>
            <version>1.7.0</version>
            <executions>
                <execution>
                    <phase>initialize</phase>
                    <goals>
                        <goal>detect</goal>
                    </goals>
                </execution>
            </executions>
        </plugin>
        <plugin>
            <groupId>org.xolstice.maven.plugins</groupId>
            <artifactId>protobuf-maven-plugin</artifactId>
            <version>0.6.1</version>
            <configuration>
                <protocArtifact>com.google.protobuf:protoc:3.19.2:exe:${os.detected.classifier}</protocArtifact>
                <pluginId>grpc-java</pluginId>
                <pluginArtifact>io.grpc:protoc-gen-grpc-java:1.45.1:exe:${os.detected.classifier}</pluginArtifact>
            </configuration>
            <executions>
                <execution>
                    <goals>
                        <goal>compile</goal>
                        <goal>compile-custom</goal>
                    </goals>
                </execution>
            </executions>
        </plugin>
        <!-- gRPC END -->
        <!-- Remove lombok START -->
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
            <configuration>
                <excludes>
                    <exclude>
                        <groupId>org.projectlombok</groupId>
                        <artifactId>lombok</artifactId>
                    </exclude>
                </excludes>
            </configuration>
        </plugin>
        <!-- Remove lombok END -->
    </plugins>
</build>
```

14. ábra. Datahandler pom.xml excerpt II/II.
(source: self-edited illustration)

These plugins enable to generate gRPC classes on the fly to the target folder, while the user is editing the Protobuffer config file.

Protobuffer file

```
syntax = "proto3";
option java_multiple_files = true;
option java_outer_classname = "IexCloudProto";
option optimize_for = SPEED;
package iexcloud.gen;


service IexcloudService {
  rpc GetSymbols (NoParam) returns (stream Symbol);
  rpc GetBalancesheets (Symbol) returns (BalanceSheetsGrpc);
  rpc GetIncomeStatements (Symbol) returns (IncomeStatementsGrpc);
  rpc GetCashflowStatements (Symbol) returns (CashflowStatementsGrpc);
  rpc GetPeers (Symbol) returns (stream Symbol);
```

48

```
}

message DecimalValue {
  uint32 scale = 1;
  uint32 precision = 2;
  bytes value = 3;
}
message NoParam {
}
message Symbol {
  string name = 1;
}
message BalanceSheetGrpc {
  DecimalValue currentCash = 1;
  DecimalValue shortTermInvestments = 2;
  DecimalValue receivables = 3;
  DecimalValue inventory = 4;
  DecimalValue otherCurrentAssets = 5;
  DecimalValue currentAssets = 6;
  DecimalValue longTermInvestments = 7;
  DecimalValue propertyPlantEquipment = 8;
  DecimalValue goodwill = 9;
  DecimalValue intangibleAssets = 10;
  DecimalValue otherAssets = 11;
  DecimalValue totalAssets = 12;
  DecimalValue accountsPayable = 13;
  DecimalValue currentLongTermDebt = 14;
  DecimalValue otherCurrentLiabilities = 15;
  DecimalValue totalCurrentLiabilities = 16;
  DecimalValue longTermDebt = 17;
  DecimalValue otherLiabilities = 18;
  DecimalValue minorityInterest = 19;
  DecimalValue totalLiabilities = 20;
  DecimalValue commonStock = 21;
  DecimalValue retainedEarnings = 22;
  DecimalValue treasuryStock = 23;
  DecimalValue capitalSurplus = 24;
  DecimalValue shareholderEquity = 25;
  DecimalValue netTangibleAssets = 26;
  string symbol = 27;
}
message CashflowStatementGrpc {
  DecimalValue netIncome = 1;
  DecimalValue depreciation = 2;
  DecimalValue changesInReceivables = 3;
  DecimalValue changesInInventories = 4;
  DecimalValue cashChange = 5;
  DecimalValue cashFlow = 6;
  DecimalValue capitalExpenditures = 7;
  DecimalValue investments = 8;
  DecimalValue investingActivityOther = 9;
  DecimalValue totalInvestingCashFlows = 10;
  DecimalValue dividendsPaid = 11;
  DecimalValue netBorrowings = 12;
  DecimalValue otherFinancingCashFlows = 13;
  DecimalValue cashFlowFinancing = 14;
  DecimalValue exchangeRateEffect = 15;
  string symbol = 16;
}
message IncomeStatementGrpc {
  DecimalValue totalRevenue = 1;
  DecimalValue costOfRevenue = 2;
  DecimalValue ebit = 3;
  DecimalValue grossProfit = 4;
  DecimalValue incomeTax = 5;
  DecimalValue interestIncome = 6;
  DecimalValue minorityInterest = 7;
  DecimalValue netIncome = 8;
  DecimalValue netIncomeBasic = 9;
  DecimalValue operatingExpense = 10;
  DecimalValue operatingIncome = 11;
  DecimalValue otherIncomeExpenseNet = 12;
  DecimalValue pretaxIncome = 13;
  DecimalValue researchAndDevelopment = 14;
  DecimalValue sellingGeneralAndAdmin = 15;
  string symbol = 16;
}
message BalanceSheetsGrpc {
  repeated BalanceSheetGrpc balanceSheetGrpc = 1;
}
message CashflowStatementsGrpc {
```

