universität wien

MASTERARBEIT | MASTER'S THESIS

Titel | Title

A Cloud Portfolio Manager- A novel business model

verfasst von | submitted by

Valentin Haag BSc

angestrebter akademischer Grad | in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien | Vienna, 2024

Studienkennzahl lt. Studienblatt | Degree programme code as it appears on the student record sheet:	UA 066 926
Studienrichtung lt. Studienblatt | Degree programme as it appears on the student record sheet:	Masterstudium Wirtschaftsinformatik
Betreut von | Supervisor:	ao. Univ.-Prof. tit. Univ.-Prof. Dipl.-Ing. Dr. Erich Schikuta
Mitbetreut von | Co-Supervisor:	Dipl.-Ing. Dr. Benedikt Pittl BSc

Acknowledgements

I would like to thank my supervisors Univ.-Prof. Dipl.-Ing. Dr. Erich Schikuta and Dr. Benedikt Pittl for the support and patience they have shown me while writing my thesis. Additionally I would also like to thank my parents and friends for their continuous support, and Maximilian Kiessler for the great cooperation and the numerous exchanges of ideas, which contributed to the creation of this thesis.

Abstract

Cloud resources have become a commodity of ever-increasing importance, with many businesses using cloud solutions to either supplement or outright replace their existing IT infrastructure. But, as there is a plethora of providers with varying products, services and markets, it has become increasingly harder to keep track of the best solutions for each application. Cloud service intermediaries aim to alleviate this problem, by offering services that aid users in meeting their requirements. This can include, among other services, finding the right provider and suitable allocations of resources, as well as aiding with deployment and migration of applications.

This paper aims to lay the groundwork towards the development of a cloud portfolio management platform and its business model. As a foundation, it gives a brief introduction into the cloud market, surveys the term business model, cloud specific business models and looks into state-of-the-art of cloud intermediaries. Based on this, this thesis proposes a business model for a cloud portfolio management platform, defined via a business model canvas. Furthermore, a prototype of a platform is developed offering a cloud portfolio optimization service, using two algorithms developed in an accompanying work to create suitable and well utilized allocations for a customer's applications. The final portion of this paper is being dedicated to discussing the steps, which would be necessary to turn the prototype created into a functioning public platform.

Kurzfassung

Cloudressourcen haben in den letzten Jahren enorm an Wichtigkeit gewonnen, was unter anderem, dazu führt, dass immer mehr Firmen ebendiese dazu einsetzen ihre eigene IT- Infrastruktur entweder zu ergänzen oder komplett durch sie zu ersetzen. Aufgrund des weitreichenden Angebotes an Anbietern, mit unterschiedlichen Produkten, Services und Märkten, wird es allerdings zunehmend schwerer den Überblick über den Markt zu behalten, und zu erkennen, welche die besten Lösungen für unterschiedliche Anwendungen darstellen. Cloud-Service Intermediaries stellen einen Versuch dar, dieses Problem zu lösen, indem sie Services anbieten, welche Kunden dabei helfen sollen ihre Bedürfnisse im Cloud Bereich abzudecken. Dabei kann es sich, zum Beispiel, um Hilfe bei der Suche nach dem richtigen Anbieter und der passenden Verteilung und Ressourcen handeln, aber auch um Hilfe beim Aufsetzen und der Migration von Anwendungen.

Ziel diese Arbeit ist es, die Grundlagen für die Entwicklung einer Cloud- Portfoliomanagement Plattform und eines dazu passenden Geschäftsmodelles zu erklären und legen. Zu diesem Zwecke beschäftigt sich die Arbeit anfangs mit einer kurzen Einführung in den Cloud-Markt, gibt einen Überblick über den Begriff des Geschäftsmodelles, sowie cloud-spezifische Geschäftsmodelle, und befasst sich mit dem Stand der Technik von Cloud Intermediaries. Auf Basis dessen wird ein Geschäftsmodell für eine Cloud-Portfoliomanagement Plattform vorgestellt, definiert mit Hilfe des Business Model Canvas. Des Weiteren wird ein Prototyp einer Plattform entwickelt welche ein Cloud-Portfolio-Optimization Service anbietet. Dabei kommen zwei Algorithmen zum Einsatz, welche in einer - in Kooperation durchgeführten -Arbeit entwickelt wurden, und dazu dienen, gut ausgelastete Cloud Kontingente für Applikationen von Kunden zu erstellen. Zu guter Letzt, beschäftigt sich dieses Paper mit einer Abhandlung darüber, welche Schritte notwendig wären, um aus dem Prototypen eine voll funktionstüchtige Plattform zu entwickeln, die öffentlich ihre Dienste anbieten könnte.

Contents

[1] `https://spot.io/`
[2] `https://www.densify.com/`
[3] `https://www.terraform.io/`

1. Introduction

The cloud resource market has been one of the fastest growing IT market segments over the past years. The biggest providers Amazon Web Services (AWS), Microsoft Azure and Google Cloud have seen annual sales increases of over 20% for several years. Just AWS itself reported a revenue of 80 Billion US dollar in the for the year 2022 and the entire cloud market achieving a revenue of 545.8 Billion dollars worldwide with 22% growth compared to 2021 [2, 4]. This large and expanding market has resulted not only in a growing cloud service provider (from here on often referred to as CSPs) market, but also in several methods of delivery of the cloud resources to the customer.

AWS for example, offers three market spaces to customers to purchase virtual machines from. An on-demand market space offers customers the opportunity to purchase virtual machine (VM) in a pay-per-hour model, enabling flexibility without long-term contracts. The reservation market allows consumers to buy VMs for one or three years with a fixed fee, and a discount in comparison with the on-demand market. Finally, the third market, called spot market, offers users the opportunity to buy leftover computing power. The 1-6 hour blocks, that can be purchased there, are significantly cheaper than the on-demand market, but can be interrupted by Amazon at any given time [1, 71]. Many of the other major CSPs like Microsoft Azure offers similar market spaces for purchasing their cloud resources [3]. With such a wide array of possibilities to purchase, it has become a non trivial task for IT management to choose which resources should be bought from which provider, and from which market space.

As mentioned by Pittl et al. (2019), one big challenge facing both industry and academia lies within finding a cost effective solution when buying cloud capacities. Pittls study concludes, that almost all observed resource requests were oversized and offer major cost reduction potential, lying not only in a reduction of the amount of resources bought, but also in their composition. Depending on the planning period of the operations to be performed, a different mix of procurement from different market spaces is optimal. The author suggests to tackle this problem via a cloud resource trading intermediary [71].

Cloud intermediaries are not an entirely new idea, but have been postulated several years ago in papers, for example by Buyya et al. in 2010, advocating a system that offers a scale-able and just-in-time provisioning of computing capabilities to enable fast expansion and reduction of resources bought from cloud providers, to easily adjust to changing demands [40]. Houidi et al. (2011) postulated a cloud broker system, that assists the users in selecting the best cloud platform for their needs, giving an overview of offers and the possibility to split single requests to be served by several cloud providers [52]. Systems like that have been launched in the past years, but have not always been successful. The Deutsche Börse Cloud Exchange (DBCE), for example, was officially launched March 2015, but was only in operation for one year. Examples like the failure of this Broker system by

an established corporation, show that there is still much not entirely understood about the cloud market and the possibilities of intermediaries. This is why the author is of the opinion, that not only researching cloud providers and intermediaries, but also looking into the business models of these companies, is a field of interest.

1.1. Motivation

The rapid adaption of cloud infrastructure across most industries has opened up many new opportunities for businesses, but studies like Pittl et al. (2019) show, that much of the cloud capabilities procured by companies are over-provisioned [71]. While there is a wide array of cloud resources with varying capacities available with different CSPs, each in turn often offering cloud resources in different market spaces, it can be very challenging for a customer to keep an overview.

Therefore it seems, that offering a service that would aid businesses in managing their cloud portfolio and proposing efficient allocations to run their applications, even across various providers, would find great interest in the market. While most larger CSPs offer some functionalities and services that claim to help prevent over-provisioning, like AWS Lambda and Fargate, it is ultimately not in the provider's best interest to reduce the costs for the customer. For the same reason, offering cross platform support should not be expected of them either. While there have been some attempts to create similar services in the past, it is the authors believe that this work offers enough novelty to warrant its creation, with the research goals outlined in the next section 1.2.

1.2. Research goal

The cloud market has seen significant developments and expansion, as mentioned above. One way to better leverage the opportunities this shift offers, while making the market more accessible, is via the use of cloud intermediaries between the CSPs and the customer. In view of that, this master thesis aims to tackle these two main questions:

- **Under what kind of business model could such an intermediary operate?**: To answer this question, the detailed proposal and description of a viable business model for a cloud resource intermediary, will be presented. Intermediaries in a similar fashion have been put forward, and still do exist in similar fashion, but as far as the author is aware, there is no conclusive business model on how these intermediaries could operate.

- **How could a such a platform be implemented?**: The other goal of the thesis, is the design and implementation of a cloud resource intermediary similar to what is proposed by Pittl et al. [71]. This is achieved in cooperation with the work of [59], which focuses on formulating the mathematical problem of cloud portfolio management, and the development of optimization techniques to find suitable solutions. This work is discussed in more detail in chapter 3, while this thesis will

incorporate the developed solutions into a prototype of a Cloud Portfolio Manager platform, which offers customers a service, finding a suitable and cost-effective allocation for their cloud portfolio.

1.3. Structure of the thesis

This thesis deals with a discussion about Cloud Portfolio Management and related business models, as well as the implementation of a prototype for a working platform. It is structured as followed: The current chapter 1 gives a short introduction into the topic of cloud portfolio management, as well as stating the motivation for the thesis creation and discussing its research goals. Next up, chapter 2 gives the reader an overview of the topics which this thesis deals with. First off, in section 2.1, we give an introduction into cloud computing and the cloud market, discussing trends and presenting the most common terminologies. Section 2.2 delves into what a business model is, how it is defined, and presents a widely used framework for the creation of business models. Next, section 2.3 expands upon the previous section by discussing business models tailored to cloud related companies. The final section 2.4 of this chapter discusses existing approaches to offer cloud brokerage between customers and cloud service providers. Chapter 3 discusses the work and findings, which were conducted in tandem with this thesis, focusing on cloud portfolio optimization, and the algorithms that were created. This consists of firstly looking into previous approaches in section 3.1, followed by defining the problem at hand in section 3.2. Afterwards, in section 3.3, we present ERICH and GEORG, the two algorithms developed. Finally, this chapter closes on how the algorithms perform and an evaluation of their results in section 3.4. Outlined in chapter 4 is our proposal of a business model for a cloud portfolio management platform. This starts with discussing various use cases for our business model in section 4.1. In section 4.2 we present our business model consisting of the nine building blocks defined by the business model canvas framework of Osterwalder [70]. Said business model is then classified in section 4.3, before taking a look into the business models of similar platforms in existence in section 4.4. On the basis of the previous sections, we present our prototype of a cloud portfolio management platform in chapter 5. This starts off by defining what requirements the platform should meet in section 5.1. Afterwards the platforms architecture is defined in section 5.2, with the help of a 4+1 view consisting of various UML (Unified Modeling Language) diagrams. After presenting the technology stack in section 5.3, section 5.4 showcases the prototype created. Finally this chapter discusses the user experience (UX) evaluation conducted. The penultimate chapter 6 goes into some detail on how the prototype was programmed, and can be maintained. It closes by discussing which further developments would be necessary and possible, to turn the Cloud Portfolio Manager into a fully functional platform. Finally chapter 7 summarises the findings of the thesis, including what future work could be done to expand upon the topic.

2. State of the art

In this chapter, we will review the literature surrounding the topics of this thesis. This will, initially, entail a general overview of cloud computing, its main characteristics and the cloud market, showcasing its relevance in modern IT. Next, we will discuss the concept of the business model in general, followed by cloud specific business models. Finally, a short overview of cloud intermediaries will be given.

2.1. Cloud computing and the cloud market

When discussing the topic of cloud computing, it is necessary to define the term and its main characteristics. Even though the term is in frequent use today, it is not that easy to define. While the technical aspects and business applications of cloud computing have been in constant development and change over the years, the definition of cloud computing made by the United States National Institute of Standards and Technology (NIST) in 2011 still holds true today:

> *Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.* [66, p.2]

Cloud computing therefore enables companies to perform various IT tasks and operations like computation, storage and access to IT services by procuring them from a service provider that offer these in a scalable, and out-of-the-box, manner.

2.1.1. Cloud characteristics

Besides a definition of cloud computing, the NIST also describes a range of key-characteristics and concepts that can be associated with cloud computing [66]:

- **On-demand self-service**: This refers to the ability of a customer to obtain cloud resources, such as storage or computing power, without direct interaction with human representatives from the service providers side, at their own leisure. This not only increases the flexibility of service provisioning, but also helps to save time and costs for both the providers and the customers.
- **Broad network access**: To enable cloud computing to be a viable alternative to on-premise computing resources, stable, affordable and high-bandwidth network access

must be available to be used via a range of client platforms such as workstations, laptops and other mobile devices.

- **Resource pooling**: By this concept, the NIST refers to cloud providers having their computing resources pooled for many customers, where each customer's needs are dynamically served by assigning the suitable resources to them. This happens without the customer being able to control which exact physical resources at which location are assigned to them, though cloud service providers (CSPs) often allow customers some higher-level control about location, such as choosing the continent or region the physical resources are placed in.
- **Rapid elasticity**: This concept allows customers to elastically obtain and return cloud resources, which enables scaling the capabilities of customers cloud portfolios to meet their current demand. From the customers view, this scalability is unlimited and not time bound.
- **Measured service**: This refers to cloud systems being able to automatically meter the resources, such as storage and processing power, used by the customer, enabling easy monitoring and reporting of usage. This not only provides transparency for both the customer and the CSP, but is also crucial for the most common billing method for cloud resources, which is pay-per-use.

2.1.2. Cloud service models

Besides said characteristics, the NIST and the majority of cloud literature tend to divide cloud services into three distinct models based on what they deliver to the customer. Those are [54, 66, 67]:

- **Software as a Service (SaaS)**: With this, most high-level model, the cloud provider offers the capability to directly deploy customers applications on their cloud infrastructure. Said applications can then be accessed via various client interfaces, such as a web browser or an application interface. With this model, the customer does not have to manage or provide the cloud infrastructure the application is running on, like an operating system or storage. Popular examples using this model would be Gmail and Google Docs [79].
- **Platform as a Service (PaaS)**: This model enables the customer to not only deploy their applications, but are also provides them with the tools and the environment to develop, test, deploy and host applications such as programming languages, libraries and related services, while still not giving the user control over the underlying cloud infrastructure. Examples of this model are AWS Elastic Beanstalk by Amazon and Googles App Engine [79].
- **Infrastructure as a Service (IaaS)**: Finally, IaaS describes a model in which customers are only provided with the basics like processing power, storage and network, leaving all other options to them. The customer is free to deploy and

run any kind of software, including operating systems, storage options and even some network options like firewall configurations. The most used, and well known example for this model is Amazons Elastic Compute Cloud (Amazon EC2).

While the usage of the above described models of cloud computing have proven quite useful, and is still in use today, despite the rapidly developing field that cloud computing presents, it has not gone entirely unchallenged. One example for this development, by Duan et al., speaking directly in reference to the established definitions of the NIST, is a new model they call 'Anything as a Service' (XaaS). It refers to a very broad approach to cloud services, as being any technology delivered via the internet. Due to advances in both server virtualization and internet quality and speed, cloud providers can deliver a wide range of platforms and services to their customers, which enable them to rapidly adapt to ever-changing markets [46, 67].

One example of XaaS would be Business Process as a Service (BPaaS), in which business process outsourcing is delivered via the cloud and done fully automated wherever possible. For those tasks, that require human input, a general pool of labour, not dedicated to just one process or client, is used. Another one, called Database as a Service (DBaaS), entails a service which does not only take care of hardware, software and configuration of the database for a customer, but also any administrative and maintenance tasks that are usually required to operate a database. XaaS can even extend to the black market and illegal practices, with Malware as a Service (MaaS), where customers can purchase services ranging from malware itself to services that help to launder money via scams and ransoms [46, 67].

6	SaaS	Applications			*End User*
5	App Services	Apps in the Cloud	Communications and Social Media Apps	Data as a Service	*Any User*
4	Built-up PaaS	Business as a Process	Social Media PaaS	Data Analytics	*Rapid Developers*
3.5	Serverless Computing				*Speed Developers*
3	PaaS				*Developers*
2	Foundational PaaS	Application Containers	Routing, Messaging, Orchestration	Object Storage	*DevOps*
1	Software Defined	Virtual Machines	Software Defined Networks (SDN)	Software Defined Storage	*Infrastructure Engineers*
0	Hardware	Servicers	Switches, Routers	Storage	
		Compute	***Communicate***	***Store***	

Figure 2.1.: New model for Cloud Computing, from Miyachi, 2018, page 9 [67]

To reflect the developments in cloud technology and the models used to categorise cloud services, Miyachi offers a layered framework that subdivides the existing models in order

to reflect changes to the range of offered solutions, and integrates newer developments like serverless computing. Serverless computing represents solutions with a high level of abstraction, where developers can deploy applications without having to manage the underlying infrastructure. Such services, like AWS Lambda, have the advantage of not over-provisioning for an application's needs [67].

This new model, depicted in Figure 2.1, focuses on changes in PaaS services. Layer 2 in this model is targeted towards DevOps operations and the deployment of binaries, while layer 3 is meant for professional developers deploying code. Finally, layer 4 targets non-developers, like business engineers, that don't write code but set up and connect existing applications or use model-driven development to generate code from models [67].

2.1.3. Cloud deployment models

The final aspect to discuss in regards of a cloud computing solution, is the model of deployment. The NIST defines four types of clouds that can be used, which are the following [45, 54, 66]:

- **Private Cloud**: This kind of cloud infrastructure is meant for use in a single organisation, which entails advantages in terms of security management, maintenance, upgrades and control over deployment, as well as its use and customisation to the user's needs. Unlike its usage, it can be owned, managed and operated not only by the organisation using it, but also by a third party, and does not necessarily have to be on premise of its user.
- **Public Cloud**: Public clouds are available to be used by the general public, most commonly via web browser interfaces. They are usually based on a pay-per-use model, with the customer paying for the used resources on a monthly basis, and the amount of available resources being scalable pretty much without limit. Businesses, academic and governmental institutions can operate this sort of cloud, with the infrastructure being located on premise of the provider.
- **Community Cloud**: Cloud infrastructure adhering to this model is created for the use of a specific community of consumers, which share common concerns like core business or projects. This entails sharing software and hardware to be a good approach in reducing IT costs for all involved community members. A community cloud can be owned and operated by its members or an external cloud service provider, and can exist on or off premise.
- **Hybrid Cloud**: Hybrid clouds are made by combining two or more cloud infrastructures, which while remaining independent entities share common technology to enable shared use of apps and data.

2.1.4. Cloud market

The cloud market currently consists of a wide range of cloud service providers, each of which featuring various ways and markets for a customer to access their products, with no

single space, from which all suppliers can be accessed. The market though, is basically an oligopoly, dominated by the biggest three players. In the past years they have consistently made up around 2/3 of the worldwide cloud market, with the top 10 cloud providers making up almost 80% of the global cloud market according to [32]. The largest cloud service provider is Amazon Cloud Services (AWS) which, in the past years, has made up 32% to 34% of the global market share. Behind that follows Microsoft Azure, which has been catching up with its share growing from 20% to 23% from 2021 to Q1 2023. Finally, Google Cloud with a market share of around 10% makes up the third largest provider [19, 21, 32].

As it is by far the most relevant, this thesis will focus on AWS, while mentioning the equivalent markets for Microsoft Azure and Google Cloud as far as they exist. Amazon Elastic Compute Cloud (Amazon EC2) currently offers three different markets with varying pricing models, which are as followed:

On-demand marketplace

Instances bought on this market are handled in the classic pay-per-use model, where customers can purchase computing power on-demand and by time period, down to 60 seconds increments. This allows for a maximum of flexibility and easy scalability with no advance notice, as there is no long-term commitment needed. The drawback here being that this market has the highest cost per hour for instances [8, 9]. Both Microsoft Azure and Google Cloud offer the same pay-as-you-go pricing model for their respective services coming with the same advantages and disadvantages as with the AWS On-Demand market [14, 26].

Saving plans marketplace

As the name implies, this model offers significant price reductions for the customer in exchange for a long-term commitment to a certain amount of usage. These 1- or 3-year plans offer up to 72% cheaper instances in price-per-hour in comparison to the on-demand market. Therefore they are most suitable for use-cases where steady and predictable usage is expected, like covering the base load of an application running 24/7. Currently, AWS has three different models of saving plans: Compute Saving Plans is the most flexible of them, allowing usage independent of the various instance families offered, size of instances and the region they are located in, but realising slightly lower cost reductions with up to 66% cheaper instances in comparison to on-demand. EC2 Instance Savings Plans go up to 72% savings, but limit the customer to one instance family in one region. Changing between instance types within a family is still possible though. Finally, there is Amazon SageMaker Savings Plans which applies the same principal to AWS machine learning platform SageMaker and offers up to 64% cost reduction [7, 9].

Microsoft Azure has a very similar model in place, which is also called Saving Plans, and offers similar amounts of price reduction with up to 65% cheaper resources in comparison to their on-demand market. Its basic mode of operation is depicted in Figure 2.2. [16]. Google Clouds equivalent to Saving Plans is called Committed Use Discounts, and

unsurprisingly also offers high discounts of up to 80% for 1 or 3 year purchases [25]. Besides Saving Plans that are based upon commitment to a certain amount of $/hour spent, AWS and Azure also offer Reserved Instances. This model involves purchasing the usage of certain instances for 1 or 3 years and also comes with discounts up to 80% comparable to Saving Plans [10, 15].

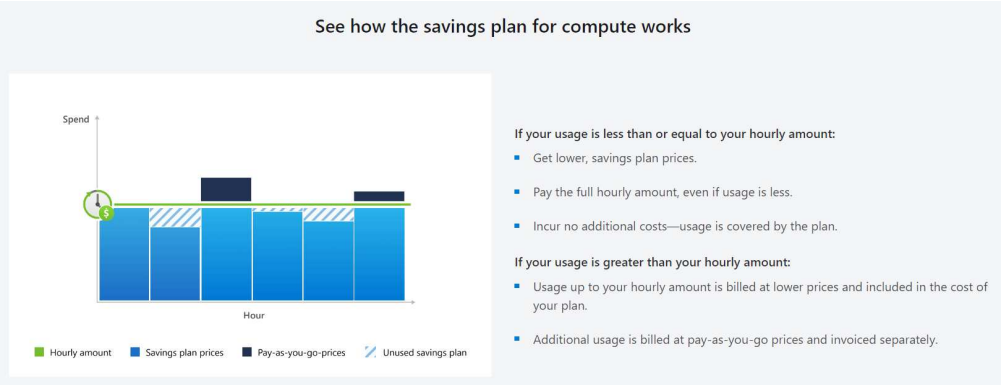


Figure 2.2.: Azure Saving Plans [16]

Spot marketplace

This final market offers customers the opportunity to benefit from the varying loads that AWS and other cloud providers experience on their services. Amazon EC2 offers instances of their currently unused capacity with even greater discounts than the Reserved Market, going up to 90% reduced prices in comparison to the on-demand Market. The prices in this market are in constant flux, and vary by region and instance type while being constantly updated. As seen in Figure 2.3 below, which shows the price development of three example instances over the past months, not all instances will fluctuate at the same rate with one instance type dipping around 25% price around May while another stays at an almost constant prize over the whole observed period [1, 11].