```
  repeated CashflowStatementGrpc cashflowStatementGrpc = 1;
}
message IncomeStatementsGrpc {
  repeated IncomeStatementGrpc incomeStatementGrpc = 1;
}
```

The available 5 service definitions are listed in the

```
service IexcloudService
```

part, and they related response message types are defined as

```
message <typeName>
```

Within the message type, besides its member data's type and name, the most important information is its position.

Two special types are used, namely the NoParam, and DecimalValue type.

NoParam type is needed for the service getSymbols() as it does not accept any parameter. The gRPC standard google.protobuf.Empty type did not work for me, so this is a workaround.

The DecimalValue type is needed to represent BigDecimal java type in the gRPC ecosystem, as no data type resembling BigDecimal exists within gRPC, yet BigDecimal is the most common datatype in the services response data.

DecimalValue contains all the relevant data for converting it into a BigDecimal value.

The following is an algorithm for the conversation:

BigDecimal to DecimalValue:

```
DecimalValue serializedBd = DecimalValue.newBuilder()
.setScale(bigDecimalValue.scale())
.setPrecision(bigDecimalValue.precision())
.setValue(ByteString.copyFrom(bigDecimalValue.unscaledValue().toByteArray()))
.build();
```

DecimalValue to BigDecimal:

```
MathContext mc = new java.math.MathContext(decimalValue.getPrecision());
BigDecimal deserialized = new java.math.BigDecimal(
new java.math.BigInteger(decimalValue.getValue().toByteArray()),
decimalValue.getScale(),
mc);
```

Nullchecks were omitted for clarity, but exists in the codebase.

gRPC maven plugin's generated classes will appear as the Proto Buffer file is edited. I named the file iexcloud.proto and stored in the *src/main/proto* directory.

As the generated classes will appear in the target folder they aren't usable in the source directory.

In Eclipse, they are needed to be added to the project's sources, as shown in the picture below:
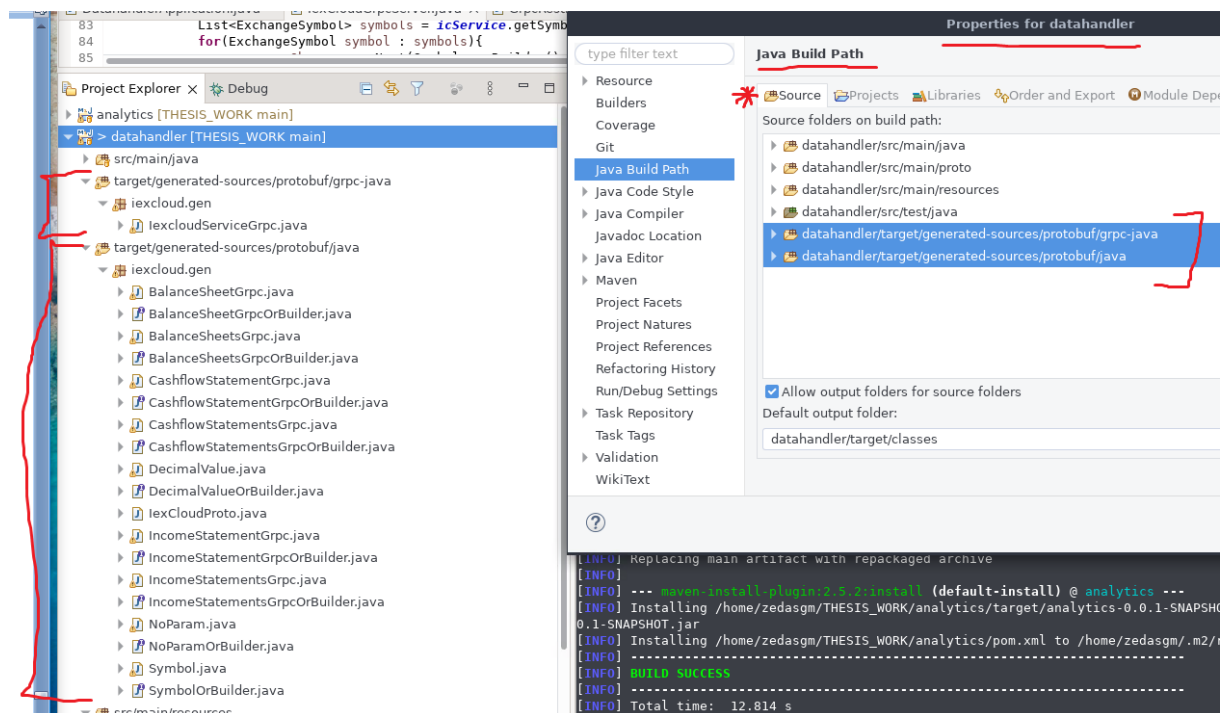
15. ábra. Adding target folders to sources
(source: self-edited illustration)

Datahandler is providing gRPC endpoints inside the cluster. It is done by implementing a running gRPC server implemented in the IexCloudGrpcServer class, running on port 8980. Besides the server setup, it also bootstraps services provided by the server. This IexCloudGrpcServer class instantiated within the DatahandlerApplication class, where the main method resides and the services are contained in the private inner IexCloudGrpcService class, inside IexCloudGrpcServer class.

The IexCloudGrpcService class extends IexcloudServiceGrpc.IexcloudServiceImplBase, and it allows us to override the service methods based on our need, an excerpt can be seen below:

```
private class IexCloudGrpcService extends IexcloudServiceGrpc.IexcloudServiceImplBase {
@Override
public              void              getBalancesheets(iexcloud.gen.Symbol              request,
io.grpc.stub.StreamObserver<iexcloud.gen.BalanceSheetsGrpc> responseObserver) {

List<BalanceSheetGrpc>                              statements                              =
getStatements(icService.getBalanceSheets(request.getName()).getBalanceSheet(),
BalanceSheetGrpc.class, BalanceSheet.class, BalanceSheetGrpc.newBuilder());

BalanceSheetsGrpc                              response                              =
BalanceSheetsGrpc.newBuilder().addAllBalanceSheetGrpc(statements).build();
responseObserver.onNext(response);
responseObserver.onCompleted();
System.out.println("balance sheet response_____" + response.toString());
}
}
```

The above method is providing the requested company's balance sheets back to the caller. The method is supplied with the input parameter and a callback object, responseObserver.

The input parameter is supplied to the getBalanceSheets method of the IexCloudService class, which wraps around the third party IEX Cloud API implementation.

As the REST service returns with the relevant POJO objects, it supplies its content to the getStatements method. As explained earlier, in Datahandler I have defined manually a mapper between the POJOs and the generated gRPC classes.

The method getStatements within IexCloudGrpcService class, and the GrpcRestDtoParser class's (that is defined as a singleton) parseRestToGrpc, and setRestToGrpcField methods are responsible for the conversion.

Converting between a gRPC generated class and a POJO class has its own set of challenges that needed to be solved, however, with the extensive use of generics, I accomplished to parse all 3 different statement types with the same methods.

After the conversion, the gRPC objects are sent to the requesting microservice, via the

```
responseObserver.onNext(response);
```
call.

There were numerous challenges here with the conversation, future improvements might be done to simplify this method, and introduce authentication and tracing for the calls. The dataflow between Microservices is also not encrypted, which poses a security risk if an attacker gets into the cluster. As the Datahandler is the source-of-truth in the MAP system, this is a very important issue to solve. Introducing a service mesh, for example Istio, might solve some of these problems.

Furthermore, standardizing the error handling and log messages would be a further improvement.

Two important files left to mention here, one is the Dockerfile, and the ec33nw-datahandler.yaml.

Dockerfile has been already explained in the Technical glossary, so I rather focus on the exact functionality that Datahandler's dockerfile provides:

```
FROM openjdk:8
ARG JAR_FILE=target/*.jar
COPY ${JAR_FILE} datahandler-0.0.1-SNAPSHOT.jar
ENTRYPOINT ["java","-jar","/datahandler-0.0.1-SNAPSHOT.jar"]
```

I am using an openjdk:8, as the base image. An argument has been set (select a jar within the target folder) and then evaluated on the next line, which copies the jar from the mentioned folder into the container, and name it as