EC2 offers significant advantages in terms of pricing, but there are also drawbacks to procuring resources on this market. Mainly a potential lack of availability, as Amazon reserves the right to stop any instances bought on the Spot Market with only a two minute notice in advance. Amazon keeps track on how likely such interruptions are for certain instances in all regions, with the interruption frequencies ranging from below 5% to over 20%. The company claims, that the average rate over all regions and instances is below 5% [13]. The specific characteristics of this market make it especially suited to applications with save states or stateless workloads, as well as those, that do not depend on specific hardware and have flexible start and end times [1, 11].

As is the case with Saving Plans, Azure and Google Cloud also operate a Spot market very similar to AWS, with very high potential savings up to around 90% reduced costs in comparison to their respective on-demand offers, but at the cost of potential interruptions of instances if their capacity is needed. One notable difference between the big three

Figure 2.3.: AWS EC2 Spot price history over three months [12]

providers in this market is, that Google Cloud, unlike its competitors, guarantees that its spot prices will only change once per month and that the price reduction achieved is at least 60% [17, 24].

2.2. Business model

This subsection consists of a brief overview of definitions and general business model categorisations. It will also include an overview of the Business Model Canvas by Osterwalder and Pigneur [70] which is the basis for our own business model described later in the thesis.

2.2.1. Definition

A business model is something every company posses, regardless whether it is explicitly stated or not. If the same product or technology is introduced to the market with two different business models, the results will also be different, making it a central part of building and managing a business. While the term business model was first used around the 1970s, academic research and widespread usage of the term started towards the end of the 1990s with early papers like Timmmers "Business Models for Electronic Markets" from 1998 [81]. Over the years many researchers have written papers about business models, with many of them focusing on three areas: The definition of business models, business model frameworks and elements, and the classification and archetypes of them [50].

The definition of a business model has not only been one of the earliest focuses of research, but also one of the most hotly debated. Almost every paper concerning this topic has defined its own take on this task, and to this day, there is no general agreement on a universally accepted definition. To showcase this, Figure 2.4 gives an overview of some of the definitions that have been made over the past years [50].

Table 1: A selective overview of business model definitions (ordered by year and author name).

Author(s)	Definition
Chesbrough and Rosenbloom (2002)	The business model provides a coherent framework that takes technological characteristics and potentials as inputs, and converts them through customers and markets into economic inputs. The business model is thus conceived as a focusing device that mediates between technology development and economic value creation. (p. 532) It "spells out how a company makes money by specifying where it is positioned in the value chain" (p. 533)
Morris et al. (2005)	A business model is a concise representation of how an interrelated set of decision variables in the areas of venture strategy, architecture, and economics are addressed to create sustainable competitive advantage in defined markets. (p. 727)
Shafer et al. (2005)	We define a business model as a representation of a firm's underlying core logic and strategic choices for creating and capturing value within a value network. (p. 202)
Chesbrough (2006)	At its heart, a business model performs two important functions: value creation and value capture. First, it defines a series of activities that will yield a new product or service in such a way that there is net value created throughout the various activities. Second, it captures value from a portion of those activities for the firm developing the model. (p. 108)
Johnson, Christensen, and Kagermann (2008)	A business model, from our point of view, consists of four interlocking elements that, taken together, create and deliver value. The most important to get right, by far, is the customer value proposition. The other elements are the profit formula, the key resources and the key processes. (p. 52-53)
Demil and Lecocq (2010)	Generally speaking, the concept refers to the description of the articulation between different BM components or 'building blocks' to produce a proposition that can generate value for consumers and thus for the organization. (p. 227)
Osterwalder and Pigneur (2010)	A business model describes the rationale of how an organization creates, delivers, and captures value. (p. 14)
Teece (2010)	In short, a business model defines how the enterprise creates and delivers value to customers, and then converts payments received to profits. (p. 173)
Zott and Amit (2010)	A business model can be viewed as a template of how a firm conducts business, how it delivers value to stakeholders (e.g., the focal firms, customers, partners, etc.), and how it links factor and product markets. The activity systems perspective addresses all these vital issues [...]. (p. 222)
George and Bock (2011)	[...] a business model is the design of organizational structures to enact a commercial opportunity. (p.99) [...] three dimensions to the organizational structures noted in our definition: resource structure, transactive structure, and value structure. (p.99)

Figure 2.4.: From Fielt, 2013, Journal of Business Models, page 88 [50]

2.2.2. Business model frameworks

Similar to the definition of the term, there is also a wide array of frameworks describing the elements a business model is made up of. Because listing several frameworks without going into any detail would not be very informative, the thesis will focus on the Business Model Canvas from Osterwalder and Pigneur [70], which is one of the most established and widely used framework, and was thus chosen to be basis of the later described developed business model. This framework proposes to describe any business model with nine

building blocks, which can also be grouped into four areas of a business: customers, offer, infrastructure and financial viability. They showcase how a company intends to make money, and are as follows [50, 70]:

Customer segments

This block describes the groups a company tries to offer their products to, and is the key to every business model, as without them, no company is viable. At least one customer segment has to be described, and once chosen, the rest of the model should be constructed with said customer's needs in mind. There are several different types of customer segments with varying focuses. Some examples would be: The mass market, in which there is not a lof of difference between customer segments. Within this segment customers have mostly similar needs and problems, and it is often the focus in business models found in the consumer electronics sector. Another example is the niche market, in which a business has a small, specialised customer segment. This requires tailor-made solutions, such as those offered by businesses producing car parts. On the other hand, companies with a diversified customer base, focus on at least two different groups that have little in common with each other. One case would be Amazon, which added cloud computing services to its retail business, both with separate customer segments.

Value proposition

The value proposition describes the reason of why a customer chooses to work with one business over others and must offer a solution to a customer's problem or satisfy a need. This can be a wide array of things, ranging from innovations to improvements or cheaper offers of an existing product. Some elements that help offer a value creation to the customer can be a newness to a need that has previously not existed, or a product offering an improvement of performance to something already existing, like a PC with better hardware than previous ones. Besides a lower price than similar existing products, there is also the possibility of value creation through reducing costs for customers. Examples would be a customer relationship management application or the topic of this thesis, a cloud resource optimization platform.

Channels

Channels describe the way a company reaches its customer segment in order to make its proposition of value. A business model can include several channels which can be divided into different types, like its own channels versus partner channels and direct versus indirect ones. An example for an own, direct channel would be web sales, whereas a wholesaler would be an indirect and partner channel. Each channel must serve one or more of the five channel phases which are the raising of awareness, helping to evaluate the value proposition, allowing to purchase a product or service, the delivery the value proposition, and finally, post sales support.

Customer relationships

The different kind of relationships that companies can have with their customers are described in this block. They can range from very personal to completely automated. They start with acquisition of customers continuing with their retention and up-selling (getting an existing customer to spend more on your products). Some examples are personal assistance, where human interaction is the key and a special version of this, the dedicated personal assistance, where each customer has a representative dedicated to them. On the other end of the spectrum there are automated services, a kind of self-service relationship, which is paired with an automated process that simulates a personal relationship.

Revenue streams

This block deals with the income a company receives from its customers, and can be seen as its lifeblood. There are several kinds of revenue streams, but they can be separated into two main categories: Transaction revenues for one-time payments and recurring revenues for ongoing payments. Some avenues to create a revenue stream include:

- The classic **asset sale** as a transfer of ownership of physical products that the customers then can use as they please.
- **Usage fees** and **subscriptions** both do not transfer ownership, but allow usage or access to a product or service. Subscriptions work on a timely basis like, gym memberships, where the members can use the facilities as much as they want to. On the other hand, usage fees increase with the amount of usage, like hotels charging for every night spent there.
- Another revenue stream variant are**brokerage fees**, where a business offers mediating services between two or more parties. An example for this would be credit card providers, which facilitate payments and, for that service, get a percentage of each transaction performed.

The pricing mechanisms a business chooses for its services or products can also be broadly divided into two categories: Fixed pricing, like list prices, are dependent on features, volume or customer segment on one hand. On the other hand is the mechanism of dynamic pricing. They encompass pricing, based on negotiations, yield management, where prices depend on inventory and time of purchase, real-time markets using dynamic supply and demand functions and auctions that rely on competitive bidding.

Key resources

The resources described by this term allow the company to operate its business and earn revenues. They depend on the type of business and can be divided into four categories. Physical resources like machines, vehicles and buildings, intellectual resources like patents, knowledge and customer databases, financial resources like cash, stock options and credit

lines, which are contracts to provide funds upon request to predefined conditions, and finally human resources like experienced programmers for a software developer.

Key activities

Described in this block are the most important activities a company must perform to be successful. Like key resources, they are necessary to create value and earn revenue, and can differ widely, depending on the company. The key activities can be separated into three different categories: Production activities focus on the design, creation and delivery of products and are most important in manufacturing. The second category comprises activities related to problem solving, where a company tries to find new solutions to a customers problem. Examples for this are consultancy firms and hospitals. Lastly there are those business models whose key activities are centred around the operation of a platform or a network. One example of such a business model is eBay, which focuses on maintaining and developing its website "eBay.com".

Key partnerships

Key partnerships are the suppliers and partners a business model includes in order to make it work. There are four different kinds of partnerships a company can have: Strategic alliances between non-competing companies, partnerships between competitors, joint ventures to create new businesses and relationships between buyers and suppliers to keep a reliable flow of supplies going. Each partnership can also have one of three main motivational factors for its existence: A very common reason behind partnerships is the optimisation of resources and activities, as it would be unreasonable for one company to do everything by themselves. Outsourcing, or the sharing of infrastructure, therefore exist mainly to reduce costs. Other partnerships focus on the reduction of risks by sharing them, like the joint development of a new technology standard, where each partner of the group can then sell their own products. Finally, there are relationships which have the goal of gaining access to certain resources or activities, like buying a software license instead of developing an in-house solution.

Cost structure

The last block part of the Canvas deals with all operating costs of the business model. There are two different approaches to cost structures. Cost-driven businesses focus on having as little costs as possible, with the help of, for example, automation and outsourcing. Budget airlines are a classic case of this kind of cost structure. The other approach is a value-driven business model, which strives towards value creation instead. This can be exemplified by luxury hotels, which are more about their exclusive offers than price. A cost structure can include several characteristics such as fixed costs, like salaries and rent, that remain unchanged by the volume of goods and services and variable costs, which change depending on the volume of goods or services created. Economics of scale are cost advantages a company gains by producing a larger output, like lower prices for raw

materials, if more of them are acquired. Lastly economics of scope are those advantages gained by operating a larger operation as a whole. For example, a company doing a marketing campaign that can be used for several of its products.

Business model canvas

These nine blocks result in the business model canvas, which is often used to easily and clearly visualise a business model and is depicted below in Figure 2.5.

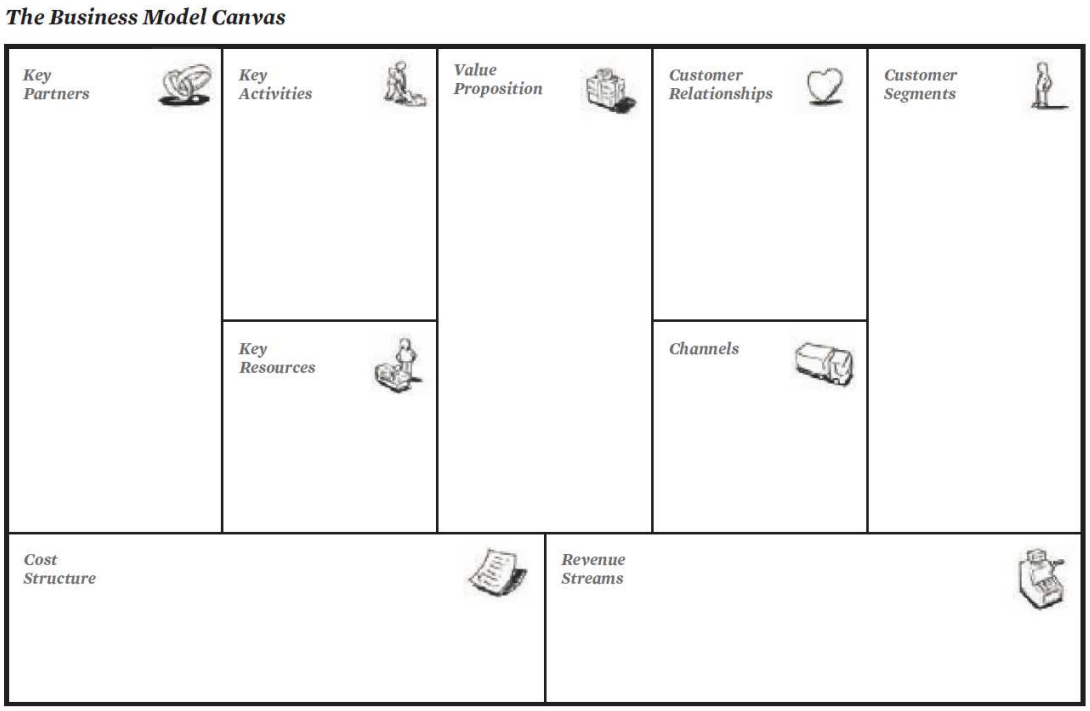


Figure 2.5.: From Osterwalder, 2010, page 44 [70]

2.2.3. Classifications

While there is an array of wildly different business models, researcher have come up with ways to describe and identify different classifications of them. As with the previous two sections, there are a lot of approaches to this, some focusing on creating individual business model archetypes, often based on certain companies like the "low cost carrier model" of SouthWest Airlines. Other research resulted in classifications that encompasses several archetypes and create a typology of business model classes.

Many authors worked on classifications for e-business models like the classification made by Timmers [81], which, in spite of the age of the paper, still applies well to today's e-business models. The classes are the E-shop, E-procurement, E-auction, E-mall, third party marketplace, virtual communities, value-chain service provider, value-chain

integrator, collaboration platform and information broker, trust and other services. These different models are all aligned along two criteria, functional integration, from a single function to multiple, and their degree of innovation from lower to higher. Depicted below in Figure 2.6 are these classes and how they are aligned within Timmers model. [50, 81].

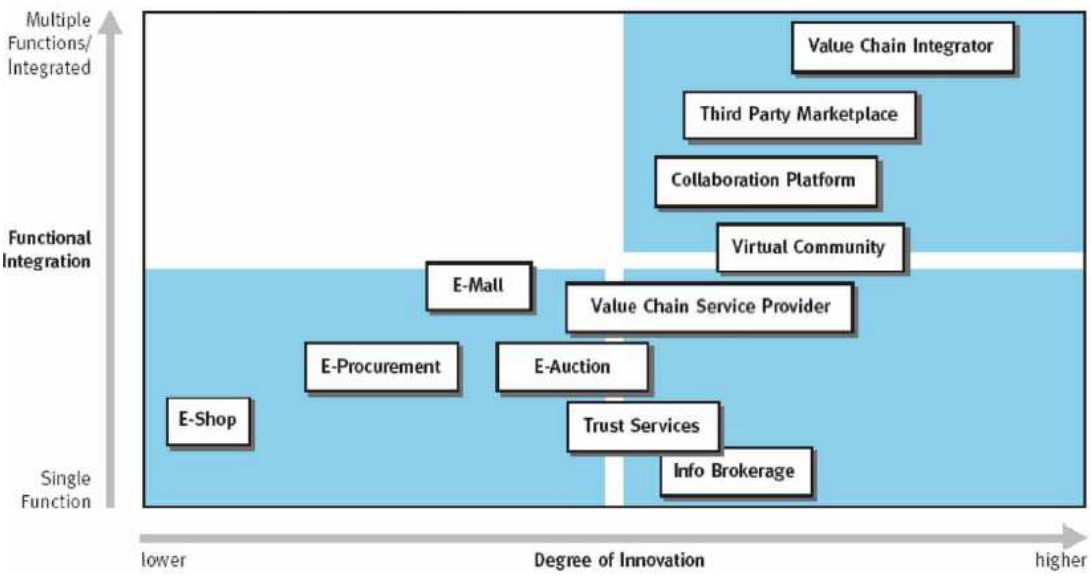


Figure 2.6.: Classification of Internet business models, figure from from B. Wall et al., 2007, Production Planning and Control, page 248 [82]

Another approach for finding similarities within business models are the patterns described in Osterwalder's and Pigneur's Business Model Canvas [70]. Based upon literature research, they come up with five patterns that can be observed in a business model. These are not exclusive, meaning one business can express several of these patterns.

Unbundling

This concept postulates that there are three different kinds of businesses with varying structures and imperatives in the fields of economics, culture and competition. A single company may express more than one of these types, but ideally they are separated from each other, in order to prevent conflicting foci.

- **Customer relationship businesses**, as the name suggests, focus on acquiring customers and developing good relationships with them. They therefore put up with high costs in customer acquisition as it is important to gain a large share of customers, focusing on an economy of scope, in which a few big players dominate the market. In terms of competition, they are highly customer oriented with a customer first approach.

- The second type is the **product innovation business**, which works on creating new and innovative products and services. Their economy has a strong incentive towards speed, as early market entry comes with the ability to charge higher prices and obtain a large market share. This also results in a culture, where many smaller firms can coexist and a more employee-centred approach to competition is observable.
- Finally, with a business model built around platforms to facilitate high volume repetitive tasks, there are the **infrastructure businesses**. In order to deal with their high fixed costs, they need an economy of scale to enable low unit costs. Battling for said scale results, same as with the first type, in a few large players dominating the field. Being very cost focused, they tend to gravitate towards standardization, predictability and efficiency.

The long tail

This pattern describes business models that are built around the idea of offering a large amount of niche products, where each product on its own is only sold somewhat infrequently. While in need of low inventory costs and a competitive platform to make its products easily available, business models following this pattern can be just as profitable as the more traditional model, which sells high quantities of a few products. The term "long tail" in this case stands for the around 80% of products that are not sold in high number. One industry that has seen a shift towards this pattern, is the media business, which used to sell a small amount of "hits" in large volumes. Nowadays, mostly contributed to an easier access to tools of production, global distribution through the internet and lower search costs enabled by the likes of Google, many players of this industry apply the "long tail" concept, selling many different niche products.

Multi-sided platforms

Businesses operating under the multi-sided platform pattern, create value by acting as an intermediary between two or more independent groups of customers and facilitating their interactions. While worthless when only one group of customers is present, they grow in value when more users are attracted. Due to their business model, these companies have to be able to simultaneously gain and hold all customer groups involved. As one side of the platform needs a certain number of users from the other side to see value in the platform, and vice versa. This often results in firms adhering to this pattern to face the "chicken and egg" dilemma. In an attempt to combat this problem, some businesses subsidies one of their customer segments. As this incurs some costs and cannot be done with all segments, the operators of the platforms have to carefully choose which segments to foster and which not.

FREE

The defining feature of working with the FREE business model is that at least one segment of customers can indefinitely access the services or products of the business for

free. There are of course different approaches to implementing this pattern, all having in common that another part of the business or another customer segment has to finance the operation. The underlying economic idea of this pattern is the observation, that a price of zero generates a significantly higher demand, than even the lowest price above zero. In the past years, majorly influenced by the rise of the internet, the number of businesses adhering to this pattern has exploded. To still generate revenue there are three avenues of approach a business operating under this pattern can use:

- First there is the classic model of **advertising**, offering a free service or product and showing the customer ads which the business is paid for displaying.
- The second model has been dubbed **"freemium"**, which refers to a tactic of offering a basic version of the product for free, enabling access to the basic features and is being used by the majority of customers. Besides that, there is one, or more, premium versions which have to be paid for and expose the user to more features. These are usually only used by a small share of customers.
- The last pattern is the so called **"bait & hook"**, in which an attractive, cheap or even free, initial offer "baits" in customers in order to incentivise them to purchase related products. A common example for this tactic is mobile telecommunication companies, offering free phones for signing a contract with monthly subscriptions over a certain amount of time.

Open business model

Finally, the fifth pattern defined is the so called "open" business model, which relates to a business that creates and captures value by working together with partners outside the business. This refers to integrating knowledge, intellectual property, and products from other businesses into their own innovative process. It can be achieved in two ways: "Outside in" refers to innovation by bringing external resources and ideas into a company, while "Inside out" innovation occurs when firms license or sell their own technologies and/or assets to outside parties.

2.3. Cloud specific business models

After having described business models in general, this section will look into cloud specific business models and important characteristics of successful examples.

As there are quite the number of cloud providers on the market nowadays, the question of what makes such a company and its business model successful, may come up. A study done by Labes et al. in 2017 looks into this topic, by analysing 45 providers and coming up with a list of indicators for their success and varying them by conducting a number of expert interviews [61]. Based on the EBIT (Earnings before interest and taxes) margin and the firm's visibility in the web, which has been found to positively correlate with a company's business performance by Wang et al. [83], eight critical success-related business model components and their characteristics were analysed. The results can be seen in

Figure 2.7 below. The importance of the components were then ranked, coming to the following results: Resources and Activities, Value Proposition and Business Strategy were the most important, followed by Costs, Revenue, Distribution and Customer Relationship, Target Market and Partner Network [61].

No.	Critical success-related BMC of the analysis results in the components of a business model
1	Business strategy: know-how transfer (***/*), vertical diversification (**/**), market expansion (**/**)
2	Partner network: partners in similar field (***/*)
3	Resources and activities: know-how – (***/*), human – (**/**), hardware – (**/), network resource (**/), data/content (/*), production activities (*/), consulting activities (**/), integration activities (**/), comparison and categorization (/**)
4	Costs: fixed operational costs (**/)
5	Value Proposition: manifold width (***/***), -depth (**/), computing service (*/), development environment (**/*), -tool (/*), consolidation (***/), cost savings (/**), administration (***/), private – (**/), hybrid cloud (**/), database – (***/*), consulting – (**/), integration – (*/), billing – (/**), search – (/**), messaging service (/***), individual support (/***)
6	Distribution and customer relationship: print media communication (***/), monitoring (***/), customer community (/***), support (/**), on-site interaction (**/)
7	Revenue: one-time-charge (***/), pay-per-use revenue (**/), revenue with supplementary service (**/**), membership fees for partners (/**)
8	Target market: SME customers (/**), branch market (*/)

Spearman's rank correlation (EBIT margin/inlink count)

* $p < 0.05$, ** $p < 0.025$, *** $p < 0.01$, two-tailed test

Figure 2.7.: Critical success-related business model characteristics, figure from S Labes et al., 2017, Business & Information Systems Engineering, page 5 [61]

A further analysis of the business model characteristics also revealed three different types of business models, with differing value creation, proposition and delivery that all providers could be divided into [61]:

- **Newcomer:** This type describes newcomers to the market that adapt already existing cloud strategies or form co-operations with already established cloud companies. Their value proposition focuses much on individual customisation and often result in larger initial costs. The newcomer's main customer segment are niche markets, which they reach by traditional channels, such as print media and personal contact. Revenue is generated by a one-time charge and later by supplementary services or partner revenue models. This type of cloud provider has to deal with a crowded market in which it has to stand out by having a well-developed market entry strategy, and a defined target market. On the other hand, they have the advantage of being able to work with flexibility and develop a specialised cloud service, instead of focusing of commodity services.

- **Experienced player:** Companies of this type mainly work with their existing know-how, offering a plethora of software and consultancy services. The services offered are standardised and provided via a public cloud with high security standards, resulting in excellent scalability and possibilities for time and cost reductions. The experienced players target both the mass and individual market and use support systems and online communities to reach their customer base. Subscription based services usually account for the biggest source of revenue. While profiting from an

economy of scale and a good understanding of their technology, they have to be careful that less direct customer contact and lower trust levels are compensated by good CRM, branding and marketing strategies.

- **Specialised provider:** Providers belonging to this type stand out by expanding the usual cloud services on a vertical level, by adding services such as data processing, administration, marketplace and migration services. Their target customer segment is branch-specific and may include the public sector. Firms following this business model often implement a usage-based revenue model. The specialised providers have the advantage of a high quality and innovative model, that, in combination with good security and customer orientation, results in high levels of trust and loyalty by their customers. While not easily imitated, the model does not support high scalability and needs constant innovation to satisfy the customers ever developing needs.