```
datahandler-0.0.1-SNAPSHOT.jar
```

then it is started with the standard jar execution command.

Originally, I tried to use the openjdk alpine image, as it's the smallest size base image that would work for the service. However, during the container run I faced an error that might be related to a bug within that base image. That is why openjdk:8 is used both for Datahandler and Analytics.

After this Datahandler service is completed, a Maven jar build is generated to the target folder via the command:

```
mvn clean install
```

and with the following Docker command and image can be built

```
docker build -t zedas/ec33nw-datahandler .
```

and if authenticated to Dockerhub, with the

```
docker login
```

```
docker push zedas/ec33nw-datahandler
```

commands, the image can be pushed into my own repository.

The ec33nw-datahandler.yaml is a descriptor file for Kubernetes. To deploy an image on Kubernates a Deployment descriptor is needed, to make it available for other services a Service descriptor is needed as well. The .yaml syntax makes it possible to include 2 descriptors within a single file, if 3 dashes (---) separates them, as can be seen below.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: ec33nw-datahandler
  labels:
    app: ec33nw-datahandler
spec:
  replicas: 1
  selector:
    matchLabels:
      app: ec33nw-datahandler
  template:
    metadata:
      labels:
        app: ec33nw-datahandler
    spec:
      containers:
      - name: ec33nw-datahandler
        image: zedas/ec33nw-map-datahandler
        ports:
        - containerPort: 8080
        - containerPort: 8980
        imagePullPolicy: IfNotPresent
---
apiVersion: v1
kind: Service
metadata:
  name: ec33nw-datahandler-service
spec:
  selector:
    app: ec33nw-datahandler
  type: LoadBalancer
  ports:
    - port: 8980
      name: grcpport
      protocol: TCP
      targetPort: 8980
```

The Deployment descriptor here defines the name of the deployment, how many replicas do I need, the image name, the container ports and the image pull policy. As the Datahandler runs a Tomcat server on the 8080, and a gRPC server on the 8980 ports, these are the ones I defined in the descriptor. The image pull policy defines how Kubernates should get the container's image. In case of IfNotPresent, it will only try to pull it if the image is not present locally.

However, I ran into an issue, where I wanted to update the pod with a new image, and it did not fetch again from the repository. Therefore I had to set the imagePullPolicy to Always, and this way, in case if I deleted a pod via:

```
k delete pod ec33nw-datahandler-bc4dd57c5-b8vxc
```

it automatically pulled the fresh image, while restarting the deleted pod.

(It restarted automatically as it must have at least 1 pod since we defined in the .yaml file that: *replicas: 1*)
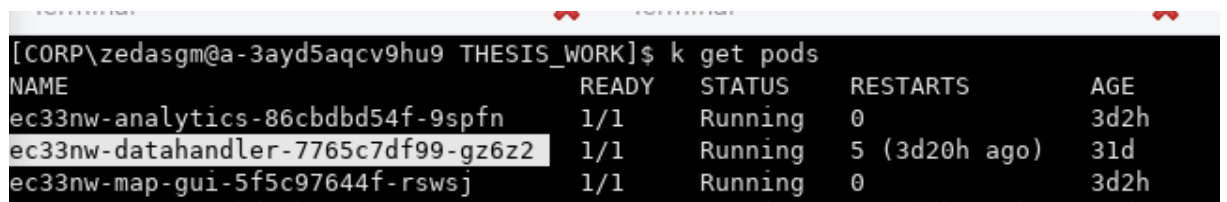
To allow other services to connect to Datahandler via these ports, a Service descriptor defines the port mapping between the cluster and the container inside. Here I mapped them to the same port inside the cluster.

After that if the configuration is done, we can run the following command:

```
k apply -f ec33nw-datahandler.yaml
```

and check if our pod is running correctly:

```
k get pods
```


16. ábra. Running MAP pods
(source: self-edited illustration)

Analytics development, issues and areas to improve

Analytics pom.xml excerpt I./II., only showing the relevant dependencies:

```xml
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-boot-starter</artifactId>
    <version>3.0.0</version>
</dependency>

<dependency>
    <groupId>org.mapstruct</groupId>
    <artifactId>mapstruct</artifactId>
    <version>${org.mapstruct.version}</version>
</dependency>
```

17. ábra. Analytics pom.xml excerpt I./II.
(source: self-edited illustration)

With Springfox, the API descriptions of Analytics's offered services are automatically generated via an annotation. With this description, Frontend will be able to automatically generate the client side of the API. Mapstruct will be used to map gRPC classes to POJOs

55

returned to the Frontend. In Datahandler, I wrote my own mapper, however here I showcase how much easier such mapping can be generated, even for gRPC classes.