2.4. Cloud intermediaries

Now, several years later, while quite a few papers having been written and models have been proposed, actual cloud broker implementations on the market are still sparse, while the high number of cloud service offerings make it a hard task, to find the right service as a customer. Brokerage can be several services the intermediary offers the cloud customer, such as decision support or enhancing the delivery of services. The benefits for the customer are lower costs and the ability to seamlessly switch between different cloud providers [47].

A new field that has risen within the cloud brokerage market, is cross-cloud computing, which aims to support interoperability of applications for different clouds, and has brought to light different models on how interoperability is to be achieved. The first of which is the so called "cloud federation", which urges cloud providers to implement common technologies, allowing for easier deployment of a customer's applications on different clouds. The multi-cloud model, on the other hand, sees the broker in the role of facilitating the switch between different cloud providers without common technologies. Another possible approach is the broker offering a decision support system, which does not aid in the practical part of the deployment, but offers recommendations on the best provider for the customer's needs [47].

2.4.1. Why cloud intermediaries?

To gain a deeper understanding of the cloud intermediary market, Elhabbash et al. posed research questions in 2019 as part of a systematic survey of cloud brokerage literature, looking into the motivation for designing a cloud broker and their functionality [47]:

- **Motivation:** Regarding motivation for a cloud broker, four key motivating factors could be identified: **Dimensionality**, meaning the overwhelming number of cloud providers making the task of selecting a provider a substantial challenge, was the

first factor mentioned. It leads to many broker systems being developed as a decision support to find the best cloud service. **Vendor lock-in** refers to the challenges and costs for customers related to the switching of cloud provider. Furthermore, there are several references in the literature talking about the need for brokerage, as one cloud provider cannot always meet the requirements of customers. For example, the need to scale over multiple providers, when the resources of one provider are not sufficient at a certain time. Another example for this motivating factor of meeting requirements, is **cloud bursting**, which refers to the practice of only using public cloud resources at times when the own private cloud is not sufficient. For this task, an intermediary needs to enable a seamless addition and removal of public cloud resources, when their own infrastructure is in need of it. Lastly any other motivation stated was summarised under the term **pathological factors**, like the addressing of specific interoperability guarantees [47].

- **Functionalities:** Over the sum of the articles surveyed by [47], eight different functionalities of cloud broker could be identified. **Decision Support** focuses on aiding the customer with finding the best selection of resources for their needs. The functionality of **Resource Monitoring** is the collection of data from cloud services, often concerning performance and availability, in order to help make decisions in the future. **Policy Enforcement** has the broker implement certain constraints set by the customer, like costs and location of deployment. This can then be used to form decisions about for example VM Migration. A similar approach is that of **Service Level Agreement (SLA) negotiations**, where the broker takes the user's requirements to clouds providers, with the intention of creating services fitted to said requirements. This functionality requires a cooperation between the broker and the cloud provider. This form of functionality has also been expanded upon in research like that of Mach et al. in 2012, where they presented a framework that allows for automatic and dynamic negotiation and re-negotiation of SLAs between CSPs and customers based on the consumers needs and the CSPs offers [62]. Further services provided by brokers can be the **aiding with deployment** of applications to a cloud and the migration of already existing deployments. The scope of these functionalities vary, ranging from recommendations of where to deploy to offering the whole deployment or migration process as a service. **API Abstraction** is an important feature of many cloud intermediaries, which is either achieved by finding overlapping APIs offered by providers to find a least common denominator offer-able to customers, or by the broker creating its own API, which translates calls to the providers APIs. The last functionality found to be offered by brokers, is that of **VM Interoperability**, meaning the conversion of VMs from one provider to another [47].

Further findings of the survey concern both general observations and a critical look into the state-of-the-art. It was found that most intermediaries focus on the IaaS provisioning level, with most brokers not including PaaS and SaaS solutions at all. This is mostly due to IaaS being more flexible, easier to migrate and offering the most potential for

cost reductions. Another critical observation was the lack of real-world testing of the proposed brokerage systems. While many works used simulations and prototypes, only few researchers used both and less than 20% evaluated their research through a real cloud. Some of the further limitations identified were a lack of work towards solving interoperability challenges, not many deployment tools tackling migration of VMs and the general wrongful assumption that cloud customers are completely aware of the technical requirements of their applications before deployment [47].

2.4.2. Cloud intermediary business models

As previously mentioned, while there has been a plethora of research conducted in the field of cloud intermediaries, the amount of implementations on the market is very limited. This, in conjunction with the fact that the services provided can very wildly, has resulted in a lack of research concerning business models of cloud brokers so far. Especially in comparison to cloud provider business models, which have been discussed in previous sections. Nevertheless, the work of Filiopoulou at al. gives an overview of benefits, common pricing models and an evaluation of cloud brokers [51].

The main benefit cloud brokers offer to their customers, is the opportunity to focus on their core business, while the broker deals with issues related to the cloud, like deployment strategies and pricing. Businesses often lack the knowledge and time to find the best provider for their needs. The intermediary represents a trusted and reliable partner, offering guaranteed resources and uniform Service Level Agreements (SLA), which otherwise vary between different providers and possibly confuse customers. Furthermore, brokers can regularly achieve better discounts and access more information, as large providers of cloud resources such as Google and Amazon form cooperations with cloud providers to reach enterprise customers more easily. Overall, brokers assist companies in developing themselves and in creating a more competitive environment for providers, while earning revenues themselves [51].

According to Filiopoulou at al., there are four common pricing and acquisition models used by cloud broker [51]:

- **Financial brokering based on derivative contracts:** This kind of contract derives its value from underlying assets. A common version of these contracts are options, which give the buyer a right to buy a resource on a later date for a, beforehand agreed, price. They can be used by brokers to avoid the risk of not knowing future supply and demand of cloud resources. In this model, clients inform the broker about probabilities of needing an instance in the following month. Based on this data, the broker buys options from cloud providers. This kind of strategy is more profitable for long term options, but depends heavily on the client's ability to forecast his requirements correctly.

- **Broker model for IaaS resources:** Based upon provider tariffs instead of providers, this model is based upon the customer's specific needs for CPU, RAM and storage capacity. After receiving this information, the broker collects the tariffs

from the cloud providers and calculates the cost-performance of each tariff. The client is then presented with the solution offering the best price per performance. Based upon the data from [51, 71], larger instances often have a worse price per performance.

- **Dynamic cloud resource reservation:** A broker operating under this model reserves a large amount of instances from cloud providers, which are then in turn offered to users at a discount. This has the potential to exploit the benefits of long-term reservation and can be optimised by dynamic predictions of the customer's needs to reduce service costs incurred by unused reservations. Users can purchase instances from the broker on-demand, with neither the need for upfront payment for reservations, nor the costs for idle reserved instances. The broker, on the other hand, makes profit by leveraging his reservation model.

- **Quantized billing cycles (QBC):** The term QBC stands for a model, in which the customer pays the same prize for an on-demand instance, no matter if the instance is used for the whole duration of the billing cycle. This problem is often faced by users with sporadic demands, while a broker trying to serve this demand risk severe underutilisation of bought instances. It can be solved by dynamic pricing, where sudden, unexpected demand will result in higher prices in order to reduce demand. This will lead to a small loss, but prevent a bigger loss that would occur from buying an additional VM that ends up underutilised later. This model profits greatly from demand predictions and while static pricing is a possibility too, dynamic pricing increases its profitability.

3. Cloud portfolio management

This thesis was created in cooperation with my colleague, whose focus on developing optimization approaches for cloud portfolios. In this chapter, there will be presented a synopsis of the findings of the work of cooperative work of Kiessler and me, which encompasses related approaches, a formulation of the problem at hand, a description of the two algorithms developed, and finally an evaluation of their performance [59].

3.1. Related approaches

The main goal of cloud portfolio management is usually achieving the lowest costs for running a specific set of applications over a certain amount of time. When trying to optimize a portfolio, one of the biggest challenges faced is that the relevant marketspaces are very heterogeneous in nature. Related research into this field has often preferred looking into the spot market, while disregarding the on-demand and reserved markets. The main reason for this being that the spot markets unique attributes lend itself best for optimization approaches. One major downside to focusing on the spot market, is the necessity for preemptibility of any application or service run on spot instances to prevent loss of data [59].

Some research like Jangjaimon and Tzeng [55] and Sharma et al. [78] tried to deal with this problem by creating a checkpointing mechanism, or by focusing on preemptible servers in combination with concepts taken from financial modelling, in order to meet the requirements of applications when using spot instances. Meanwhile Pittl et al. [71] took a more comprehensive approach to cloud portfolio management, which resulted in the findings that a more heterogeneous portfolio tends to be more cost efficient. Another finding of that paper was to highlight the significance of right-sizing, where server instances should be chosen to most closely fit the capacity requirements of the applications the workload consists of. Otherwise an expensive over-provisioning of resources could be the result [59].

On the topic of right-sizing, Hwang and Pedram [53] developed a portfolio-based optimization approach with a probabilistic model, which, while created for data centre operations, is also applicable to cloud portfolio management. Each assigned workload in this model has a resource demand, not defined by a fixed value, but via probabilistic model using a normal distribution. This idea of using probability based problem formulation was also used in further research, for example by Martinovic et al., who combined it with a stochastic bin packing approach to find efficient server allocations [65]. One further approach proposed for optimization of virtual machine placement in order to reduce energy consumption in data centres, is using a genetic algorithm, as was done by Wu

Table 3.1.: Notation for problem formulation

Parameter	Description
I	A set of selected cloud instances (hosts)
A	A set of applications
T	A set of time slots for which to optimize
x_{ait}	Variable denoting assignments of applications to instances
S_a	The starting time of application a
F_a	The finishing time of application a
U_a	Indicates if application a is preemptible
μ_a	The expected resource demand of application a
σ_a	The std. deviation of the resource demand of application a
R_i	The resource capacity of instance i
C_i	The cost of instance i for one time slot
B_i	The first available time slot of an instance
E_i	The last available time slot of an instance
O_i	Indicates if instance i is suitable for non-preemptible applications
D_{it}	Aggregated resource demand of instance i at time t
Q_{min}	The desired quality of service

et al. [84]. Another interesting concept to be regarded is the one by De Cauwer et al., which, while using deterministic resource demands, introduced the idea of a temporal component to server allocation schemes. This additional dimension better reflects real computational needs, where applications often do not run forever, but with a certain start and end time [42, 59].

3.2. Problem formulation

This section will deal with formulating the problem, which has to be tackled by the cloud portfolio optimization approaches. The notations used for this task are defined in 3.1 [59].

First of we define a cloud portfolio as a set of *cloud instances* I, which are used to run a set of *applications* A on them. The main goal of our optimization problem hereby,is being able to find a cost-efficient allocation of these application and instances [59].

All applications A have a certain resource demand, which, as proposed in literature by Hwang and Pedram, have their resource needs not defined as a fixed value, but with fluctuation taken into account [53]. Therefore we denote the capacity requirements of our applications as an expected demand mean μ_a and with a corresponding standard deviation σ_a. As our workloads also have varying run times, each application sports a starting time S_a and finishing time F_a, for which the statement $S_a < F_a$ must always be true. To model which applications are suitable for spot instances, meaning they can be interrupted at any time, we use the variable U_a which can either be 0 or 1 [59].

$$U_a = \begin{cases} 1 & \text{if } a \text{ is preemptible} \\ 0 & \text{else} \end{cases} \tag{3.1}$$

For modelling instances, each instance $i \in I$ has a predetermined resource capacity R_i, which represents, for example, the number of CPUs and RAM of an instance. Furthermore, each kind of instance has a price per time unit C_i, where the total cost of any instance is calculated by the price per time slot and the overall up-time. As with applications, each instance has a given starting time B_i and ending time E_i, with the inequation $B_i < E_i$ once again having to be fulfilled at any time. Preemptible spot instances are denoted by the binary parameter O_i [59].

$$O_i = \begin{cases} 1 & \text{if } i \text{ is only suitable for preemptible applications} \\ 0 & \text{else} \end{cases} \tag{3.2}$$

To model the summed up demand of all applications assigned to an instance $i \in I$, for a certain time slot $t \in T$, we use the parameter D_{it}. As with the demand of a single application, the aggregated demand is also not a fixed variable, but a random variable, which means it can only evaluate the probability with which an instance stays within the designated capacity limits. Our proposed model also defines a desired minimum quality of service Q_{min}. An allocation is invalid, if the probability of the aggregated demand D_{it} staying below the provided capacity of the instance R_i does not satisfy the minimum quality of service Q_{min} for time slot t. This approach also builds upon the model proposed by Hwang and Pedram (2012), but has to take into account the temporal component of our model, meaning that the capacity restrictions have to be fulfilled for every time slot an instance is used [59].

For modelling the formal assignment of applications to instances, while taking into account the temporal restrictions of the problem, we used the approach postulated by Dell'Amico et al [43]. The variable x denotes which hosting instance an application has been assigned to, at a specific point in time. The resource demands for an application $a \in A$ are considered to be 0 for any time slot $t \in T$ if $t < S_a$ or $t > F_a$ [59].

$$x_{ait} = \begin{cases} 1 & \text{if application } a \text{ is assigned to instance } i \text{ at time slot } t \\ 0 & \text{else} \end{cases} \tag{3.3}$$

Having outlined the requirements and assumptions of the cloud portfolio optimization, we can now present our exact problem statement. The main optimization goal is to find the minimum of the following cost function [59]:

$$min \sum_{i \in I} C_i * (E_i - B_i) \tag{3.4}$$

3.4 is the main function to optimize. It minimizes the costs of the entire portfolio, which consists of the sum of all prizes incurred by each assigned instance. Hereby $(E_i - B_i)$ is the time an instance is running, which is multiplied by its costs per time slot C_i to arrive at the price the instance will account for. While trying to minimize 3.4, the following statements have to be fulfilled [59]:

$$s.t. \sum_{i \in I} x_{ait} = 1 \qquad \forall a \in A, \forall t \in [S_a, F_a] \tag{3.5}$$

$$\sum_{i \in I} x_{ait} * U_a \geq O_i \qquad \forall a \in A, \forall t \in [S_a, F_a] \tag{3.6}$$

$$P(D_{it} < R_i) \geq Q_{min} \qquad \forall i \in I, \forall t \in [B_i, E_i] \tag{3.7}$$

$$x_{ait} \in \{0, 1\} \tag{3.8}$$

$$U_a \in \{0, 1\} \tag{3.9}$$

$$O_i \in \{0, 1\} \tag{3.10}$$

$$Q_{min} \in [0, 1] \tag{3.11}$$

The first constraint 3.5 asserts that each application has to be assigned to an instance, while at the same time, each at any point of the run time of an application, it can only be hosted by one instance. The next constraint 3.6 states that each host has to come from a suitable market space, which is necessary to guarantee that non-preemptible application are not assigned to spot instances, which could be interrupted at any time. The equation formulated in 3.7 ensures that for every time slot an instance is running, the probability of the resource demand of the applications assigned being within the capacity of said instance, is at least the quality of service Q_{min}. The final four constraints 3.8 to 3.11 state that the auxiliary variables have to be within a valid range from 0 to 1 [59].

3.3. Optimization approaches

Having modelled our cloud portfolio optimization problem in section 3.2, this section will now present our approaches to solving it. As our problem is essentially a multi-dimensional packing problem, it is NP-hard, very complex to solve and finding an optimal solution is most of the time not computationally feasible [41]. We therefore developed two optimization heuristics, which aim to find good approximations of an optimal solution [59].

3.3.1. Greedy algorithm

Our first algorithm is called *Efficient Resource Inference for Cloud Hosting* (ERICH). It integrates the approach of the widely known bin packing algorithm first fit decreasing (FFD) [63], combing it with the proposed portfolio management strategy by Hwang and Pedram [53]. It is executed in the following four stages [59]:

Stage 1: As the first step, the algorithm sorts the preemptible and non-preemptible applications it receives as input by increasing starting dates, with applications that start

earlier being allocated first. Applications with the same starting date are then sorted by non-increasing standard deviation of their resource demand, based on a proposal by Hwang and Pedram, suggesting that reduced capacity needs can be achieved by grouping together workloads with similar resource demand deviation [53]. To ensure the algorithm preferring cost-efficient hosts, this step also includes sorting all received instance types by cost per time slot for the provided capacity in an increasing order [59].

Stage 2: Next, the algorithm tries to allocate all non-preemptible applications to reserved instances by using the first fit decreasing approach. Iterating through all such applications, the algorithm first tries to find a suitable host, which provides the needed capacity over the entire run-time, in the already existing portfolio. If that is the case, the application is assigned to said instance, otherwise a new instance covering the applications needs is added to the portfolio [59].

Stage 3: While reserved instances offer significant discounts in comparison to those procured on the on-demand market, the portfolio created in the previous step, may be inefficient, due to the requirements for minimum run-time in reserved instances. Therefore, step 3 tries to condense the portfolio by removing instances chosen in step 2 and trying to replace them with on-demand instances. To achieve this, the algorithm iterates through all instances, at each step creating a new temporary portfolio with one reserved instance removed. Next, applications from this instance are assigned to on-demand instances with the same first fit decreasing approach used in the previous step. Should this new allocation allow for a cheaper portfolio, it replaces the old one in the next iteration [59].

Stage 4: The final step of the algorithm deals with finding fitting instances for all preemptible applications, which can be assigned to multiple hosts during their lifecycle and can therefore be allocated on an individual timeslot basis. To leverage this, the algorithm will first find any time steps for which a suitable candidate hosts exists and assigns preemptible apps to these instances for the respective periods. Should there be any more need for workload to run the apps, the difference is made up by adding new spot instances [59].

The algorithm 1 below describes the steps just discussed in form of a pseudo code. To check if an application fits into any given instance, the equation 3.7 is used.

Algorithm 1: Efficient Resource Inference for Cloud Hosting

Input : A set of non-preemptible apps $A1$; A set of preemptible apps $A2$; A set of reserved instance types RES; A set of on-demand instance types ON; A set of spot instance types $SPOT$

Result: Packing pattern $portfolio$

sort applications $A1$ and $A2$ by increasing start time and non-increasing σ_a
sort RES, ON and $SPOT$ by non-increasing C_i per R_i and time slot
$portfolio \leftarrow$ empty allocation variable
forall $a \in A1$ **do**
  assign a to $portfolio$ (FFD) while only considering RES instances
forall $i \in$ *reserved instances from* $portfolio$ **do**
  $tmp\_portfolio \leftarrow$ copy of $portfolio$ without instance i
  forall $a \in i$ **do**
    reinsert a into $tmp\_portfolio$ (FFD) including ON instances
  if *total cost of* $tmp\_portfolio <$ *total cost of* $portfolio$ **then**
    $portfolio \leftarrow tmp\_portfolio$
forall $a \in A2$ **do**
  assign a to $portfolio$ without allocating new instances
  $gaps \leftarrow$ consecutive time slots where a is not yet assigned to $portfolio$
  forall $gap \in gaps$ **do**
    assign a to $portfolio$ for time slots in gap by allocating $SPOT$ hosts

3.3.2. Genetic algorithm

Using a genetic algorithm (GA) for solving a bin packing problem is not an entirely new idea, but has been proven to work well in dealing with combinatorial optimization problems [49, 58, 72, 73]. GAs are based on genetic operators that can be adapted to fit a certain problem enabling them to perform an efficient and targeted search in the problem space. Our genetic algorithm has been named *Genetic Optimization of Resource Groupings* (GEORG) and will be described in this subsection. The algorithm is described in pseudocode in algorithm 2, which is followed by a description of the algorithms building blocks [59].

Algorithm 2: GEnetic Optimization of Resource Groupings

Input : A set of non-preemptible apps $A1$; A set of preemptible apps $A2$; A set of reserved instance types RES; A set of on-demand instance types ON; A set of spot instance types $SPOT$

Result: List of packing patterns (portfolios) *population*

population ← use semi-random heuristic to create initial population

while *termination criteria are not met* **do**

- $parents$ ← fitness-based selection of individuals from *population*
- $offspring$ ← apply temporal biased crossover for each tuple in $parents$
- $offspring$ ← repair broken chromosomes in $offspring$ after crossover
- $offspring$ ← apply domination mutation operator to random $offspring$
- $offspring$ ← repair broken chromosomes in $offspring$ after mutation
- *population* ← fitness-based merge of $offspring$ and current *population*

Encoding scheme

Grouping of items and usage of bins are essential for building a GA according to Falkenauer [49]. Their approach of encoding, where each chromosome consists of an array of bins holding a set of items, is not suitable to our problem with its temporal component. That is why we have chosen a temporal group encoding, where every locus of a chromosome represents a certain time step, while the allele is a set of instances running at the certain point in time. Finally, every host is assigned a set of applications, which are then allocated to the corresponding instance during this time slot. [59].

Population initialization

To enable the operations of a GA like crossover, mutation and survivor selection to occur, an initial set of individuals (a population) is needed. For our initialisation process a hybrid approach was chosen to both achieve comparatively high fitness from the start, while also offering good genetic diversity. Therefore, half of the assignments are done at random, with the other half having applications that have been assigned with the aim to reduce the number of allocated instances and the overall costs [59].

Fitness evaluation

This building block of our GA uses the equation 3.4, the main function to optimize, in order to evaluate the fitness of each individual [59].

Parent selection and crossover

For each new generation of the GA a group of individuals is chosen for parentage of the following generation. These individuals are chosen based on the fitness proportionate roulette wheel method. Out of these, there is a crossover applied by pairs of two parent solutions to create new solutions (offspring). To perform this crossover, a new biased temporal crossover operator was built on the concepts proposed by Quiroz-Castellanos et al. [72], which aims to encourage the passing on well-fitting genetic material to the following generation. Within each gene, which represents a time step, all active instances are sorted by decreasing average rate of capacity utilization and cost per time slot. By using a zip-merging approach, a new partial solution is then created from both parent solutions. Following that, by removing hosts of already assigned applications the partial solution is then pruned. If any instances violate the constraints outlined in the problem formulation, they will be pruned in the subsequent time steps of the crossover process. Culling instances from solutions may break some chromosomes, as applications can end up with no or only partial assignments. This issue is addressed by employing a basic heuristic to reintroduce any applications that haven't been completely assigned [59].

Mutation

This operator is used on freshly produced offspring at random to introduce new genetic characteristics to individuals and enhance the populations overall fitness. The concept of dominance, which has been introduced by Martello and Toth [64], can lead to tighter packing patterns by replacing a subset of items with an item of larger or equal size. This approach has been shown to be incorporable into the mutation operator of a GA [49, 72], though for usage in this work the definition of dominance has to be adapted. Application a, hosted by by instance $i \in I$ for the time slots $[S_t, F_t]$, dominates a partition of apps P from instance i if the time span denoted by $[S_t, F_t]$ contains all assignments slots of the partition for the respective host. Additionally, the probability of the resource demand of the dominating application being higher than the summed up capacity requirements of all elements of partition P, has to exceed fifty percent. Applying the mutation operator on an individual results in the removal of a number of instances from the portfolio. Each application that is now unassigned, is checked against candidate hosts from the portfolio. By creating partitions of applications of the size two from the respective candidate instance, we can try to find partitions that are dominated by the unassigned applications. Should one be found, the application is swapped with the partition. As these operations may leave broken chromosomes with unassigned applications, just like the crossover operator, the insertion heuristic mentioned previously is used again, in order to repair those corrupted individual [59].

Insertion heuristic

One of the main constraints of our problem formulation is equation 3.5, which requires each application to be assigned to a fitting instance at any time slot of its run time. As previously mentioned, the crossover and mutation operators may result in violation of said constraint. The insertion heuristic chosen to alleviate this problem uses a naive first-fit approach, which chooses a candidate instance from the existing chromosome to fit orphaned applications. Should there be a non-fitting host in the portfolio, a new random instance is generated to accommodate the corresponding application. The element of randomness is used on purpose, to reduce the risk of converging on a local optimum by creating additional genetic diversity [59].

Survivor selection

The process of survivor selection decides which individuals at the end of an iteration are going to represent the next generation. This can be achieved in several ways, with one of the simplest being the selection of the fittest individuals. While there exist more sophisticated methods, and simply choosing the best individuals could lead to a lack of diversity and converging on local optima, both our crossover and mutation operators introduce enough randomness, meaning that the selection of the fittest individuals is sufficient for our algorithm.

Termination

For termination, our GA can use one of several common stopping criteria, like a maximum number of generations, the fitness scores of the individuals of a population converging to a certain degree and therefore becoming very similar, or the highest level of fitness of an individual not changing any more with more generations [59, 77].

3.4. Evaluation and results

In this section we will discuss how the previously described algorithms were evaluated, and discuss the results of this evaluation in order to present what kind of optimization results a customer could expect from the cloud portfolio optimizer. Further examples created in the Cloud Portfolio Manager platform will be discussed in chapter 5. To evaluate our optimization heuristics we used synthetic data. The implementation of the algorithms were done in Python 3.9 and the tests conducted on a PC running a Windows operating system with an Intel Core i7-4770 processor (3.4 GHz base clock, 3.9 GHz turbo) and 16 GB of DDR3 memory at 1600 MHz [59].

3.4.1. Data set description

With our optimization problem there are multiple dimensions, which influence the difficulty of any test set. As is the case with any bin packing problem, a main contributor to this is

number of items to be assigned. Unlike many other bin packing problems, our problem also has to regard the temporal component as main driver of execution time, with the number of allocation periods increasing the difficulty. Therefore the created test data sets reflect a variety in both these attributes [59].

App. Set	Non-Pre.	Pre.	Avg. Res. Dem.	Std. Res. Dem.	Avg. Res. Dev.	Std. Res. Dev.	Avg. Alloc. Periods	Std. Alloc. Periods
apps_1	14.0	6.0	3.27	1.71	0.53	0.48	43.15	33.4
apps_2	59.0	41.0	3.0	2.62	0.53	0.74	63.93	43.94
apps_3	10.0	10.0	3.02	2.01	0.71	0.63	212.2	167.88
apps_4	42.0	58.0	3.1	2.57	0.5	0.59	237.16	171.57
apps_5	7.0	13.0	3.12	2.69	0.6	0.56	2758.55	1996.98
apps_6	41.0	59.0	2.78	1.97	0.49	0.57	2871.74	2055.67

Table 3.2.: Summary of application data sets

Even though the test data used for the evaluation was synthetically created, it takes into account realistic scenarios, including anticipated price discounts for spot and reserved instances compared to on-demand instances. This was based on observations done on the major cloud service providers AWS, Google Cloud and Microsoft Azure. Additionally, for the creation of our test instances, the price to capacity ratio has been modelled similar to offerings observed on the previously mentioned CSPs. The table 3.2 above shows our chosen six sets of applications with their key resource demands and allocation characteristics, while the table 3.3 below describes the three sets of instance types created for testing. These were combined in the following pairings to create six test cases: *case*_1 (*apps*_1, *types*_1), *case*_2 (*apps*_2, *types*_1), *case*_3 (*apps*_3, *types*_2), *case*_4 (*apps*_4, *types*_2), *case*_5 (*apps*_5, *types*_3) and *case*_6 (*apps*_6, *types*_3) [59].

Instance Type Set	Number of Instance Types	Avg. Capacity	Std. Capacity	Avg. Res. Prc.	Std. Res. Prc.	Avg. On. Prc.	Std. On. Prc.	Avg. Spot Prc.	Std. Spot Prc.
types_1	500.0	9.60	8.77	2.27	2.85	3.10	2.16	2.53	2.21
types_2	500.0	10.32	11.40	2.16	2.38	3.13	2.57	3.15	4.84
types_3	500.0	9.79	9.91	2.41	3.80	3.10	2.41	2.33	1.74

Table 3.3.: Summary of instance type data sets

3.4.2. Results

The results of our tests focus on three key criteria for evaluation: speed of execution, packing density and overall costs incurred by the resulting portfolio. In order to avoid side effects other tasks running on the test machine may have, each algorithm was run 10 times to get data on execution speeds.

As you can see in Figure 3.1 and Figure 3.2, the optimization approach *ERICH* offered way faster execution speeds, being faster by the magnitude of close to 10. It also results in an almost static execution speed by being deterministic. In contrast, the optimization approach *GEORG* is not only slower, but also way more volatile in terms of execution speed. The slowest run can take 2-3 times longer than the fastest one, which is due to genetic operators being highly influenced by randomness [59].

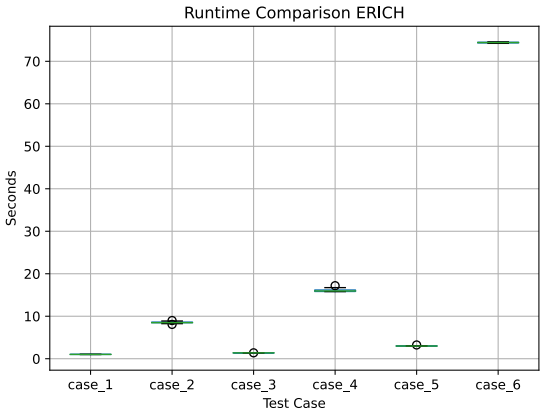


Figure 3.1.: Execution time ERICH

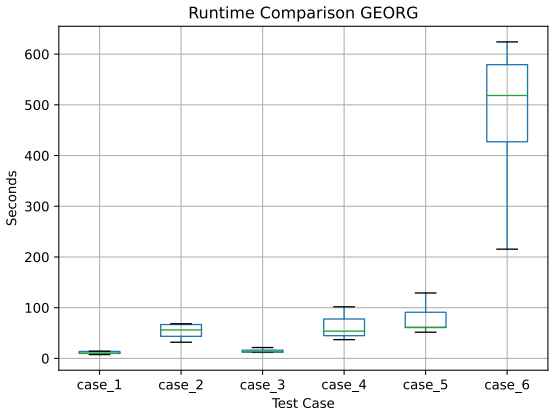


Figure 3.2.: Execution time GEORG

Our second criteria, the utilization rate, measures the relationship between the total expected resource demand of all assigned applications across relevant time slots and the absolute capacity provided for that time period. Depicted in Figure 3.3, it is easy to see, that once again *ERICH* delivers better results than *GEORG*. Depending on the test case, the gap can range between rather minor like in case 1 to very significant as in cases 4 and 6 [59].

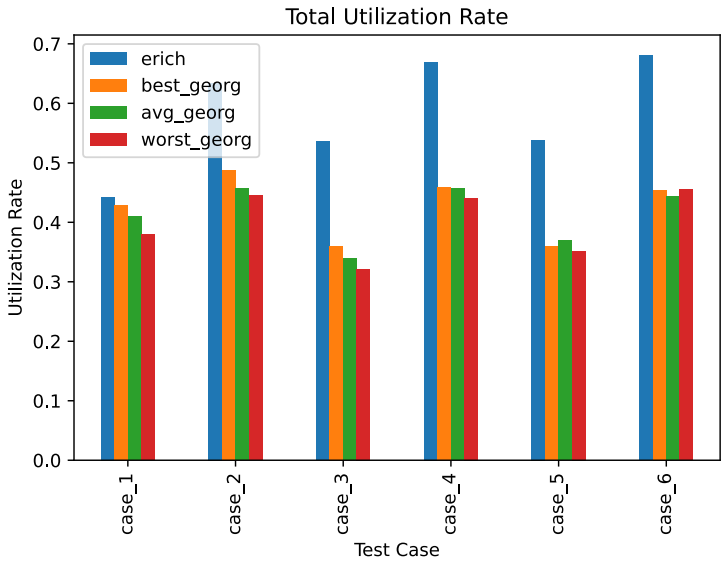


Figure 3.3.: Utilization comparison between ERICH and GEORG

Finally, in terms of overall costs of the generated portfolio, *ERICH* also outperformed the GA by a significant margin, which can be seen in Figure 3.4. While at first this may lead to the conclusion, that the GA did not work properly, this is not the case. In Figure 3.5 one can observe, that the costs, meaning the fitness level of the multiple generations for the data set *case*_6 improve continuously, with each new generation being fitter than the previous one. The initial population starts with a very high degree of genetic diversity and improves with each generation. After 10 generations, which is the duration the test ran, the average costs have been reduced by more than 50%. As the initial population of the GA can serve as an example, how an unplanned allocation of resources would look like, it also shows that both algorithms can delivery notable lower costs for a portfolio and therefore offer a usable basis to build our Cloud Portfolio Manager platform upon [59].

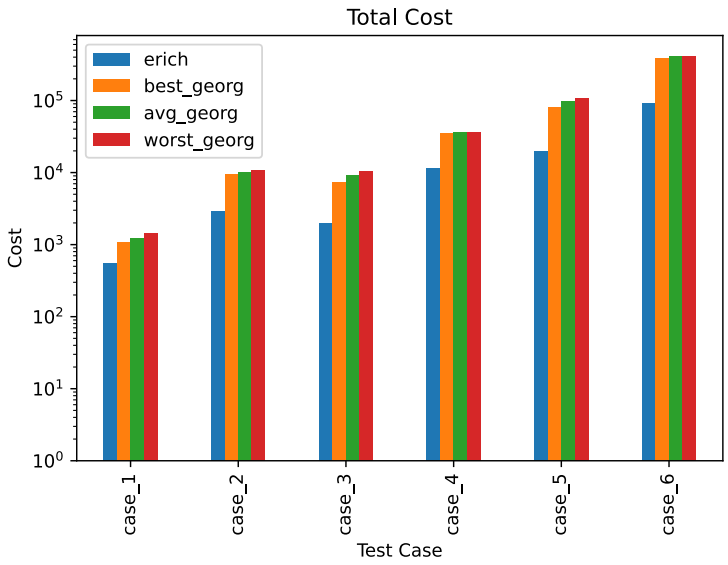


Figure 3.4.: Portfolio comparison between the two algorithms

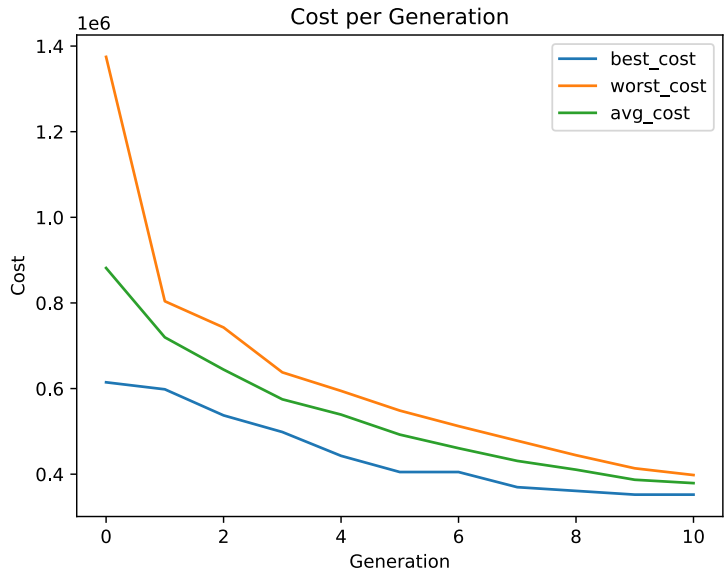


Figure 3.5.: Cost for each generation for set *case*_6 using GEORG

4. Business model

Within this chapter we will first propose a few possible use case scenarios for our platform, discussing both a best and a worst case scenario, followed by one that falls in between. Furthermore, this chapter will present one of the two main contribution of this thesis, a comprehensive business model upon which our Cloud Portfolio Manager could operate. Besides the main model, which will be described within section 4.2, we will also mention additions that could be made to the base model in regard to different business models. Afterwards, we will discuss how our model can be classified within the context of various classification approaches. Finally, we will compare our model to similar platforms like Spot.IO, which is a platform offering a service with some similarity to our own.

4.1. Use cases from a business perspective

Organisations have drastically varying needs regarding their IT infrastructure, which may be influenced by factors, such as the business domain or technical aspects of managed applications. As a result, what may work in theory, cannot always be applied to solve real world use cases. In this section we therefore will outline a number of scenarios and discuss potential benefits of our proposed approached to active cloud portfolio management. When discussing a best-case scenario in this section, we are talking about an optimal scenario creating a need for a business like our Cloud Portfolio Manager to offer its services. Conversely, when considering worst case scenarios, we deliberated over scenarios and markets in which our Cloud Portfolio Manager would find no purchase.

4.1.1. Best case scenario

Suppose a company develops a SaaS product that utilises machine learning technology. Its requirements regarding the provisioning of computing resources will likely be split into two categories, one being the day-to-day operations and the other being the development process.

Depending on the offered functionality, the needed infrastructure may include - among other things - web servers and storage to host static websites, business logic, data persistence applications. Modern day software architecture is often built around the concept of micro-services. Rather than having one monolithic application that provides all the previously mentioned needs, an array of independent services is used. Each of these services usually only features a very limited subset of the required features, but they bring along the benefit of more flexible deployment patterns and easy scalability [38, 57].

The main advantage of this architectural design principle in the context of cloud portfolio management is the fact that micro-services can be spread across multiple hosts

more easily. It here now becomes clearer, how this setup can relate to the problem formulation given in section 3.2. The DevOps team needs to provide an appropriate infrastructure for the set of services that it wants to operate. The host instances have to be selected and packed in a manner such as the desired quality of service is satisfied, while also being as cost efficient as possible. Assuming that the load on the individual services varies over time, it might be advisable to factor in the impact of scaling out the application for short periods. During the horizontal scale-out process, additional hosts are added in order to provide more computational resources. In case this is only done to be able to handle certain periods of higher loads, a subsequent scale-in step will follow. Our proposed optimization approach does account for services, which only have a limited lifespan, with its temporal component. Time is treated as a further dimension to optimize for, allowing better scheduling and dynamic reshuffling of the cloud portfolio.

So far we have discussed how the proposed solution might assist in the cost-efficient operation of SaaS companies. However, in this particular scenario, it was also assumed that the developed product uses components based on machine learning concepts. It is the nature of the underlying technology that the development of such, requires substantial resources. The training process involved in creating machine learning models can be quite demanding in terms of time and needed computational units. However, the same process is also quite well suited for a concept called checkpointing, which describes the act of continuously saving progress in order to be able to resume smoothly after an abrupt termination. This feature has attracted quite a bit of interest for neural networks and other machine learning applications [44, 75]. Furthermore, it is already provided out of the box in many popular machine learning frameworks such as PyTorch or Google's TensorFlow [31, 33].

As has been outlined previously in subsection 2.1.4, the spot market offers highly discounted prices for instances with a rather lifespan, with the additional caveat, they may be preempted at any time. Because of that, such spot instances are not well suited for long running jobs or workloads that do not tolerate any interruptions. Although the required time for the training process may vary from model to model, spot instances can be utilised for this purpose because of checkpointing. In case any interruptions occur, services can be assigned to newly booked hosts, or to the ones that are already available in the current portfolio.

4.1.2. Worst case scenario

As an example for a very badly fitting scenario for our Cloud Portfolio Manager, one could imagine a company running a constant amount of applications, which do not significantly vary in load over a long time. Additionally they do not lend themselves to being easily terminated and restarted. One possible scenario for this could be a monitoring service that observes a somewhat constant flow of data over an extended period of time, as could, for example, be found in a hydroelectric plant monitoring the flow of water.

These cases would not easily benefit from the functionalities offered by our cloud optimization approach. Firstly, the solution to such a problem is somewhat trivial, with a constant need in computing power and little to no variance. Instead of a fluctuating need

with some applications, either running only for a subsection of the entire planning period, or having a strong variance in computing needs, the company only has to account for a constant baseline requirement. This results in no need to ever choose on-demand markets over the reserved market spaces, as the latter always offers lower prices. Additionally, with a lack of varying resource demand and easily stopped and resumed applications, the spot market cannot be considered a viable option in this scenario because it will often result in applications being terminated on a short notice. Therefore, while our optimization approach would find a solution to a scenario like this, said solution is too trivial that it can offer any meaningful value proposition to the company described.

One more type of unsuitable scenarios for which our Cloud Portfolio Manager could not be applied, are those cases in which cloudification is not feasible due to data security requirements. While for most purposes the data security policies offered by major CSPs are sufficient, there are exceptions. Examples of such are applications are those which require the data to stay within geographical boundaries, or those which handle data of the highest security concern, like military technology. Once again the previously mentioned power plant is also a good example, as having the software monitoring and/or controlling a hydroelectric power plant hosted in the cloud in another country is a security concern unlikely to be accepted by any country.

4.1.3. Mid case scenario

After having discussed possible best and worst case scenarios, we should also discuss possible scenarios falling in between those two extremes. Cases in which our Cloud Portfolio Manager could be applicable and useful, but present some challenges to successful usage. As an example we could hereby think of a medium sized company offering an IT service like a financial management tool in a SaaS environment. Depending on several factors, such as the time of day or the weeks leading up to the end of a financial quarter the service would experience varying levels of load.

A scenario like this would, on first glance, perfectly lend itself to being optimized by our Cloud Portfolio Manager. There is need for deployment of several instances of an application with varying load requirements over a certain amount of time, which offers opportunity to separate the requirements into a baseline covered by instances procured on the reserved market and peaks covered either by on-demand or spot instances. Here we can already find two potential challenges: Firstly, depending on the way the application architecture in question works, spot instances could not be usable for running instances of the financial management tool. Only if the design allows for an easy switching between instances of the application and ensures data integrity in such cases, can spot instances be used to cover peak loads. Otherwise only the more expensive on-demand market can be used.

The second problem that comes to mind, is that of good and accurate load profiles, which are needed by our application to properly calculate a good cloud portfolio. A recent industry study by Tamburri et al. suggests, that many companies do not value monitoring highly and often use very crude tools to do so [80]. In our scenario it could be possible that, for example, the company recently received a host of new customers

resulting in previous load data and predictions becoming less reliable and accurate. In such cases our Cloud Portfolio Manager may initially come up with worse solutions, but if the company in question is able to continuously supply up to date load profile data, our algorithm would be able to overcome this challenge and provide good solutions.

4.2. Cloud Portfolio Manager business model canvas

This subsection will present the business model of the Cloud Portfolio Manager with the Business Model Canvas framework by Osterwalder and Pigneur [70], which will help us describe our model with the help of the nine building blocks, already outlined in the state-of-the-art section 2.2.

4.2.1. Customer segments

For our customer segment it seems clear that the sole focus will be on the B2B area, as consumers, so far, have little to no reason to purchase cloud resources for personal use. Within the B2B sector a big emphasis will be put on the IT sector ranging from small businesses to large companies, seeing as large parts of this sector move more and more of their IT infrastructure into the cloud. One example would be Netflix, which since 2015, has moved its entire IT infrastructure to AWS [5].

Surprisingly, when it comes to smaller businesses, as suggested in a study by Jonas et al. in 2013, start-up companies looking for cloud solutions prefer the reputation of a cloud provider, over other aspects as price, security and reliability [74]. Being, at least at the beginning, a small and not well known company, could mean targeting this customer segment could be more challenging than others. This also leads us to the conclusion that establishing a reputation should be a main goal during the establishment of the business. Besides this, increasing our attractiveness to small businesses and start-ups can also be achieved by offering cloud consulting services in addition to our optimization service. These services aim to help customers to better understand cloud environments and set up and operate their own cloud portfolios. While it can be expected that larger companies already have access to this knowledge, the same cannot be said for smaller businesses. Aided by our consultancy services, we can also market our optimization service to this customer segment.

One customer segment our business will specifically focus on within the IT sector, are those companies, businesses and possibly research facilities, working with machine learning algorithms. These are uniquely well-suited to be run on cloud resources, as their implementations often offer out-of-the-box preemptibility like the previously mentioned PyTorch and Google TensorFlow [31,33]. The combination of machine learning algorithms, the applications being very resource-intensive and our optimization approach delivering the best results for preemptible applications, make them a perfect match. Our value proposition, which will be detailed within its own building block, is well suited to reduce one of the great pains of implementing machine learning solutions, the lack of processing power by offering a cheaper access to computational resources.

While the previously described customer segment will be our main focus, other sectors also offer a wide range of potential customers. Even back in 2017 Mohit et al. suggested that over 90% of organisations were either already adopting cloud infrastructure or planning to do so within the next one to three years [37]. It can therefore be expected that the potential market for our cloud resource optimization approach is both large and very diverse.

4.2.2. Value proposition

Our main value proposition, a cost reduction for their cloud portfolio, is unlike many other business models, targeting all customer segments alike. This can be applied to both existing portfolios and first time cloud deployments, as long as the customer is at least roughly aware of their applications computational needs and run-time. The setting up of a portfolio can be done in two ways: First, directly via interacting with the platform via its website. Alternatively, most interactions with the platform, outside initial registration, can also be done via API, allowing customers to integrate our service within their own systems and automate the process of creating portfolios and creating allocations.

The previously mentioned cost-reduction is achieved by a combination of choosing the cheapest instances from the right market-place and a reduction of their idle time, through continuous monitoring of the needs the applications. The resulting benefit for the customer does not only entail the direct cost reduction itself, but also offers an easier access to the complex world of cloud computing, which could be especially useful for small and medium sized businesses. For these clients in particular, we can extend our value proposition by offering a consultancy service aimed towards customers that do not yet possess the know-how needed, to take advantage of cloud solutions. Said service would focus on the basics of cloudification such as which applications are viable for being put into the cloud, the creation of portfolios and first-time deployment.

4.2.3. Channels

Listed here, we will address the channels used throughout the five phases of customer interaction (for details see subsubsection 2.2.2), through which we aim to reach our customers:

- **Awareness:** For the first phase, the raising of awareness of the customer to our service, a mixture of online advertisement and direct contact with potential customers through an in-house sales force seem applicable. Furthermore, targeted online advertisement, for example via Google ads[1], could be considered as an option. While having the potential to result in a higher click-through rate, the advertiser has to be careful not to be too intrusive, as this can result in having the opposite effect. It also appears that targeted advertising does not work well with every demographic [39]. After the first customers have been acquired, it can also be reasonably expected that word-of-mouth between different businesses could

[1] `https://ads.google.com/intl/de_at/home/`

raise further awareness levels for the business. Another option to reach potential customers are trade fairs with a focus on the IT sector.

- **Evaluation:** The main channel used for evaluation, and centre of the operation, is our website. It provides example calculations, which showcase the potential cost-reductions offered by the service, gives an overview of the pricing structure and offers a guidance on how to set up an account. Later, customer success stories, a widely adopted practice among online businesses today, will be showcased on the website. These stories serve to further highlight tangible implementations of our service and provide credibility to our claims of assisting customers in optimizing their cloud portfolio.

- **Purchase:** Regarding the purchase of our products a wide array of online payment services such as PayPal, Amazon Payments and Credit cards are available and can be implemented with relative ease. Besides directly integrating single payment options, companies such as Stripe[2] offer a single API which enable the user to choose from a range of common online payment options.

- **Delivery:** The delivery of the optimization service to the customer can also be achieved through the direct channel of the platform, either by direct customer interaction on the website or via API call. As for the consultancy side of the business, it is to be expected that delivery would be done via personal interaction, both physical and via online channels, depending on the customers preferences.

- **After sales:** In providing a post-purchase support of customers the Cloud Portfolio Manager will at first mainly focus on doing this through our website. Here the user will be able to have an overview of his portfolios, optimizations, subscription and general account information. A FAQ page can also help answer general questions. For more complex support cases direct support through a personal customer support force should also be implemented.