Analytics pom.xml excerpt II./II., only showing the relevant plugins:

```xml
<plugin>
<groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-compiler-plugin</artifactId>
<configuration>
    <source>1.8</source>
    <target>1.8</target>
    <annotationProcessorPaths>
        <path>
            <groupId>org.mapstruct</groupId>
            <artifactId>mapstruct-processor</artifactId>
            <version>${org.mapstruct.version}</version>
        </path>
    </annotationProcessorPaths>
</configuration>
</plugin>
```

18. ábra. Analytics pom.xml excerpt II./II.
(source: self-edited illustration)

This plugin helps Mapstruct to generate the mapping.

Dockerfile, .yaml, .proto file that I mentioned also in Datahandler, exists in a similar form at Analytics as well, so instead I will focus on the main parts of Analytics.

In the AnalyticsApplication class, the service has its gRPC client definition, that includes the IP address of the gRPC server inside the cluster, as well as the class has a `@EnableSwagger2` annotation, that allows the Rest API description autogenerated.

The Rest API generated JSON description can be found at: http://ec33nw.map.analytics.api.com/v3/api-docs/ on the host machine.

The Analytics gRPC client stubs are defined in GrpcClientStub class, while the Rest endpoints are defined in the AnalyticsRestController class, annotated with `@RestController`. These Rest endpoints are generated and called by the Frontend. Upon returning data from the Datahandler class, the requested objects will be mapped from gRPC to POJOs via GrpcToJsonMapper, that is a Mapstruct mapper class. This is an abstract class with abstract methods, and during runtime, Mapstruct will generate its own mapping

implementation for each method. In case a custom mapping is needed, for example like the previously mentioned and explained DecimalValue BigDecimal mapping, I defined my own implementation that uses the above mentioned algorithm for mapping.

Due to the special structure of gRPC classes, in case of member collection types, it uses adder methods instead of setters, the mapper class needs to be configured with the following annotation:

```
@Mapper(collectionMappingStrategy         =         CollectionMappingStrategy.ADDER_PREFERRED,
nullValueCheckStrategy = NullValueCheckStrategy.ALWAYS)
```

To resolve the different naming conventions between the gRPC and the POJOs that Mapstruct can't solve its own, these fields were marked with the annotation:

```
@Mapping(source = "incomeStatementGrpcList", target = "income")
```

As each Microservice is a self contained program on its own, I had issues with CORS errors when calling the services from the Frontend side. As the Frontend is hosted on an NGINX server (in a container), calls initiated towards the Analytics service resulted Cross-origin resource sharing error which is a security mechanism that by default, blocks access to resources on a webpage requested from another domain. To resolve this issue the following annotations are needed to be used on the REST methods:

```
@CrossOrigin(origins = {"http://ec33nw.map.analytics.com",
"http://ec33nw.map.analytics.api.com"})
```

This way, every call is allowed from http://ec33nw.map.analytics.com or http://ec33nw.map.analytics.api.com .

Further improvement for the Analytics service would be necessary to implement authentication, encrypting, and query caching.

Frontend development, issues and areas to improve

Frontend, or as its image is called MAP-GUI, serves as an interface between the users and the MAP backend.

It contains automatically generated API methods from Analytics service. The Analytics service methods are generated (after the generator installed, as explained with the command earlier) via the ng build-api script, defined in the package.json file:

```
"build-api":        "openapi-generator-cli        generate        -o        ./api        -i
http://ec33nw.map.analytics.api.com/v3/api-docs/ -g typescript-angular",
```

This script will put the relevant files into a foler called: /api.

Frontend is an Angular application, built with Typescript, the HTML/JS/CSS standard, and used Angular Material for component library. It contains 3 main components. The app component, the main-page component and the value-calc dialog box.

The app component, which is the first component in the Angular ecosystem contains the outer side of the application: the toolbar, the search field, report generation and the value-calc dialog.

The toolbar offers 2 options, the report generation basically utilises html2canvas to take a snapshot of the current state of the GUI and save it as pdf, including the calculation presented there, while the value-calc dialog allows to enter different calculations into the platform, with a draggable dialog box that remains open until it's closed via its close button.

The search field has an autocomplete logic that activates on typing, and only shows the relevant symbols containing the letters typed into the search bar. The search activates when the user selects a ticker and hits enter. Both the toolbar and the autocomplete features are coming from the Angular Material library.

The main component is inside of the app component, and it contains the 3 displayed tables: Balance sheet, Cash flow statement, Income statement, with the data supplied for the Symbol. Upon loading the component, it will fetch the 3 tables's API data, synchronise them with the forkJoin method from RxJS, and upon its completion, plus 1 second, it requests the peers, or competitors of the selected company as well.