4.2.4. Customer relationships

When it comes to the methods of interacting with the customer base, the Cloud Portfolio Manager will focus on two areas: For the majority of interactions, such as setting up an account, creating and managing a portfolio, an automated service based on our website will be used. Complementing these services is a sales- and CRM- (Customer Relationship Management) force, which can be contacted personally via channels such as e-mail, phone or video calls. This enables a more personal relationship with the customer and is meant to answer complicated and personal questions, regarding single customers which cannot be served via an automated service as easily. This additional service is available to the customer during the whole interaction from pre-sale evaluation until the purchase is completed. Another important aspect of this building block is the interaction between the consultancy force and the customers. As for offering a consultation service, direct

[2] `https://stripe.com/en-gb-at/payments`

human interaction is preferable, each customer is being assigned and mainly interacts with one consultant. This can be over various channels, but due to the close nature of the relationship, it will result in more face-to-face interactions than the other services of our business. A study by Roy et al. suggests that direct interaction with the customer is also preferable, as service experience is valued even higher than the actual service quality in B2B services [76].

4.2.5. Revenue streams

Regarding revenue streams, a plethora of options are available at first glance. But, as we state in this section, most of them are not easily applicable for our platform for one reason or another, leaving us with one very widely-used revenue stream, as our main source of revenue.

The first option we want to discuss is advertisement. While it is a main revenue stream of many large online platforms such as Youtube and Facebook, these are mass media B2C operations with millions or even billions of users, where each user only generates a rather small amount of revenue through being displayed advertisements. For our platform which offers a specialised service to business customers, advertisements would mostly discourage users and possibly damage the brand reputation [48].

Next, we have considered the option of transaction fees. This could, for example, be implemented on a usage-based model, where the customer would pay a certain amount for each optimization, based on the size of the portfolio. Another implementation could be a one-time transaction enabling the unlimited access to the Cloud Portfolio Manager. Both options are not optimal for our product. The usage-based model does not lend itself well to a product that is meant for continuous optimization. This would either lead to a need to frequently pay for a new allocation or prevent customers from getting the full benefit of an approach that is meant to adapt to changing needs in their portfolio. The one-time charge option faces another drawback making it unfeasible. Seeing as our portfolio manager is meant as a continuous service, this one-time charge would have to account for a long service-time, which in turn would increase the price a level, which would turn it into a serious deterrent for new customers that are not fully convinced of the products benefits yet.

Another alternative would be a system based upon a brokerage fee. In this case, a part of the cost reduction achieved by the Cloud Portfolio Manager would be taken as our revenue. The great flaw with this idea though is, that our system does not aim to directly access and manage the customers cloud instances. This would result in a need for the customers to accurately and honestly report their current cloud expenses and their achieved cost reductions, which lends itself to be abused way to easily.

Finally, we propose that subscription fees are the revenue stream best suited to our business model. They tackle several of the disadvantages mentioned in the previously discussed systems, such as fitting well with a continuously running service unlike pay-per-use transaction fees, and a low entry barrier in comparison to a one-time charge. The subscription fee, which would be due in a monthly interval, could either be based upon a system with different levels of subscriptions, offering support to differing sizes of cloud

portfolios and varying levels of customer support, or directly scale with the size of the optimized cloud portfolios.

As for revenue streams concerning the consultancy side of the business model, three monetization variants are possible: First a classic hourly fee, which would be most fitting for customers needing only a smaller contingent of assistance. Another variant would be offering package deals with a fixed price, such as offering to help setting up the first cloud portfolio for a customer. Finally, higher level subscription models for the cloud portfolio optimizer platform could include a certain amount of consultancy services for free.

4.2.6. Key resources

As for the differing categories of key resources, the following can be said: When it comes to physical key resources, there is not much to be mentioned here. While server resources are necessary to host the platform, the hardware required is highly interchangeable and not hard to come by. Furthermore, it could easily be more advantageous to completely forgo physical servers and host the platform itself on a cloud server as well.

Financial key resources may also not play a huge role to start off with. Of course financial resources such as cash or credit will be needed to set up the business, but, due to its nature, these will not be of huge volume. One possible option to gain access to financial resources in order to start the business would be to apply for one of the many tech start-up sponsorships available in Austria. The most important key resources are within the intellectual resource category, consisting of the portfolio management platform and in particular the optimization algorithms, which are at the heart of the operation, and are needed to realize all of the other components of the business model.

Finally, when it comes human key resources the following groups can be expected to be part of those: Especially for development and improvements to the platform further full-stack developers could be needed. Furthermore, a small team of cloud consultants would be responsible for providing customers with know-how on cloud solutions. Besides that, a group of employees, helping with sales and CRM-related topics, should be employed as well.

4.2.7. Key activities

The most important key activity of the business is the operation and maintenance of the Cloud Portfolio Manager platform. In this capacity, the platform offers the customer an automated service. After having created an account and logging in, the customer has the possibility to create, delete and change their cloud portfolios. Besides a simple interface to directly manipulate a portfolio, the main feature for management consists of the possibility to upload load profiles based upon which an optimized portfolio of instances is calculated and displayed to the customer. The upload of the load profiles can be done both manually on the website and through a REST API.

In addition to the service provided by the platform, the other main activity is problem solving for the customer by offering our consultancy service. This mainly entails the

sharing of cloud related know-how and aiding the customer with planning, creating and monitoring of their own cloud solutions, as well as migrating their existing applications.

4.2.8. Key partnerships

Within this building block the most prevalent partnerships are the various CSPs that the platform offers portfolio optimization for. While actively managing the customer's portfolio is not part of the business plan so far, it is crucial to the functionality of the platform to have access to the instances and their respective pricing, offered by the various providers. Luckily, all major CSPs provide APIs giving access to live data of the current availability and pricing of their offered instances.

If the business grows beyond a small scale, it would be possible that certain activities, such as customer support, or cloud consulting are outsourced to external partners, which, of course, would turn these into key partnerships as well.

4.2.9. Cost structure

The cost structure of the business model is intended to lean towards being value driven, with a focus on creating value for the customer. As the business is operating on an online basis with a web platform at its centre, scaling it should be achievable with relative ease. While an increase in the customer base will require additional staff for CRM and consulting, the platforms performance for handling a certain amount of customers can be scaled almost infinitely and with a mediocre, but easily projectable, impact on costs.

In contrast to the operating costs of the platform, which should not be too substantial, the same cannot be said about its initial creation, that can be expected to be one of the major cost factors in starting the business. Once the platform is in operation, the major cost factors are going to be the personnel required for the CRM and consulting operations and resources spent on updating and expanding the platform. Further costs that have to be taken into account, come from actions taken towards the acquisition of customers, especially those mentioned in the awareness section of the "channels" building block.

The Business Model Canvas

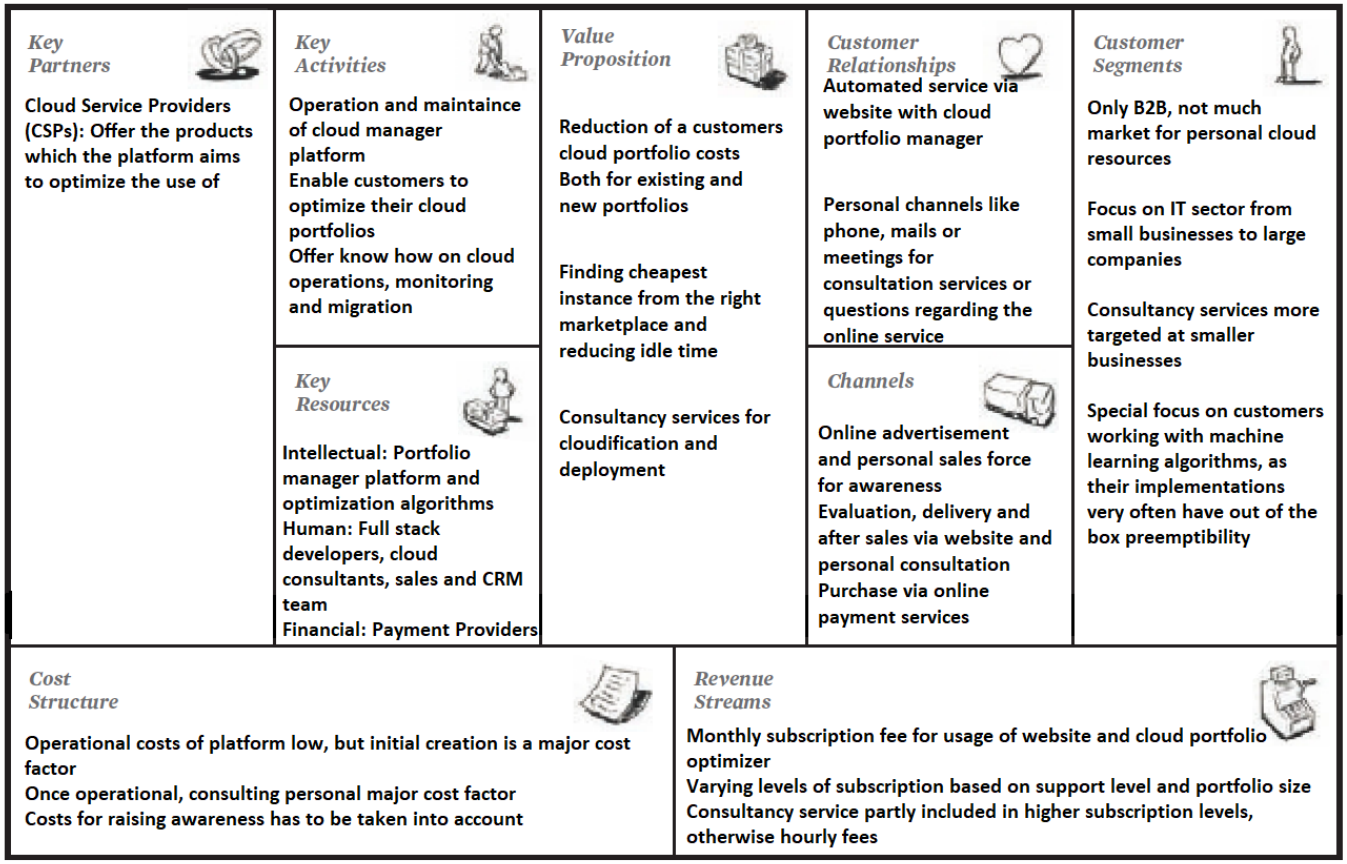


Figure 4.1.: Business Model Canvas [70] of Cloud Portfolio Manager

4.3. Classification of our business model

Having described the Cloud Portfolio Managers business model, it shall now be discussed how our model can be classified within various classification systems developed in the literature. Besides the system offered by Timmers [81], which was already described in section 2.2, we will also apply the classifications of Osterwalder and Pigneur [70] and Johnson [56].

Within the classification system of Timmers [81], both of our business models main services put the platform into the information broker model. Those focus on providing information by analysing or finding data that can benefit the customer's operations, which is the case for both our optimization algorithms and our cloud consultancy operations. Aligning our model along the two axis, Timmers' system uses, we will first find a high degree of innovation with our cloud portfolio optimization, being one of the first ever to offer this kind of service and cloud consultancy being a service that has only emerged in the past few years. Placing our business along the functional integration axis, we find that with two main functions, it falls towards the lower end of this spectrum. Both these placements fit well with the classification as an information broker, as can be seen in the Figure 4.2 below.

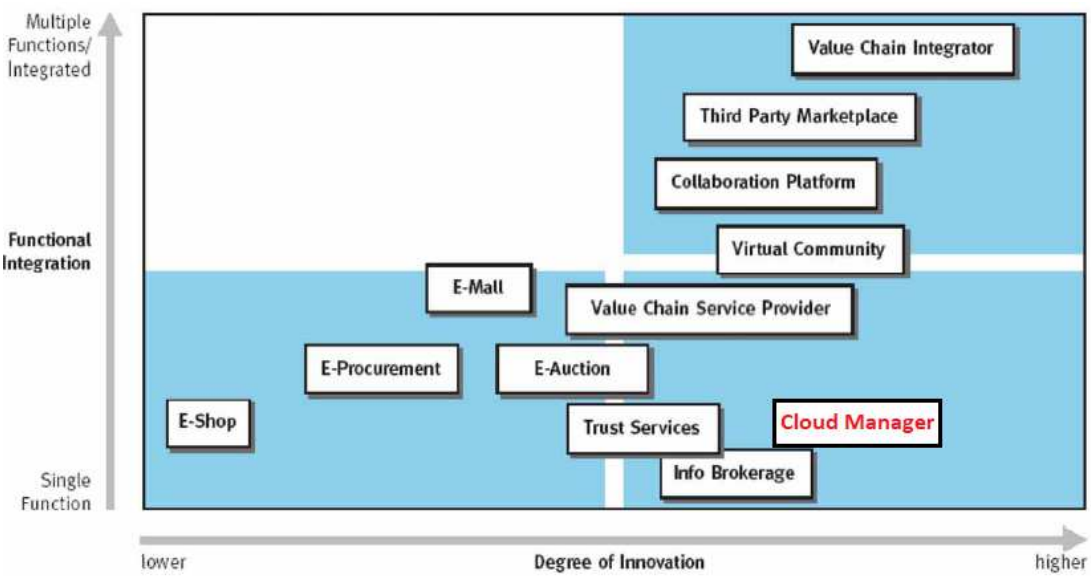


Figure 4.2.: Classification of Cloud Portfolio Manager within internet business models, figure from from B. Wall et al., 2007, Production Planning and Control, page 248 [82]

Osterwalder and Pigneur [70] call their approach to finding business models with similar characteristics and features patterns. They identify five patterns, business models commonly express, which we have discussed in section 2.2.

When comparing these patterns to our own business model, the following can be said:

- In case of the **"Unbundling"** pattern, our business model clearly falls within the category of product innovation, with a service that is quite new on the market and only a few small players present in it so far. The consultancy aspect of our business model could be seen more as a customer relationship management business, meaning that, to prevent this service from conflicting with our cloud optimization service, it may be necessary to separate them into different entities like business units.
- Next, considering the so called **"Long Tail"**, we would argue that our model does not adhere to this pattern. Our business focuses on the optimization of portfolios consisting of very common cloud instances, sold frequently, not making them niche products. We also do not offer a wide range of niche products, but only a small amount of services.
- On the other hand, at first glance, it can be argued that our product is a **"Multi-Sided Platform"** of some kind, as it brings together two interdependent groups and without the presence of CSPs our platform could not exist. On the other hand, while customers optimizing their portfolio are profiting from our service, the same cannot be said for the CSPs themselves, as they rather stand to lose extra revenue generated by unused, but paid for instances running idle. Therefore, as the "Multi-Sided Platform" pattern should be of value for all involved groups, we argue that our model does not conform to this pattern.
- The same cannot be said for the **"Free"** pattern, as our business model will include a small part of our services free of charge. While there will be various levels of subscription that will cost a monthly fee, there will also be a free trial functionality, offering a limited access to the service to lure in potential customers. This approach has been labeled as "Freemium".
- The final pattern to take into account is the **"Open"** business model. While the "inside-out" approach is not planned to be part of our business model, the "outside in" idea could potentially be explored in the future by integrating external cloud frameworks into the cloud optimization platform, which can make the product more appealing to potential customers.

4.4. Comparison to similar business models

It has become apparent during our research into the topic of cloud resource management and businesses operating in it, that, while the market is still relatively new, there are some that have, or are in the process of, establishing themselves in it. Therefore, having extensively laid out and classified a business model for our Cloud Portfolio Manager, this section will now look into how other firms, operating with similar ideas within the same market, structure their business model. In order to discuss this, we will offer a

short description of said businesses and follow this up by comparing the similarities and differences that can be found between these platforms and our own.

4.4.1. Spot.io[3]

First off, we will take a look at the platform of spot.io, which offers a range of tools for customers to analyze, manage and optimize their cloud portfolios. Two of their products offer functionality somewhat akin to our cloud portfolio optimization approach. Elasticgroup uses AI predictions to fully automate infrastructure scaling with the help of spot instances. Eco, on the other hand, tries to optimize the customers cloud portfolio by finding and off-loading unused Reserved Instances and Saving Plans. Furthermore, in a similar fashion to our platform, spot.io also offers consulting options their customers, though limited to their highest subscription plan. So overall, regarding the value proposition, this firm shows some similarity with our business model.

Spot.io offers three different plans to access its service: A free plan that supports a maximum of 20 instances and the basic functionality of the platform, a basic version with 24/7 support which is billed monthly and finally a professional version with unlimited scalability, training and consulting options. The professional edition is billed yearly and has no fixed price rates, but has to be negotiated based on usage. While the multi-tiered approach is very common and also part of our business model, there is a contrast in how the business is generating a revenue stream, as spot.io bills its customers based on the percentage of savings realised. While, as discussed previously, this approach of broker fees does not make sense for our platform, as we do not plan to directly access and manage the customers cloud portfolio, it makes more sense for spot.io, which does require access to the customer's cloud portfolio and therefore does not have to rely on accurate and honest reporting.

4.4.2. Densify[4]

Next up is the platform Densify, which offers a cloud management and optimization service with a similar value proposition to our platform. It uses machine learning algorithms and analysis of workloads to try and find the optimal resource allocation for cloud portfolios. On one hand, there is a stark contrast to our platform in regards to revenue stream and pricing model. Besides a 14-day free trial, Densify charges the customer for each managed instance per year, with the price depending on the amount of instances. While there could be no exact prizing model found on the website of the company itself, the service offered on the AWS marketplace[5] revealed the following: Prices start with $48.3 per instance and year for 1000-2000 instances and drop as low as $39.12 for 5000+ instances. So, besides the high prices and seemingly not offering solutions for a portfolio of under 1000 instances, Densify does not seem to have an incentive to optimize a customer's portfolio to need less instances, as they charge per instance.

[3] https://spot.io/
[4] https://www.densify.com/
[5] https://aws.amazon.com/marketplace/pp/prodview-hjixcyl4w5m7g

4.4.3. Terraform[6]

The final cloud optimization platform discussed is Terraform. It allows users to express their computational infrastructure needs in their own semi-structured language, which then can be deployed to a range of resource providers like AWS or Google Cloud. The value proposition is to simplify and enable management of infrastructure across multiple cloud providers. While there is also the capability of easy scalability of resource needs, it does not seem to be any optimization of the cloud portfolio to be going on. Therefore, this platform's value proposition does not directly compete with our Cloud Portfolio Manager, but could work in a complementary way. This is not just mere theory, but is already in use by Terraform, as it offers a module for easy integration of the previously described Densify to help optimize the user's cloud portfolio and reduce the amount of wasted resources. On the revenue stream side of things, Terraform has opted for a multi-tiered subscription model in combination with a pay-as-you-use model[7]. The free version enables the usage of basic functionalities and seems to be tailored for one user. The "Team" Tier costs $20 per user and month, and enables collaborative work via role-based access control. Next up, the Team & Governance Tier offers extra services like policies and cost estimation, costing $70 per user and month. Finally, the business version, as seen with previous platforms, once again offers the widest range of capabilities with an undisclosed price that has to be arranged with Terraforms sales force.

[6] https://www.terraform.io/
[7] https://spacelift.io/blog/terraform-cloud-pricing

5. Cloud Portfolio Manager implementation

Based on our business model and incorporating the implementation of the algorithms developed by Kiessler and me, we created a prototype of a cloud portfolio platform [59]. This section will discuss the requirements, architecture and implementation of this prototype, as well as a short evaluation of the user experience (UX) the prototype delivers.

5.1. Requirements analysis

This subsection will outline the various functional and non-functional requirements that the prototype of our platform should implement. It also plays a major role in governing the type of technology stack that will be used for its implementation, which will be detailed in section 5.3.

5.1.1. Functional requirements

With the aim of this thesis, being the creation of a business model for a cloud portfolio management platform, the implementation created in tandem, should be able to give a good idea how such a platform could look and operate for an actual business, while still being a proof-of-concept prototype with reasonable scale. This fact informs both functional and non-functional requirements. We have therefore come up with the following functional requirements:

- **Persistence**: The prototype should have a form of persistence, allowing for both data created directly by users and the results of applying the optimization algorithms to be saved outside the program's run-time. This can be achieved in several ways, though in order to be closer to a real life platform, it should not be in a file format like CSV, but using a database model and some SQL or NoSQL dialect.

- **User interface**: Serving as a prototype for an end-user platform, some considerable focus should be diverted to the user interface. It is therefore advisable to create a fully-fledged graphical user interface (GUI) to allow users, not accustomed to operating programs via a command line interface, to use our platform. Furthermore, a usable and intuitive UI is a prerequisite for a modern web application, which the Cloud Portfolio Manager aims to be.

- **Data visualisation**: In order to provide the customers of the platform with a straightforward and fast means of interpreting and comparing the results of our optimization approaches, there should be some simple data visualisation function offered. This does not have to be overly complicated, but could be achieved via bar charts or comparable visualisation techniques.

- **API support**: While interacting directly with the platform via a web UI is a central function of the platform, it should not be the only possible way of interaction. The Cloud Portfolio Manager should therefore offer various APIs that allow customers to perform most action that can be done with the web interface. This also allows easier integration of the platform into various existing systems of a customer.

5.1.2. Non-functional requirements

- **Performance and parallelism**: Seeing, as the optimization approaches described in chapter 3 are of rather high computational needs, the platform built around them aims to be rather lightweight, in order to not put too much further strain on the system it is deployed on. Furthermore, the prototype should be able to run normally, while optimization calculations are ongoing, meaning it has to be able to asynchronously run the computations needed.

- **Adaptability**: The prototype of the cloud portfolio management platform is expected to be implemented in a way that it would be feasible to adapt it to a working platform offering its services to customers. This includes possible future adaptations to the modelling of concepts such as applications, instances and portfolios, as well as the addition of further optimization methods to the platform. Moreover, while the prototypes scope does not include directly accessing CSPs, getting live data for instances and prizes should be feasible with little adaptations to the program.

- **Interoperability and scalability**: The prototype should not require extensive setup or a special ecosystem to be deployed. Therefore, it is desirable for it to utilize widely used and proven technologies and dependencies. In addition, the deployment of the platform should be scalable to larger operations without requiring a major overhaul of the systems design and implementation.

5.2. Architecture

Within this part of the thesis, we will use the 4+1 View Model created by Kruchten [60] to formally describe the architecture of our system. This model uses several diagrams belonging to the different views, to describe a software system covering varying viewpoints of the product.

At first scenarios, representing the +1, describe the most important use-cases of the system, which also showcase, how the other four views work together to enable them. Use-case diagrams are usually chosen to describe this view. The logical views main objective

is to support the functionality the system provides to the users. By abstracting these, we get objects, which can be well modelled by a UML (Unified Modelling Language[1]) class diagram. The process view focuses on how parts of the system interact and communicate with each other, taking into account non-functional requirements like performance and scalability. Activity and sequence diagrams are the most common methods used to represent this view. In our case we have decided upon using activity diagrams. Showcasing, how the modules and subsystems of the software are organised and aiding with separating the software into parts that can be developed by different people and groups, is done via the development view. Most commonly used for this view is the Component Diagram. Focusing on the physical configuration of the system is the so-called physical or deployment view. Via a deployment diagram, this view models components, like different servers and their interconnections [60].

5.2.1. Scenarios - use case diagram

The first diagram of the view, the UML use-case diagram, visualises the various functionalities the application will offer and the actors involved. In our case of the Cloud Portfolio Manager there is only one type of actor, which is the customer. The various use-cases are divided into the necessary system use-cases on the right side and the main use-cases on the left side of the diagram. The former consists of basic use-cases most online platform will have to implement. First, there is the registration of a new user in the system, which requires a valid e-mail address, a username and password. This, in combination with the use-cases of logging into and out of the system, enables access to the platform.

Depicted on the left side are the main use-cases needed for the creation and management of cloud portfolios. Starting at the top of the diagram, we have the creation, updating and deletion of applications, which form the building blocks for any portfolio. Below that are the use-cases for the management of portfolios consisting of the same three operations as for handling applications. Finally, we have use-cases for allocations, which only include "create allocation" and "delete allocation", as allocations cannot be adjusted once created. Allocations cannot be created without a portfolio, which in turn cannot exist without applications which are depicted in the diagram via the "include" relationships for allocation to portfolio and portfolio to application. A program implementing the described use-cases should be able to satisfy the basic requirements for a Cloud Portfolio Manager.