As the test API is not allowed to handle more than 3 connections from the same host at the same time, this fix was needed to be used.

The bookkeeping style is kept when designing the tables, since there are so many fields, horizontally it would be impossible to fit the 3 tables nicely. Therefore in MainPageComponent the transposeTables(...) method was used to manipulate data for the tables, as well as the field names are formatted. The tables are also from the Angular Material tables.

Improvement here would be to have translation and prompt for each field, as well as to extend the functionality.

Dockerfile

The main difference in Frontend's Dockerfile is the base image, as its running on an NGINX image, as well as after building the Frontend with the command:

```
ng build --prod
```

the artifacts will be created inside the /dist/map-gui folder, and Docker will copy them to the default NGINX html host folder: /usr/shar/nginx/html.

ec33nw-map-gui.yaml

The main difference between the 2 previous .yaml files is the usage of Ingress. Ingress is an API object that manages the access to the cluster.To use an Ingress, it must be enabled on Minikube with the

```
minikube addons enable ingress
```

command.

The .yaml file ingress part excerpt below contains a few important fields.

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ec33nw-map-gui-ingress
  annotations:
    kubernetes.io/ingress.class: "nginx"
  namespace: default
spec:
  rules:
    - host: ec33nw.map.analytics.com
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: ec33nw-map-gui-service
                port:
                  number: 80
    - host: ec33nw.map.analytics.api.com
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: ec33nw-analytics-service
                port:
                  number: 8081
```

As of the development of this application, only the networking.k8s.io/v1 apiVersion supported Ingress. This Ingress also utilises the default NGINX control plane, which is the default Kubernetes Ingress implementation.

It maps the ec33nw.map.analytics.com to the Frontend's Kubernates service, on the 80 port, while the ec33nw.map.analytics.api.com will be mapped to the Analytics Kubernates service on the 8081 port. These selectors provide the way, just like in the previous .yaml files, to connect a Kubernates Deployment with a Kubernates Service.

This explicit Ingress connection between the Frontend and the Rest API server is especially important, since upon requesting the Frontend on port 80 from ec33nw.map.analytics.com, the whole Frontend will arrive into the end user's client, which is a browser that is of course outside the cluster.

Therefore it won't be able to reach Analytics Rest endpoints, not even with the generated clients. That is the reason why ec33nw.map.analytics.api.com is mapped to the Analytics Kubernates service on the 8081 port via the Ingress, so each API call will be directly channeled into the cluster.

The path type is also very important part of the configuration. I had an issue where I set the path type to Exact, and I only received the base html from the server, but no CSS, or JS files, (404 error) even though they were next to the HTML file. First I had to realize that it's not an NGINX configuration issue by running locally a perfect NGINX container with an Angular build in it. Then I found out that in this case the path type should be set as Prefix, and it solved the issue well.

After applying this file to the Minikube cluster, I was able to query the created Ingress's external IP via

```
minikube  ip
k get ingress
```

commands, then mapped it to the ec33nw.map.analytics.com and ec33nw.map.analytics.api.com addresses to have a domain name locally.

At this point the MAP is ready to use.

Use case

19. ábra. Use case
(source: self-edited illustration)

In this Use case, a user is performing a Return On Asset valuation method on multiple companies, with the "Define valuation" dialog. To calculate ROA, the user must divide the company's Net Income with the Total Assets.

In the dialog's first field the user writes the valuation method's abbreviation, along with the company symbol, whose data are currently loaded into the tables. On the second field, the user provides the numbers to be processed, with the relevant mathematical operators.

The user can search multiple companies and save their calculated values into the Frontend panel.

At the end, the user can generate a pdf report of the calculated valuation results.

Summary

Due to MAP's design and architecture, MAP's strength lies in its expandability, accessibility and scalability. While it is possible to perform financial analysis and reporting on multiple companies to some degree on the platform, it needs to be developed further on many angles.

However, MAP has demonstrated a small step towards a more value focused financial technology concept. A concept that might promote a more sustainable way of investing using lightweight, modern technologies, and as it should be for all SaaS, setting an example by building on Kubernates from day one [41].

Irodalomjegyzék

[1] S. Aramonte, F. Avalos (01 March 2021): The rising influence of retail investors https://www.bis.org/publ/qtrpdf/r_qt2103v.htm (download date: 2021.08.30.)

[2] K. Martin, R. Wigglesworth (09 March 2021): Rise of the retail army: the amateur traders transforming markets https://www.ft.com/content/7a91e3ea-b9ec-4611-9a03-a8dd3b8bddb5 (download date: 2021.08.30.)

[3] Cambridge Dictionary https://dictionary.cambridge.org/dictionary/english/retail-investor (download date: 2021.08.30.)

[4] B. Barber, X. Huang, T. Odean, C. Schwarz (23 Oct 2020): Attention Induced Trading and Returns: Evidence from Robinhood Users https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3715077 page 28-29 (download date: 2021.09.01.)

[5] F. Chague, R. De-Losso, B. Giovannetti (22 Jul 2019): Day Trading for a Living? https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3423101 page 5 (download date: 2021.09.01.)

[6] B. Barber, Y. Lee, Y. Liu, T. Odean (May 2011): The Cross-Section of Speculator Skill Evidence from DayTrading http://faculty.haas.berkeley.edu/odean/papers/Day%20Traders/Day%20Trading%20Skill%20110523.pdf page 23 (download date: 2021.09.01.)

[7]A. Bell (19 March 2020): How Dumb Money Can Become Smart Moneyhttps://www.investopedia.com/investing/surprising-benefits-when-brokers-grade-their-customers/ (download date: 2021.09.01.)

[8] CV Geambaşu, I Jianu, I Jianu, A Gavrilă (2011): Influence factors for the choice of a software development methodologypage 480,482,483 https://core.ac.uk/download/pdf/6261795.pdf (download date: 2021.11.25.)

[9]A Mishra, D Dubey(5 Oct 2013): A Comparative Study of Different Software Development Life Cycle Models in Different Scenarioshttps://www.researchgate.net/publication/289526047_A_Comparative_Study_of_Different_Software_Development_Life_Cycle_Models_in_Different_Scenarios page 64 (download date: 2021.11.25.)

[10]YB Leau, WK Loo, WY Tham, SF Tan (Feb 2012): Software Development Life Cycle: AGILE vs Traditional Approaches https://www.researchgate.net/profile/Leau-Yu-Beng/publication/268334807_Software_Development_Life_Cycle_AGILE_vs_Traditional_Approaches/links/546989b40cf2f5eb1804f3d1/Software-Development-Life-Cycle-AGILE-vs-Traditional-Approaches.pdf page 162 (download date: 2021.11.25.)

[11]V Rastogi(2015): Software Development Life Cycle ModelsComparison, Consequences https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.667.9896&rep=rep1&type=pdfpage 168 (download date: 2021.11.25.)

[12] K. Collier (2012). Agile Analytics: A Value-Driven Approach to Business Intelligence and Data Warehousing

https://accord.edu.so/web/content/69184?download=true&access_token=a6c3a1a0-9efd-408f-a7d7-4a57967a3de0 page 3 (download date: 2021.11.25.)

[13] K Beck, M Beedle, A Van Bennekum, A Cockburn… (2001) Manifesto for Agile Software Development https://moodle2019-20.ua.es/moodle/pluginfile.php/2213/mod_resource/content/2/agile-manifesto.pdf page 1-10 (download date: 2021.11.25.)

[14] Agile Practices https://web.archive.org/web/20140209152034/http://guide.agilealliance.org/(download date: 2021.11.25.)

[15] P Abrahamsson, O Salo, J Ronkainen, J Warsta, (2017) Agile Software Development Methods: Review and Analysis https://arxiv.org/ftp/arxiv/papers/1709/1709.08439.pdfpage 20(download date: 2021.11.25.)