[1] `https://www.omg.org/spec/UML`

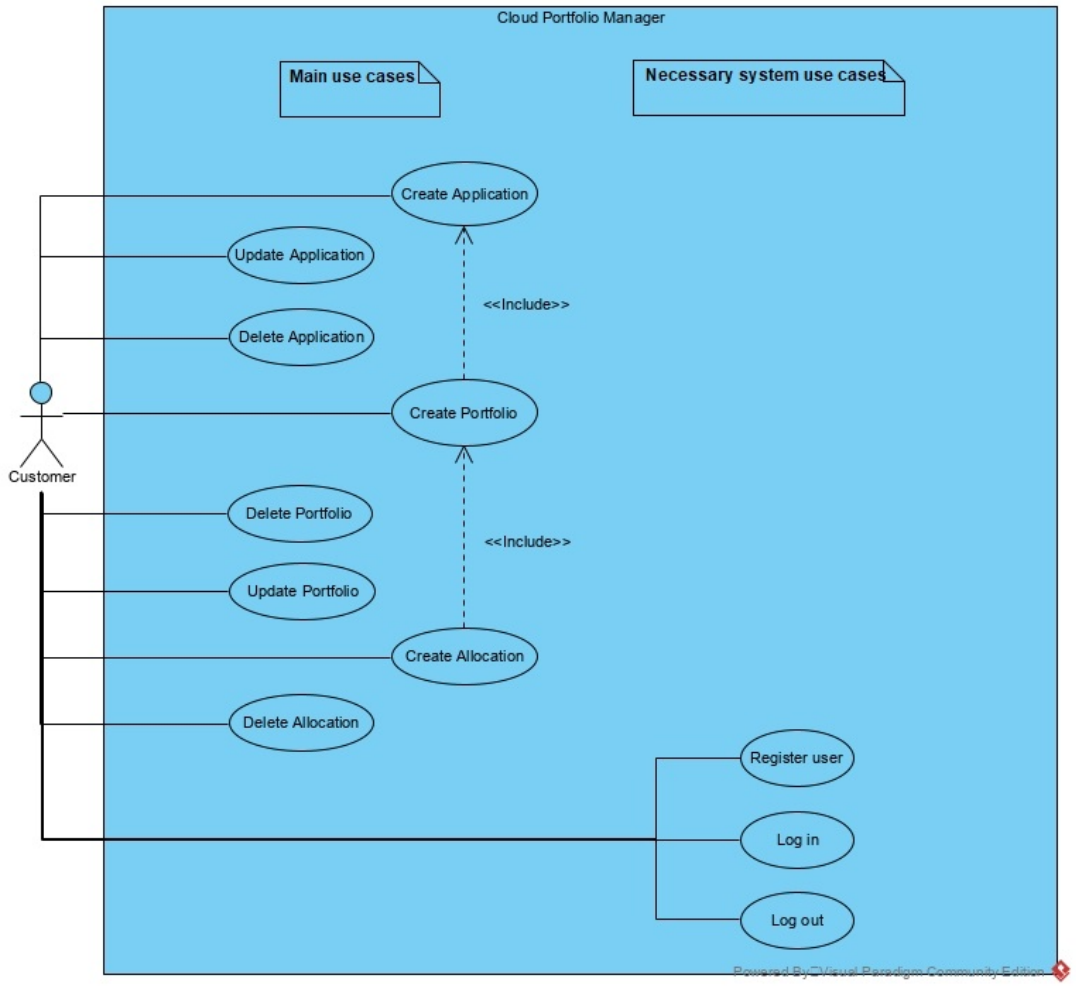


Figure 5.1.: Use Case Diagram

5.2.2. Logical View - class diagram

Our next view is the logical view, which is represented by a class diagram depicting the main classes, their variables and some important methods of our system. In our case the entity user is central and most other classes are in some way associated with it. Besides its variables username, password and e-mail, it contains methods for the registration, logging in of users and the management of both portfolios and applications, which is modelled by them being composed by the class user.

The application class contains information, like its starting and finishing time, resource demand and whether it is preemptible or not. Meanwhile a portfolio has attributes concerning the number of applications and the minimum quality of service, which states the percentage of time the applications in the portfolio have to be running. Furthermore,

portfolios have a version which helps keeping track of changes, a list of providers that can be used, and functions that help create and delete allocations. Applications can be aggregated into portfolios, which each portfolio consisting of at least one application, but each instance of that class can be part of any amount of portfolios. Providers are depicted as an enumeration and, in our version, contain the CSPs Google Cloud, AWS, Alibaba and Azure. To model timestamps we have the class *DateTime* with the attributes year, month, day, hour, minute and second and an enumeration for the months of the year.

In composition with portfolio we have the class allocation, the creation of which represents the main goal of the whole Cloud Portfolio Manager. The class itself does not contain much data about the allocation besides which approach is used and for which portfolio version it was created. This is due to most of the allocation's information being stored in the composed class statistics, which as the name implies, contains statistical data concerning the allocation. Additionally, if the optimization approach is the GEORG algorithm, the allocation is also a composition of the class configurations, which is used to store data concerning how the genetic algorithm was parameterised for the allocation in question.

The final class of the logical view is the *cloud_instance*, which represents an instance of a cloud container from a CSP contained within an allocation. Its attributes regard which instance type it is, its running time, cost and capacity. They are always associated with one allocation, which, in turn, contains one too many *cloud_instances*. In relation with applications, each *cloud_instance* has one to many applications allocated to it, while an application can only be allocated to one instance.

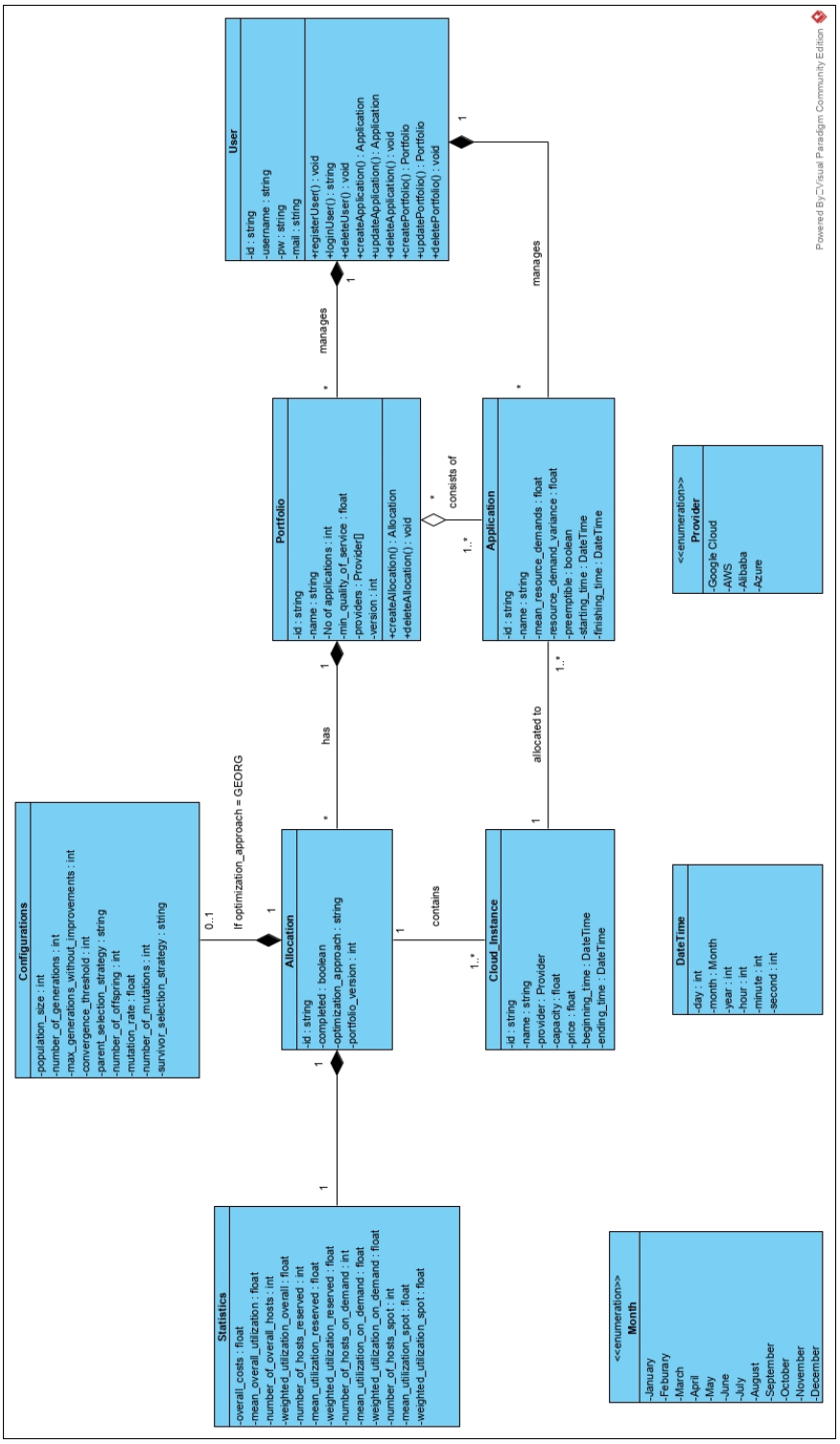


Figure 5.2.: Class diagram

5.2.3. Process view - activity diagrams

Representing the process view are activity diagrams, which show the basic processes contained within the scenarios previously described.

User management:
In this first scenario we deal with the registration and login of users. The swim-lanes used in all activity diagrams shown here, represent the platforms frontend as viewed by and interacted with the customer on one side, while the platform's backend is represented by the other.

On the left side of Figure 5.3 depicted is the process of registration. Hereby the user first navigates to the registration side, where they are presented with fields to fill in their e-mail address, password and username, which can be done in any order. Having filled out all fields the data, it is then sent to the backend, where it is checked for validity. Should the data contain any invalid inputs, like an already existing username or badly formatted e-mail address, an error message is displayed and the process goes back to the point of filling out the input fields. Otherwise, a new user is created, and a message denoting the successful operation is displayed. Finally, the user is redirected to the homepage.

Displayed on the right side of Figure 5.3 is the process of logging into the Cloud Portfolio Manager. After accessing the website, the user is presented with the login form on the homepage. Having entered e-mail address and password, the login data is forwarded to the backend, where it is checked for validity with the login data stored in the database. In case of a wrong input the user receives an error message and stays on the homepage. Otherwise, an access token is generated and returned to the frontend. Here the token is stored enabling later backend calls and the user forwarded to the landing page, from where they can navigate and interact with the platform, finishing the login process.

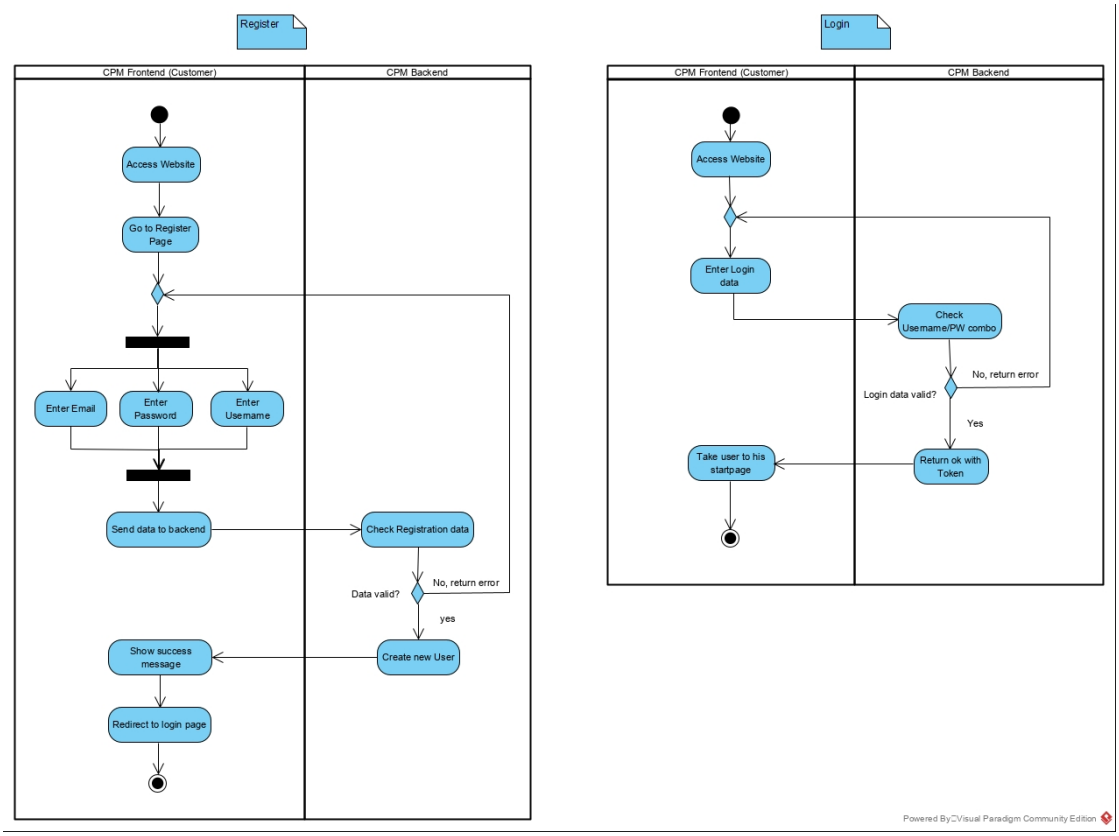


Figure 5.3.: Activity Diagram for Login and Register

Application management:
This next activity diagram describes the various ways a user can manage their applications. After navigating to the application tab, the user is presented with existing applications and has the option to perform the classic CRUD (Create, Read, Update, Delete) operations. Selecting the option to create a new application, the user has to fill out a form, containing running time, resource demand, name and preemptibility of the new application. The data is then sent to the backend, where a new application is saved into the database.

Choosing to edit an application, the user selects one of the existing apps, which opens the same form as for creating one but already filled out. The user then is able to update whatever fields they choose, after which the updated data is processed in the backend. Finally, if deletion of an application is desired, the user chooses which one to remove from the list of existing applications. The app in question is then deleted from the database, as well as from all portfolios containing it.

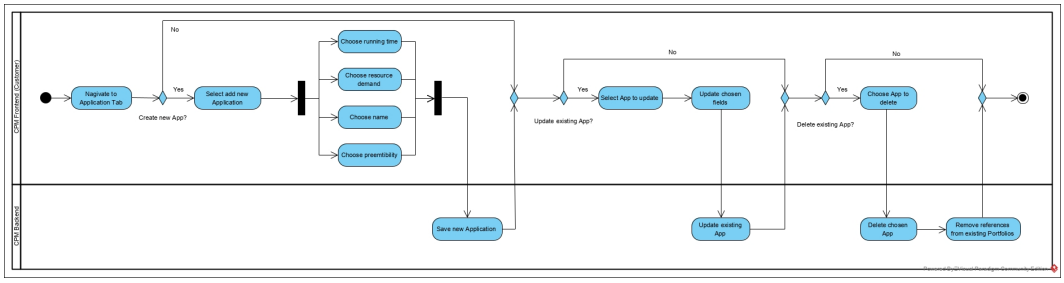


Figure 5.4.: Activity Diagram for Application Management

Portfolio management:
The activity diagram depicted the scenarios concerning the management of portfolios, is almost identical to the previous one about applications. The only real differences concern the form for creating and updating portfolios. These of course, besides the name, require different inputs like the minimum quality of service for portfolios, which providers are allowed to be considered for an allocation and which apps are to be part of the portfolio. Another small difference can be found in the process of deleting portfolios, which also result in the removal of all corresponding allocations.

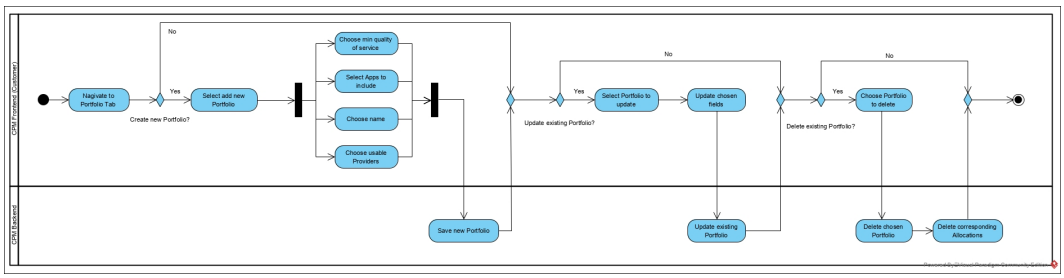


Figure 5.5.: Activity Diagram for Portfolio Management

Allocation management:
The final activity diagram visualises the management of allocations. After navigating to the allocation tab, the user first has to choose a portfolio out of his existing portfolios. For the portfolio in question a view will then be shown which lists all existing allocations for said portfolio, including some graphical representations of the data contained within these allocations. When wishing to create a new allocation the user first has to choose which optimization approach to use. In case of the genetic algorithm GEORG further input is required regarding the parameters used. These include, amongst others, population size, number of generations and mutation rate and allow more knowledgeable users to configure the algorithm to their liking. Once method and parameters are chosen, a new allocation is created in the backend and its calculation starts. As soon as the allocation has been completed, the results are displayed in the frontend. Lastly deleting allocations works identically to the removal of applications and portfolios, only without the necessity of considering dependent entities.

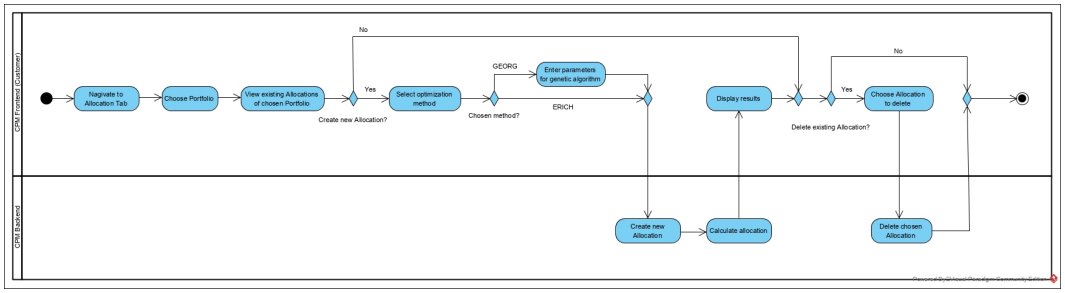


Figure 5.6.: Activity Diagram for Allocation Management

5.2.4. Development view - component diagram

To represent this view, a component diagram was chosen to represent the various components the platform is composed of, how they interact with each other and to which interfaces they are connected to. For this view the application was divided into the major components *CPM-Platform* and *Database*. While the *CPM-Platform* component could be divided further into frontend and backend, it would only make the diagram larger without adding any additional context to the interplay between the various components.

On the left side of the diagram are the external interfaces provided by the platform. Starting of at the top of the diagram, we have interfaces for *Login* and *Registration* which are provided by the *Authentication* module. The *Authentication* component uses the *Manage UserData* interface provided by the accordingly named component within the *Database* subsystem to enable manipulation of user data. It, in turn, provides the *User Session* interface to the *Applications Manager* , *Portfolio Manager* and *Allocation Manager* components to ensure that every user can only access their own data.

The external interfaces *Manage Applications*, *Manage Portfolios* and *Manage Allocations* are each provided by the accordingly named components mentioned above. Furthermore, they interact with applications, portfolio and allocation data via interfaces provided within the *Database* subsystem, enabling CRUD (Create, Read, Update and Delete) operations. The only exception to this, are allocations, which cannot be updated. In terms of communication between the *Applications Manager*, *Portfolio Manager* and *Allocation Manager* there is also an interface providing applications data to the *Portfolio Manager*, and an interface providing portfolio data to the *Allocation Manager*.

To get an allocation for a portfolio, the *Allocation Manager* uses the *Calculate Allocation* interface provided by the *Cloud Portfolio Optimizer* component described in chapter 3. For creating allocations the *Cloud Portfolio Optimizer* also needs instances, which are provided by the *Instance Manager* component via interface. Said component also provides instance data to the external interface *View Instances*.

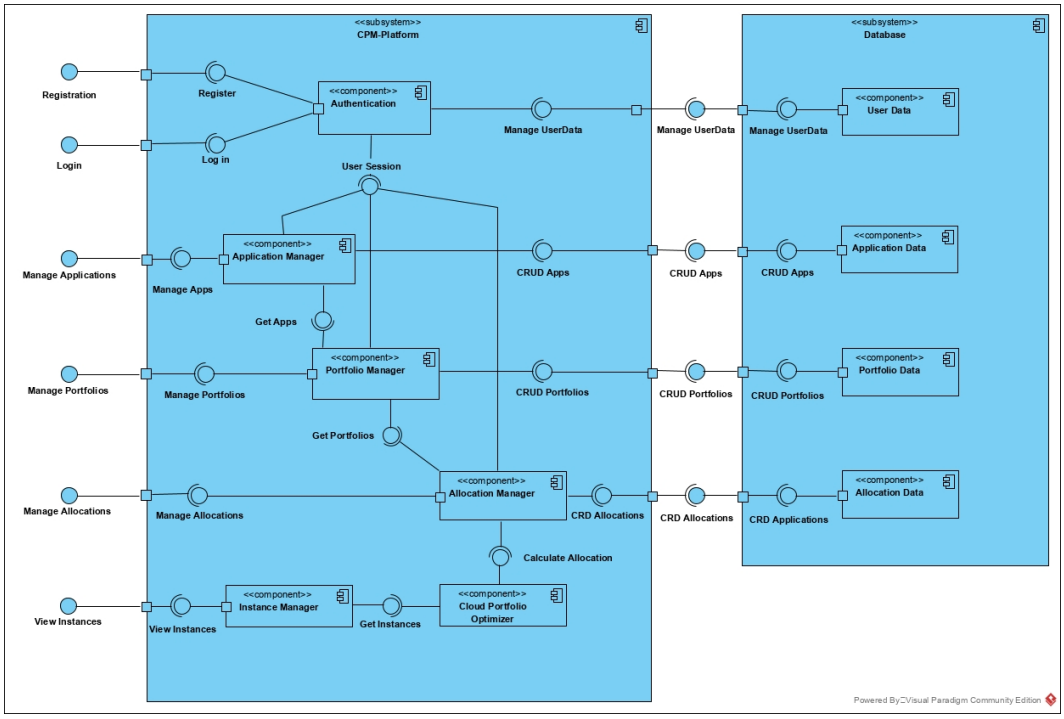


Figure 5.7.: Component diagram

5.2.5. Physical view - deployment diagram

For our final view, the physical view, we have decided upon using a deployment diagram. In order to be independent from the underlying hardware infrastructure and enable easy scalability, we have chosen to utilize Docker[2] to deploy our application. This way the app can also be effortlessly deployed on various CSP's platforms. Our program consists of three Docker containers, which in our diagram is depicted within one device called *Docker Host*. Our three containers are the *Frontend*, *Backend* and *Database*. The *Frontend* deploys our *App.vue* artifact and can be called via HTTP port 80, for example by a customer via their web browser. Our *Frontend* can also communicate with the *Backend* container, which deploys the *FastApi.main.py* artifact, via HTTP port 5000. For direct API calls, a customer's application can also use HTTP port 5000 to directly communicate with the application's backend. Finally, the *Database* container contains our *Mongo DB* database and can be reached via HTTP port 27017.