[16] A Sharma, M Kumar, S Agarwal (2015) A Complete Survey on Software Architectural Styles and Patternshttps://www.sciencedirect.com/science/article/pii/S187705091503183Xpage 17, 19, 20, 22, 23, (download date: 2021.11.30.)

[17] N Dragoni, I Lanese, ST Larsen, M Mazzara, R Mustafin, L Safina (2017) Microservices: How To Make Your Application Scale https://arxiv.org/pdf/1702.07149.pdfpage 1, 2 (download date: 2021.11.30.)

[18] D Shadija, M Rezai, R Hill (2017) Towards an understanding of microserviceshttps://arxiv.org/ftp/arxiv/papers/1709/1709.06912.pdfpage 2, 3, 5 (download date: 2021.12.02.)

[19] Y. Niu, F. Liu, Z. Li (2018) Load Balancing across Microserviceshttps://newypei.github.io/files/infocom2018.pdf (download date: 2021.12.07.) page 2.

[20] Agile Alliance: Extreme Programming (XP)https://www.agilealliance.org/glossary/xp/#q=~(infinite~false~filters~(postType~(~'post~'aa_book~'aa_event_session~'aa_experience_report~'aa_glossary~'aa_research_paper~'aa_video)~tags~(~'xp))~searchTerm~'~sort~false~sortDirection~'asc~page~1)(download date: 2022.02.07.)

[21]Agile Alliance: Scrum https://www.agilealliance.org/glossary/scrum/#q=~(infinite~false~filters~(postType~(~'page~'post~'aa_book~'aa_event_session~'aa_experience_report~'aa_glossary~'aa_research_paper~'aa_video)~tags~(~'scrum))~searchTerm~'~sort~false~sortDirection~'asc~page~1)(download date: 2022.02.07.)

[22] Agile Alliance: TDD https://www.agilealliance.org/glossary/tdd/#q=~(infinite~false~filters~(postType~(~'page~'post~'aa_book~'aa_event_session~'aa_experience_report~'aa_glossary~'aa_research_paper~'aa_video)~tags~(~'tdd))~searchTerm~'~sort~false~sortDirection~'asc~page~1) (download date: 2022.02.07.)

[23] M. Fowler (22.04.2020): DomainDrivenDesignhttps://martinfowler.com/bliki/DomainDrivenDesign.html(download date: 2022.02.07.)

[24] S. Newman (2021): Building Microservices Designing Fine-Grained Systems 2nd Edition O'Reilly Media Inc., Sebastopol, Canada, page 89-197 ISBN 978-1-492-03402-5

[25] M. Steen, A. Tanenbaum (2017): Distributed Systems, 3rd edition CreateSpace Independent Publishing Platform, Scotts Valley, CA, USA, ISBN978-90-815406-2

[26] Kubernates documentation

https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/

(download date: 2022.05.24.)

[27] Kubernates documentation

https://kubernetes.io/docs/concepts/overview/kubernetes-api/

(download date: 2022.05.24.)

[28] Kubernates documentation

https://kubernetes.io/docs/concepts/architecture/nodes/

(download date: 2022.05.24.)

[29] Minikube documentation

https://minikube.sigs.k8s.io/docs/start/

(download date: 2022.05.24.)

[30] Docker documentation

https://docs.docker.com/get-started/overview/

(download date: 2022.05.24.)

[31] Nginx documentation

https://www.nginx.com/resources/glossary/nginx/

(download date: 2022.05.24.)

[32] P. Krill (03 March 2022): Java 8 still dominates, but Java 17 wave is coming – survey

https://www.infoworld.com/article/3652408/java-8-still-dominates-but-java-17-wave-is-coming-survey.html

(download date: 2022.05.24.)

[33] gRPC documentation I.

https://grpc.io/about/

(download date: 2022.05.24.)

[34] gRPC documentation II.

https://grpc.io/docs/what-is-grpc/introduction/#overview

(download date: 2022.05.24.)

[35] gRPC documentation III.

https://grpc.io/docs/what-is-grpc/core-concepts/#service-definition

(download date: 2022.05.24.)

[36] Protocol Buffers documentation

https://developers.google.com/protocol-buffers

 (download date: 2022.05.24.)

[37] OpenAPI / Swagger documentation

https://swagger.io/docs/specification/about/

 (download date: 2022.05.24.)

[38] Mapstruct documentation

https://mapstruct.org/

 (download date: 2022.05.24.)

[39] IEX Cloud documentation

https://iexcloud.io/docs/api/#introduction

 (download date: 2022.05.24.)

[40] Angular documentation

https://angular.io/docs

 (download date: 2022.05.24.)

[41] M. Horstmann (21.07.21) Why you should build on Kubernetes from day one

https://stackoverflow.blog/2021/07/21/why-you-should-build-on-kubernetes-from-day-one/

(download date: 2022.05.26.)

Ábrajegyzék

Mellékletek

MAP software, source code and Docker image location

Video

MAP detailed demo video and explanation in Hungarian:

https://youtu.be/_NzbFQ6X1M8

To access the remote PC for the software (details are explained in the video above!):

- Download Amazon Workspace clients: https://clients.amazonworkspaces.com/
- Install and open the Workspace client
- Log in the Workspace:
    - o Username: zedasgm
    - o Password: Ec33nw_053/2022
    - o Registration code: wsfra+QYVPEE

Code and Image access:

- Github code repository: https://github.com/bordaso/ThesisWorks
- Dockerhub image repository: https://hub.docker.com/u/zedas

My contact details:

- Name: Bordás Olivér Odin
- Email: zedasgm@gmail.com
- Tel: 0670/3619916