[2] `https://www.docker.com`

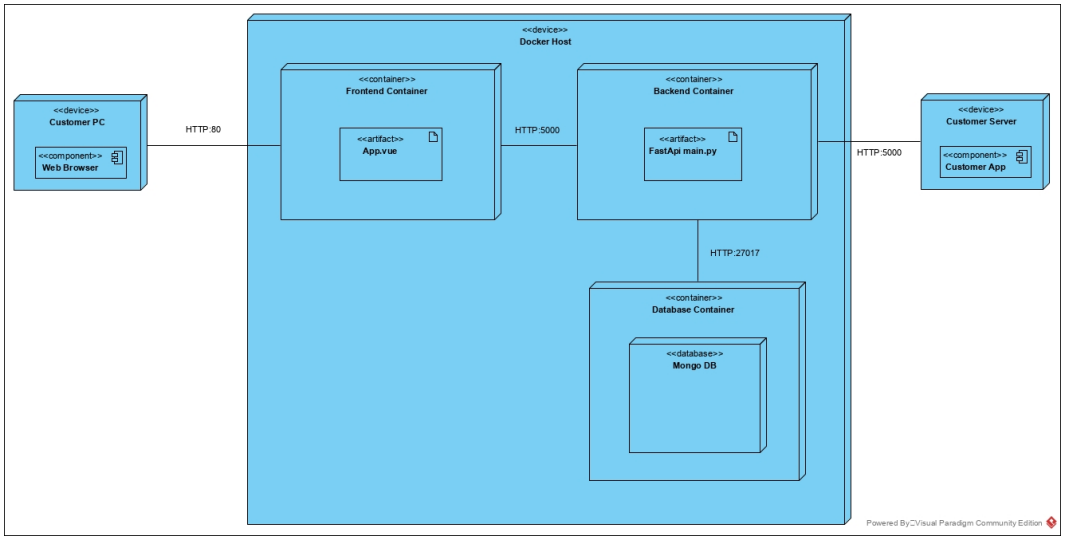


Figure 5.8.: Deployment diagram

5.3. Technology stack

Having discussed the requirements and architecture of our platform, this section will present the various technologies and dependencies that were finally chosen to implement the prototype. Said technology stack includes backend, frontend and database components.

First though, to tie together all the components of frontend, backend and database, we will deploy our platform with Docker [22]. Deploying the application in Docker containers allows us to bundle our software with its required libraries and configurations, offering the flexibility to deploy in various environments without any changes.

5.3.1. Backend

For programming the backend of the platform we have decided on using Python [29], a general-purpose programming language, which ranks amongst the most popular programming languages amongst professional developers worldwide [6]. While being not as fast as other languages such as C++, it is easy to learn and offers syntax that is straightforward to read by humans. These qualities help simplify the development and make the resulting implementation maintainable by many developers.

Another reason to choose Python is, that it is the language in which the optimization techniques described in chapter 3, have been implemented. This will help to integrate them into the platform with relative ease. Furthermore, due to its popularity, Python sports a wide range of third-party libraries and modules. One of these, pandas [28], is a prominent open-source library for data analysis and was central to the implementation of ERICH and GEORG. Therefore, it is also necessary for the Cloud Portfolio Manager for such uses as timestamps in a format the optimizer can process.

As web framework for building our REST APIs, we have chosen FastAPI [23], a relatively new high-performance framework with easy syntax and the ability to automatically generate OpenAPI (also known as Swagger) documentation for the created endpoints. This and the integrates Swagger UI allow for easy exploration, calling and testing of APIs from a web browser.

5.3.2. Frontend

For the development of the frontend we have chosen Vue.js [35], an open-source JavaScript framework for creating UI and single-page applications. Vue uses HTML, CSS and JavaScript for a component-based programming model. Each single-file component includes the logic (JavaScript), template (HTML), and styles (CSS), as can be seen in the example below Figure 5.9.

```vue
<script setup>
import { ref } from 'vue'
const count = ref(0)
</script>

<template>
  <button @click="count++">Count is: {{ count }}</button>
</template>

<style scoped>
button {
  font-weight: bold;
}
</style>
```

Figure 5.9.: Example of a simple Vue component

While JavaScript is the most commonly used programming language used by developers [6], Vue also supports the use of TypeScript [34], which builds upon JavaScript and offers some advantages in comparison with plain JavaScript. The main difference is that TypeScript, as implied by the name, uses a strict type system, while JavaScript dynamically infers types. This, in combination with better IDE (Integrated Development Environment) support, makes it easier to spot errors while coding and helps prevent type-related errors. As it is still basically JavaScript syntax-wise, it is easy to learn for the majority of developers, who have not worked with it yet. Due to these advantages, TypeScript will be the language of choice for programming the logic of our frontend.

To aid in building a visually appealing UI, the CSS framework Bootstrap 5 [18] was added to the dependency list. It offers a wide array of design templates for forms, buttons, navigation, and other interface components in a modular and customisable form. Finally, to tackle the requirement of a visual data representation, as described in subsection 5.1.1, we added the Chart.js [20] library to our project. It allows for uncomplicated creation of various types of charts, such as bar and line charts, with the extensive customisation and a focus on rendering performance playing into the one of our non-functional requirements.

5.3.3. Database

When considering which database to choose, the first decision to be made is whether to use a relational (SQL) or non-relational (NoSQL) data structure. While both have their advantages and disadvantages, offering dynamic schemas for easy future adaptations to how our data may be structured seems preferable. Additionally NoSQL databases are easier to vertically scale, which fits well to an idea of deploying our platform on Docker containers. As our database of choice, we have decided upon MongoDB [27], a popular NoSQL database with document orientation and optional schemas using syntax very similar to JSON. An example of a simple MongoDB document can be seen below in Figure 5.10. MongoDB also offers simple integration with many popular programming languages including Python, where we have decided to use Motor [30] as our MongoDB driver, which allows for easy and asynchronous interaction with databases.

```
{
    "_id": ObjectId("32521df3f4948bd2f54218"),
    "firstName": "John",
    "lastName": "King",
    "email": "john.king@abc.com",
    "salary": "33000",
    "DoB": new Date('Mar 24, 2011'),
    "skills": [ "Angular", "React", "MongoDB" ],
    "address": {
                "street":"Upper Street",
                "house":"No 1",
                "city":"New York",
                "country":"USA"
            }
}
```

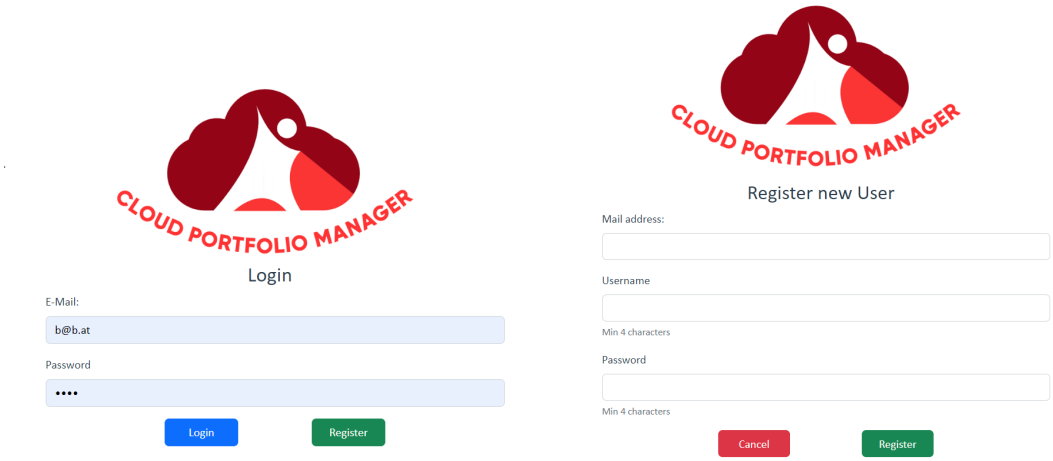
Figure 5.10.: Example of a MongoDB document
Source: https://www.tutorialsteacher.com/mongodb/documents

5.4. Prototype

This section will present the prototype of our Cloud Portfolio Manager created for this thesis. It will present an overview of the various pages and showcase the various functionalities of the application.

5.4.1. Login, registration and landing page

To start of the user is presented with the login page, as can be seen in Figure 5.11a, where the user can enter their e-mail address and password to access the website. In case of a wrong input, a short popup comes up. In case of a new customer not having an account yet, the *Register* button leads to the registration page as seen in Figure 5.11b, allowing for a new account being created by entering a valid e-mail address, a username and a password. Only the e-mail has to be unique to a new user and after successful registration the user is redirected back to the login page.



(a) Login Page (b) Registration page

Figure 5.11.: Screenshots login and registration pages

Logging in into the website with the correct credentials brings the customer to the landing page in Figure 5.12 displaying the platforms logo and listing its creators. This page, as well as all others besides the login and registration, also feature a navbar for easy navigation between the various pages. The logout button on the far right of the navbar logs out the current user and returns them back to the login page.

Figure 5.12.: Landing page

5.4.2. Instances page

This view can be navigated to through the *Providers and instances* tab in the navbar and contains information about available instances from the various providers. For our prototype we chose a range of instances from the four biggest CSPs, meaning AWS, Google Cloud, Microsoft Azure and Alibaba. They all give good examples of what is available on the market. For future development, this could be adapted to load instances provided by CSP APIs, which will be further discussed in the next section 6.1. The list on this page gives an overview of each instance's main attributes consisting of provider, name, market space, capacity and price. Furthermore there is a filter function allowing a user to simplify their search for instances. For example, as shown in Figure 5.13, the search does not include Google Cloud instances, but instances from all market spaces, with a capacity of 5 or higher and a price of up to 1000$.

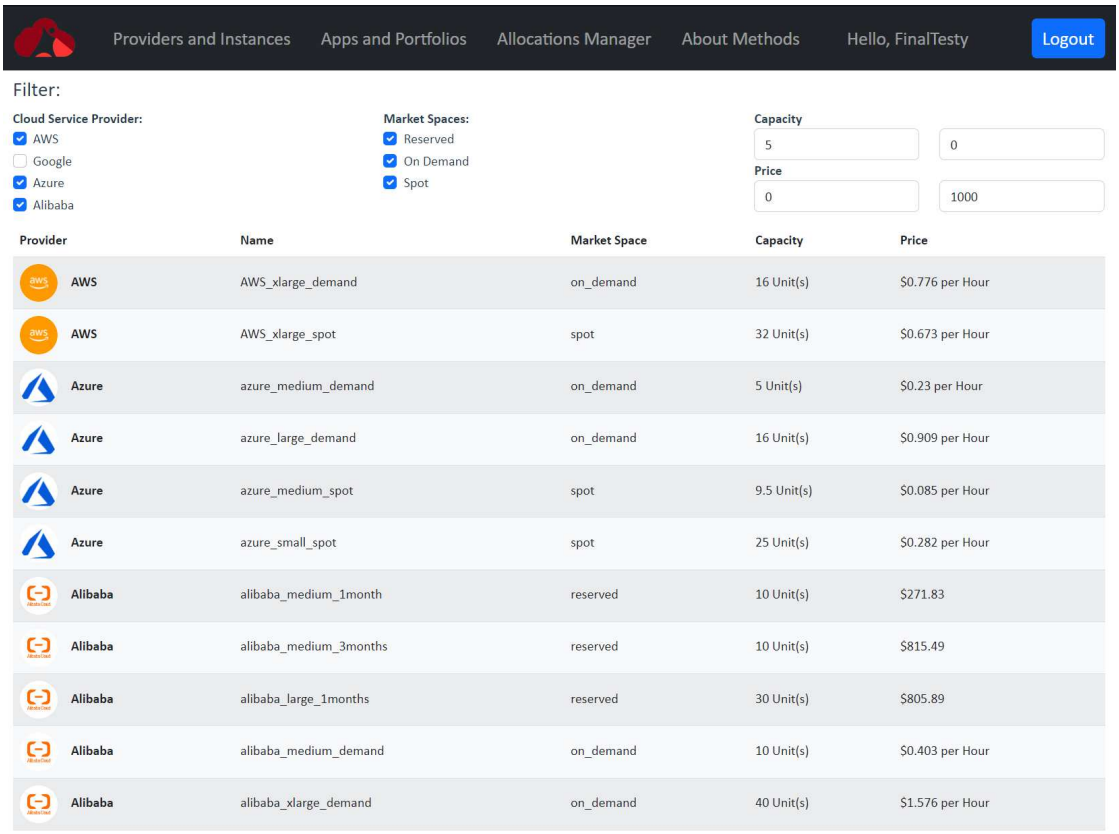


Figure 5.13.: Instances page

5.4.3. Apps and portfolios page

Next up, the *Apps and portfolios* page enables the user the management of two of the main components of the Cloud Portfolio Platform. As can be seen in Figure 5.14 below,

the left side lists the user's applications and details like mean resource demand, demand variance, preemtibility, as well as starting and finishing time. On the right side of the page is a list of the user's portfolio including details, like which providers should be considered for any possible allocation, a minimum quality of service, the number of apps in the portfolio and a list of which applications exactly the portfolio consists of. Finally, it also states the portfolio's version, which is incremented every time a portfolio or one of the applications it consists of are changed. It is used to keep track of which portfolio version a certain allocation has been calculated for. This enables the user to spot if any of their allocations are outdated, or if any specifications or the makeup of the underlying portfolio have changed.

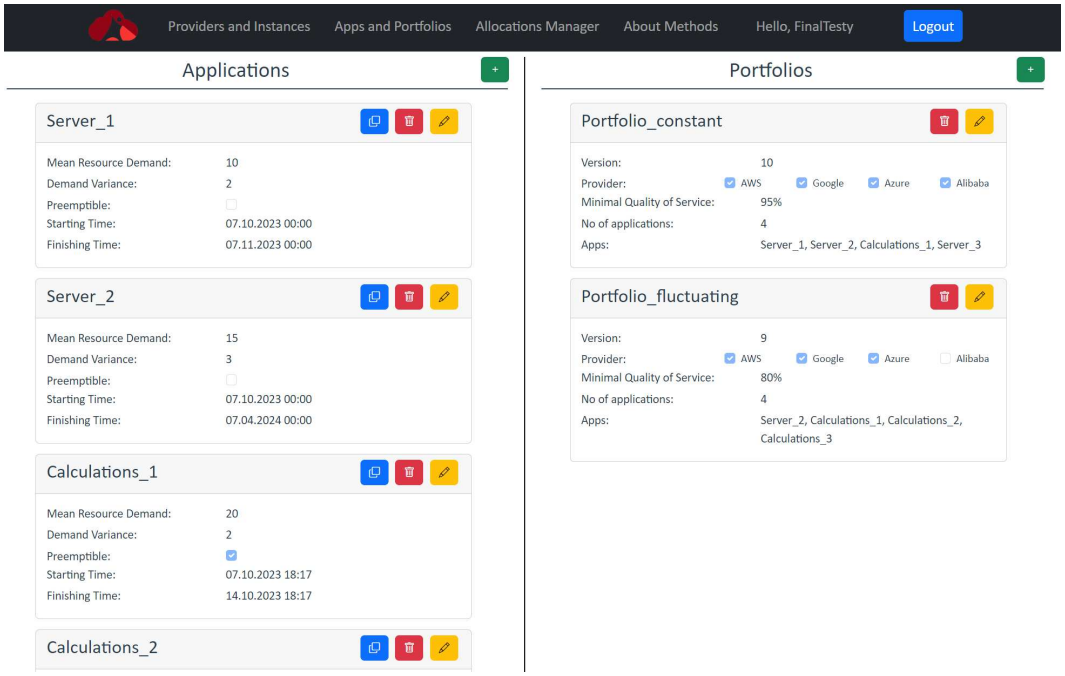
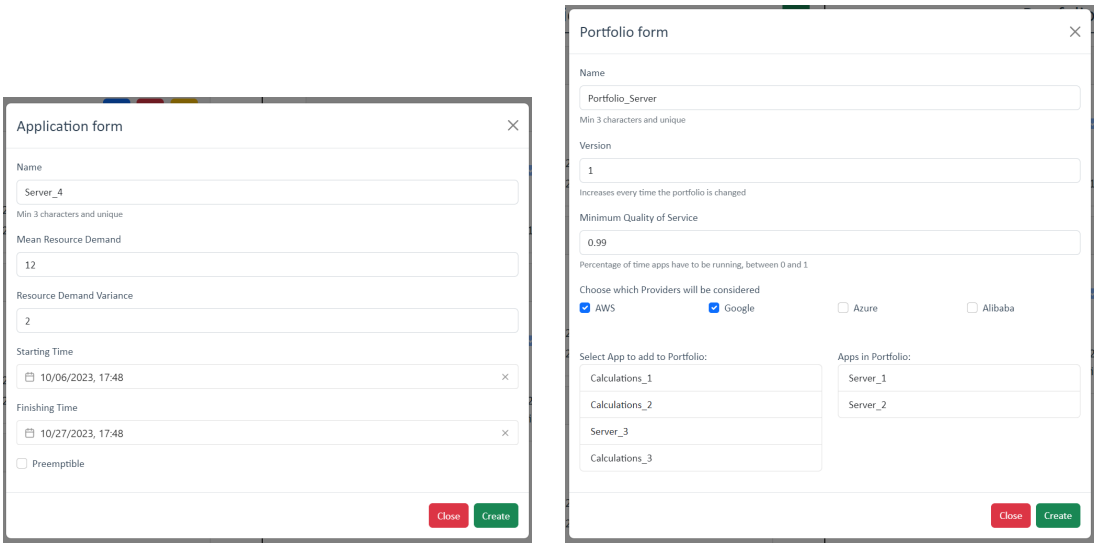


Figure 5.14.: Apps and portfolios page overview

To create a new application or portfolio there are two green buttons depicting a plus sign on each side of the page. These open the respective application and portfolio forms, as can be seen below in Figure 5.15a and Figure 5.15b. To create an application the user has to fill out the application form including a unique name, mean resource demand, demand variance, a checkbox for preemtibility, and finally the starting and finishing time chosen via a date-time picker. Should the user wish to create a portfolio, the portfolio form requires a unique name and a minimum quality of service, which gives a percentage of time the apps in the portfolio are required to run. The portfolio version, described previously, cannot be changed manually by the user. The portfolio form also requires the user to choose at least one CSP to be considered for allocations, as well as choosing which apps should make up the portfolio. To ensure suitable inputs for both forms, they also

feature various checks, giving instant feedback to invalid inputs, such as an application's finishing time being before its starting time.



(a) Application Form (b) Portfolio Form

Figure 5.15.: Screenshots of application and portfolio forms

Each application and portfolio can also be updated by clicking the yellow button with a pen icon displayed with every application and portfolio. This will open up the respective form already filled out by the app's or portfolio's data. For ease of creating several applications with similar characteristics without having to fill out the entire form every time again, applications also feature a blue copy button, which will open the application form filled out with the characteristics of the chosen application and the suffix "_copy" added to its name. Furthermore, clicking the red button displaying a trashcan icon will delete the application or portfolio of choice. Deleting an application will also remove it from any portfolios it may be part of and increment the portfolios version. To give feedback to operations performed on this page, creating, updating and deleting applications and portfolios will result in a short popup denoting a successful operation, or, should there be any errors occurring, give the user notice of there having been an error.

5.4.4. Allocations page

This tab focuses on the primary feature of the Cloud Portfolio Manager: creating allocations for cloud portfolios. At the top of the page, the user has a dropdown menu, which lists their created portfolios. Choosing a portfolio shows its details on the side and all already existing allocations for this portfolio below. Every allocation has an overview stating which algorithm was used, which portfolio version it was made for, its total costs and the mean overall utilization achieved with this allocation. As allocations can take a while to be calculated, especially in case of using the GEORG algorithm, there is also a

field stating if the allocation is already completed. Below the general stats are two fields, which can be extended by clicking on them. The first contains a full list of all instances used for this allocation, as well as some statistics like capacity, price and beginning and ending of the instance's run time. The second field contains more detailed statistics about the allocation, like separate statistics for reserved, on-demand and spot instances. Each allocation can also be deleted by clicking the red button with a trashcan icon in the header section of each entry. On the right side of the allocations list, this page also features some data visualisation with graphs comparing the price, utilization overall and utilization split by instance type for each allocation as bar charts.

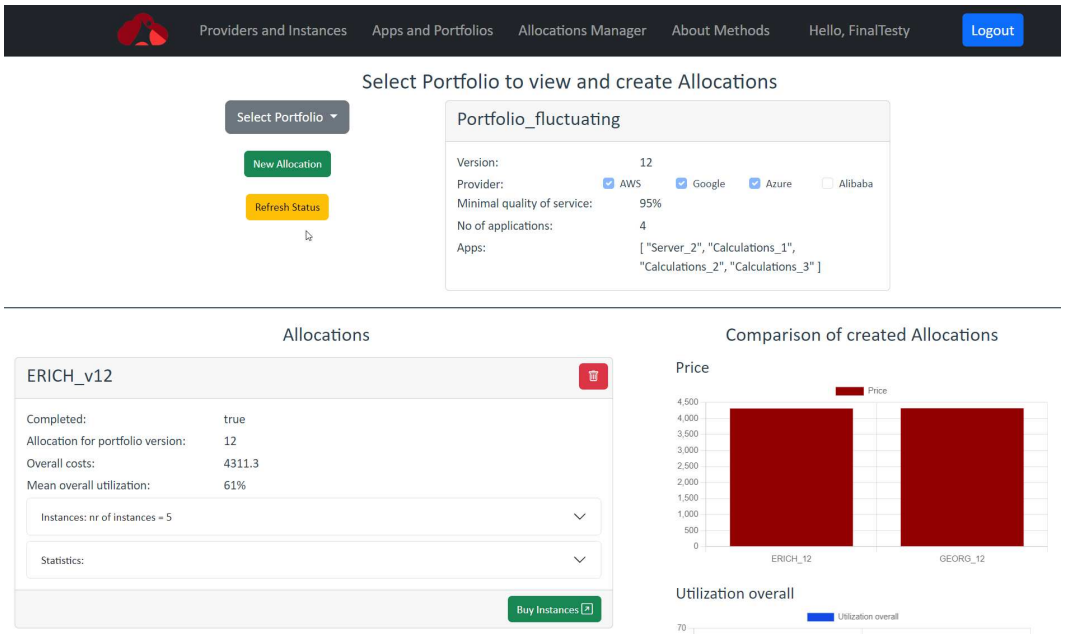
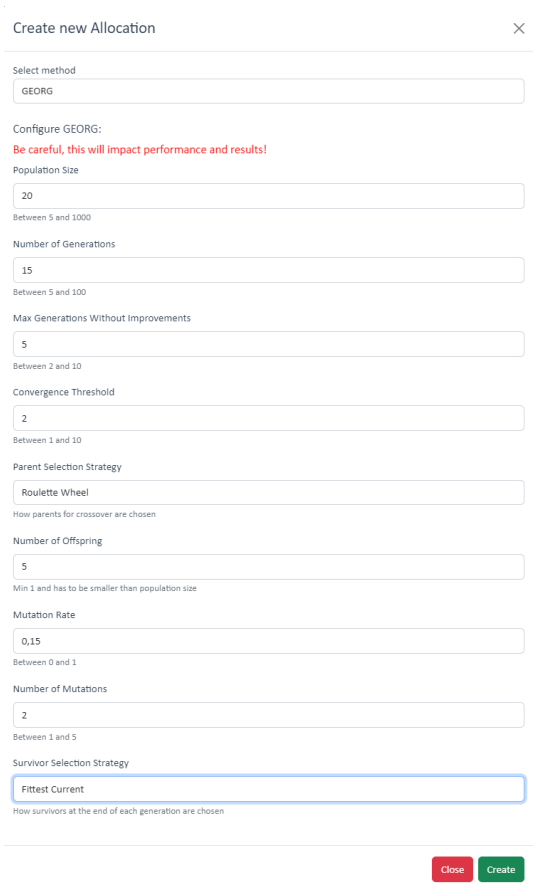
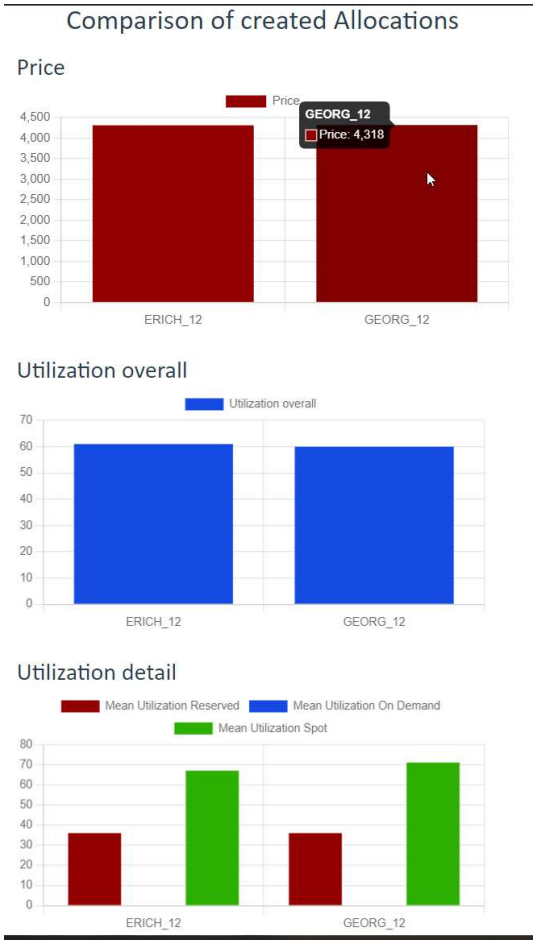


Figure 5.16.: Allocations page overview

To create a new allocation, the button "New Allocation" opens a form where the user can choose which algorithm should be used. In case of using ERICH that is all that is required to do, as there is no parameterization possible. Otherwise, when selecting GEORG, a variety of options for configuring the genetic algorithm is available. Examples of this would be the size of the population, number of generations and the mutation rate. For each there is a default value, should the user not want to look into this topic too much. Otherwise this enables experimentation with this algorithm, which can lead to varying results. The user should be aware though, that this will also impact performance and, for example, a very high population size is more resource intensive for the system the platform runs on. As the calculation of the allocations is run asynchronously, there is also a "Refresh" button on this page, which reloads the allocation data from the backend.



(a) Form for creating a new GEORG allocation



(b) Graphs for comparison between a GEORG and ERICH allocation for an example portfolio

Figure 5.17.: Details on allocation page

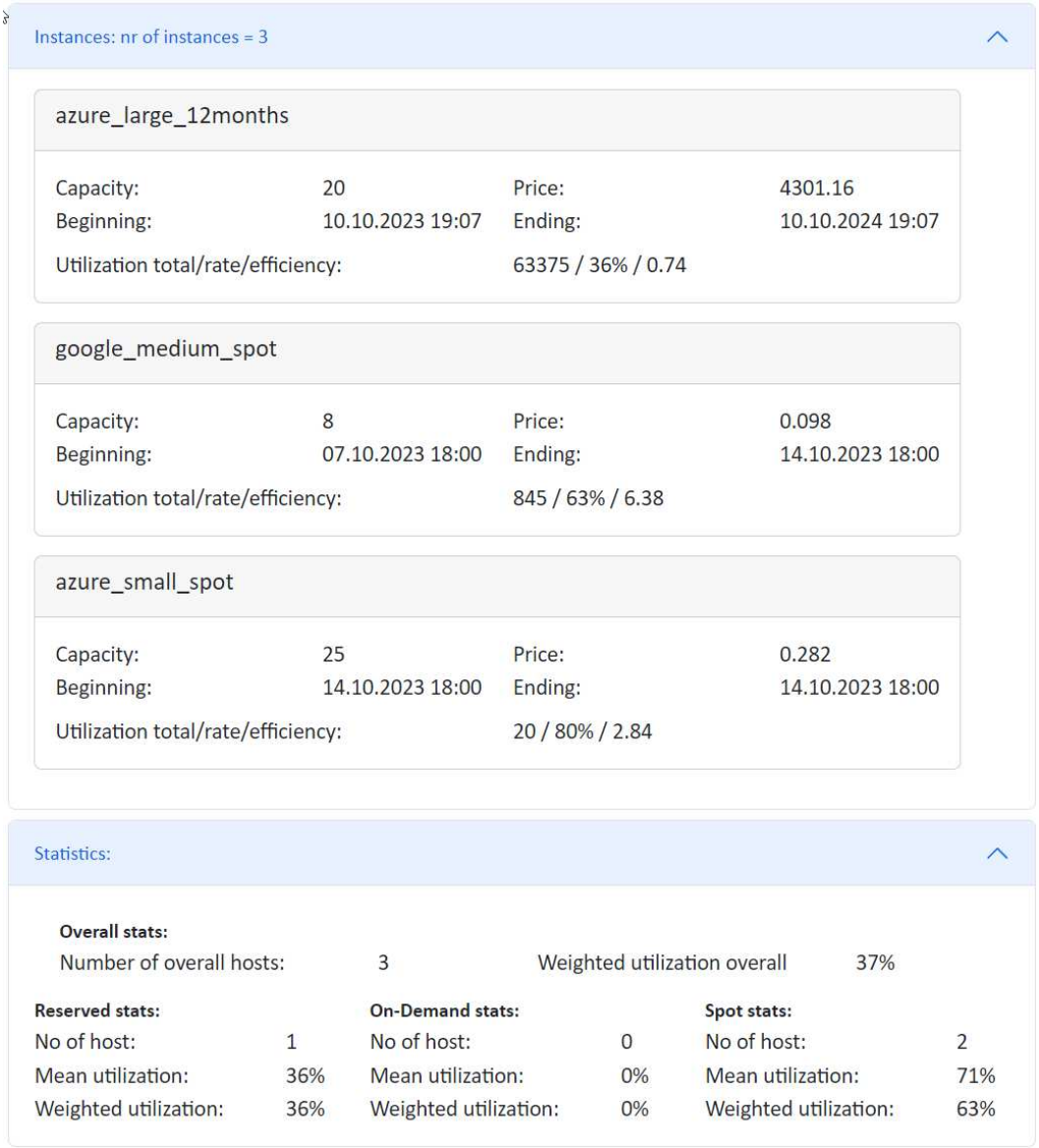


Figure 5.18.: Details for a allocation with statistics and instances

5.4.5. About methods page

This final tab in the navbar gives the customer an overview of the theory behind our optimization methods and the general problem. It consists of three pages: The first one very briefly gives an overview why cloud portfolio management is a topic of note and some basics about instance types and preemptibility. The second and third page both focus on our algorithms, giving a cursorily explanation of how they work. While ERICH is explained outright, the page concerning GEORG starts with giving an overview

of what a genetic algorithm is, before giving some specifics about our implementation of one. Generally, this page aims to give users some insight into the inner working of the platform, assuming they possess some domain knowledge about the cloud and optimization approaches. While it can be assumed that it is not of much use to users with no background in these fields, they are not the targeted demographic of the platform.

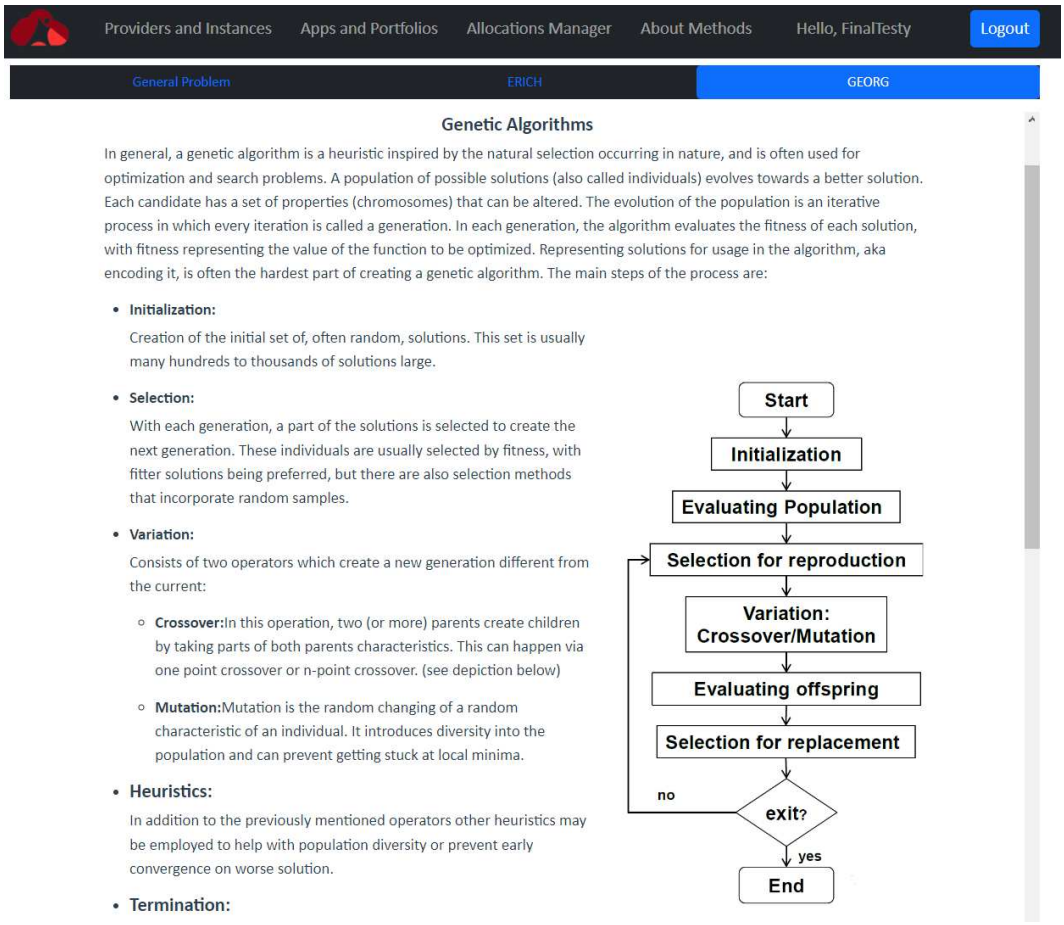


Figure 5.19.: Tab explaining the genetic algorithms and GEORG

5.5. User experience evaluation

Having presented our prototype, this section will provide a brief evaluation of the user experience (UX). It is crucial for any platform targeting success in the public market to possess a convincing user interface. To this end, we will discuss the usability heuristics developed by Nielsen et al. and, based upon this same heuristics, present a brief survey on the quality of our user interface [68, 69].

5.5.1. Usability heuristics

Heuristics are an approach used to find not perfect, but adequate solutions to a problem, such as the optimization approaches we developed in chapter 3. This methodology can also be applied to evaluate the quality of user interfaces. One of the most commonly used and widely known usability heuristics are those developed by Nielsen in the 1990s. He proposed 10 general principles for designing user interfaces, which are the following [36, 68, 69]:

- **Visibility of system status**: This heuristic describes the ability of design to always communicate to the user about what is going on within the system in an easily understandable and immediate manner. A user has to be able to know what the previous interactions resulted in and furthermore understand which following steps are possible.
- **Match between the system and the real world**: This refers to a design, that communicates by using words and concepts that a user recognised, instead of internal terms. A design that follows this heuristic, enables intuitive use of the interface without any need to learn any new words or concepts.
- **User control and freedom**: As users like to try various actions and mistakenly perform others, there should always be a clear way to go back a step and cancel any action. This prevents users from getting stuck and gives them the confidence to freely try any action within a system.
- **Consistency and standards**: To offer a good user experience, an interface should be consistent. This refers not only to consistence within the product you offer, but also in regards to similar products a customer could be accustomed to and have developed expectations from.
- **Error prevention**: This refers to designing an interface in a way that helps to prevent the occurrence of errors. Errors can be categorised into slips and mistakes. Slips are unconscious errors caused by being inattentive and can be combated by setting helpful constraints and defaults. Mistakes on the other hand happen consciously and are the result of the design not properly communicating the model to the user. They can be alleviated by enabling the undoing errors and good warnings.
- **Recognition rather than recall**: As users have a limited short-term memory, a good design does not rely on recall of elements, actions and options but encourage recognition. Additional information needed, should be easily accessible, if so required.
- **Flexibility and efficiency of use**: This heuristic refers to providing experienced users with possibilities to speed up interactions which inexperienced users may not need. For example keyboard shortcuts. These allow for flexible processes that can be executed in various ways.

- **Aesthetic and minimalist design**: The design of interface elements should prioritize essential information required for their functionality. Additional irrelevant or rarely needed information competes with relevant elements for visibility.
- **Help users recognize, diagnose, and recover from errors**: If errors occur, the system should inform the user using plain language, providing an accurate description and, when possible, a suggested solution. Technical terms, such as error codes and unusual visual design for error messages, should be avoided.
- **Help and documentation**: While in the best case a system does not need any further explanation, it may be necessary to offer documentation to enable the completion of some tasks. Said documentation should be concise, easily searchable and consist of concrete steps to be carried out.

5.5.2. Methodology

Based upon these heuristics, we will conduct an evaluation of the Cloud Portfolio Optimizer frontend design. This will consist of five testers going through a set of tasks on our platform and filling out a questionnaire afterwards. While the number of testers may seem rather low, Nielsen et al. mention in their work that in contrast to one tester often missing a lot of problems, three to five aggregated evaluations offer good results [69].

For the evaluation process each tester is expected to complete the following list of tasks:

- Create a new account and log into it.
- Navigate to the 'about methods' page and skim over the description there.
- Go to the instances page, have a look at the instances displayed and use the filters available.
- Navigate to the applications and portfolios page, create four new applications, update one of them, copy another and finally delete one.
- Create two portfolios, update one and delete one.
- Finally, proceed to the allocations page, and create one allocation for each optimization approach.
- Log out from the platform which returns the user to the sign-in page.

Having completed the tasks above, the testers will be asked to fill out a questionnaire based upon the Nielsen´s heuristics. It was created using Google Forms[3]), and asked the user to rate the platform in regards to how well it adheres to each of the design heuristics on a scale from 1 to 5, with 5 representing the best possible adherence. Furthermore, the testers were questioned on whether they found any issues or had any suggestions in

[3] `https://docs.google.com/forms/u/0`

regards to each heuristic. To gauge the testers' expertise in regards to IT in general and the topic of cloud markets, the survey also contains a self-assessment of these topics. The questionnaire can be found in the appendix in section A.1.

5.5.3. Results

The results of the questionnaire gave useful feedback to the design of the user interface, and additionally pointed out a couple of errors which were overlooked in development. Five testers completed the survey as described in subsection 5.5.2. Regarding IT-related experience, two testers reported no background in IT, one mentioned having educational knowledge, and two testers stated having work-related IT experience. The testers' knowledge about the cloud market was rather limited, with three testers stating their knowledge to be cursory and two having no knowledge about the topic.

Overall the platform was perceived as adhering to most heuristics pretty well, with some achieving better results than others. The best rated heuristics were those of "aesthetic and minimalist design", as well as "user control and freedom" achieving an average of 4.6 and 4.4 points respectively. The heuristics of "consistency and standards" and "recognition and standards" also scored well averaging 4.0 points. All other heuristics got an average of 3.4 or 3.8, with the notable exception of "error prevention" only scoring 3.0 on average. The complete list of average scoring can be seen in Table 5.1.

Visibility of system Status	3.8
Match between the system and the real world	3.4
User control and freedom	4.4
Consistency and standards	4.0
Error prevention	3.0
Recognition rather than recall	4.0
Flexibility and efficiency of use	3.8
Aesthetic and minimalist design	4.6
Help users recognize, diagnose and recover from errors	3.4
Help and documentation	3.4

Table 5.1.: Table of average assessment for each heuristic

These results point towards the platform having an aesthetic, and mostly easy to use design, with some room for improvement in terms of error prevention and handling, as well as the documentation and help. All testers, except for one, who provided no useful feedback by rating several heuristics poorly without offering any comments, also reported issues they encountered and provided suggestions for improvement. These can be summed up as follows:

- While many input fields already have some form of input check, several testers found several input possibilities that result in applications with invalid parameters.

- One tester found a timeout error, which is caused by the authentication token having too low of a timeout window.
- The copy function for applications was also requested for portfolios.
- Several testers observed that the pop-up notifications providing feedback for incorrect or missing inputs and errors are displayed for too short of a duration.
- While the documentation on the problem description and optimization methods are useful, a user short guide on how the platform is meant to be used, would be beneficial.
- Testers also suggested some small improvements on how information is displayed, such as adding currency symbols, to increase ease of recognition.

To sum up, the testers rated the platform positively in regards to the majority of heuristics and provided useful feedback for improvement. The suggested feedback concerning minor changes, has already been applied, while others inform future ways of improving the platform, such as adding a user guide.

6. Maintenance and further development

In this chapter it will be discuss how the prototype that has been created, can be maintained and explore what would have to be done to turn it from a working prototype into a full functional platform that could offer its services to the public market.

6.1. Maintenance

First off, we will discuss how the implementation is structured right now and what implications this has upon both maintenance and further developments the platform could undergo.

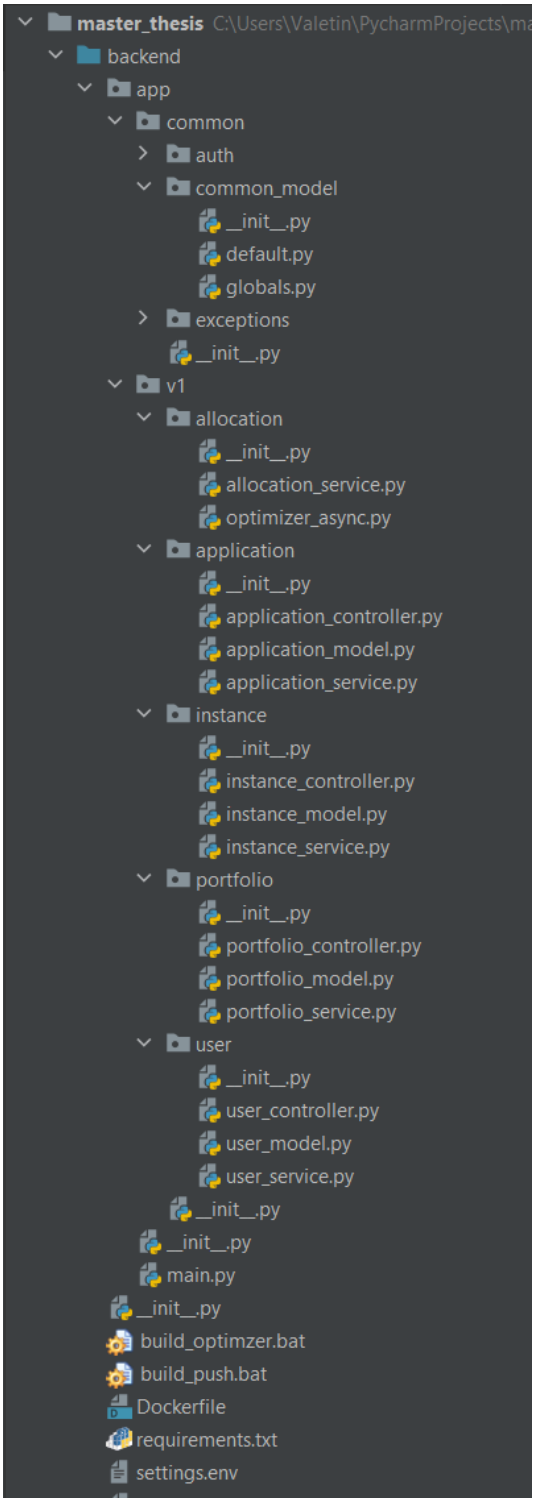
The frontend of the platform was developed with the Vue.js framework, which results in a component based structure (see Figure 6.1b), with the files *main.ts* and *App.vue* being the entry points. As mentioned in section 5.3, we have chosen TypeScript as our programming language for the frontend logic. Each component contains a HTML template, Typescript logic and CSS styles. Should there be the need to remove or add pages to the website, they are all listed within the router folder in the *index.ts* file, which also defines a pages path, which component they use and whether the user needs to be authenticated to access the page in question. Any data models used for data manipulation on the frontend side are defined in the *interfaces.ts* file located in the accordingly named folder. Finally, any variables needed in more than one component, like the userId and username, are kept in the store defined in the *store.ts* file.

On the backend side of things, the application is set up with the FastApi framework with the *main.py* file as entry point. Within the *common* package functionalities are needed across the application, such as the packages for authentication, exceptions and global variables containing the logger, database driver and threadpool used for the allocation calculations. The rest of the application is separated into packages pertaining to the various data models central to its functionalities, which are *user*, *application*, *instance*, *portfolio* and *allocation*. Each package, except the *allocation* package, is set up in the same way, as can be seen in Figure 6.1a: A $*$_controller.py file containing the APIs pertaining to the model in question, for example the CRUD operations for applications, a $*$_service.py file containing the logic behind the various API functions and a $*$_model.py file containing the schemes used to operate with the model in question. Finally, the allocation package contains a service for creating allocations and an underlying optimizer class for asynchronously running the optimizer functions provided by the program discussed in chapter 3.

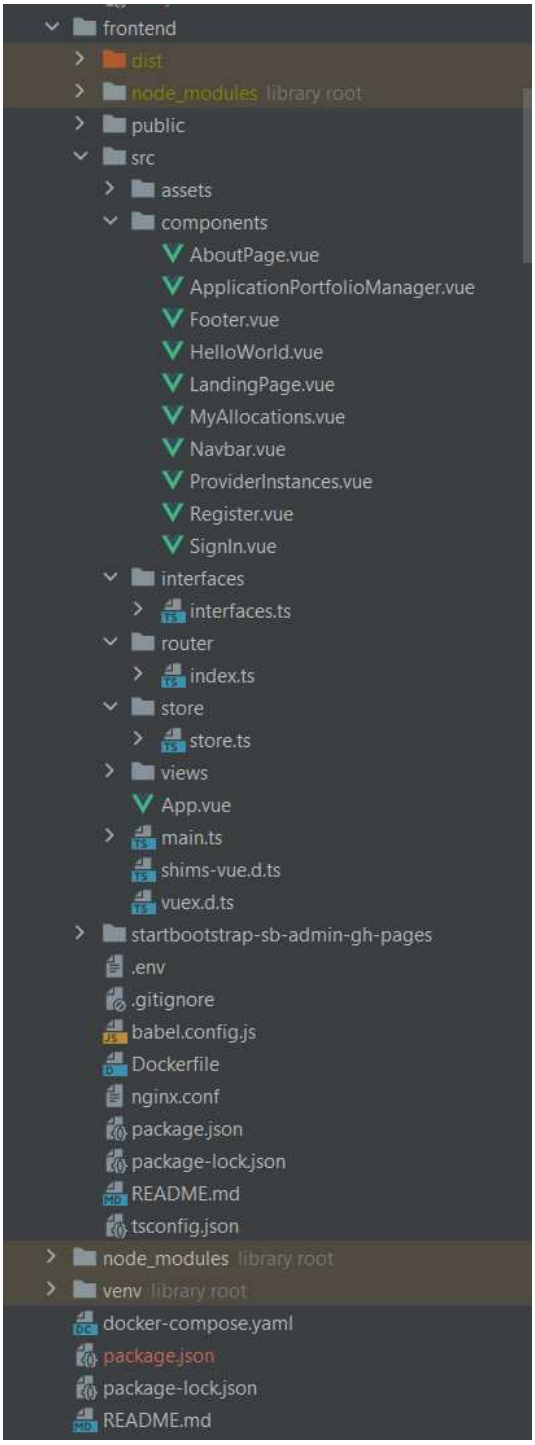
The final major component of the platform is the MongoDB database. Due to it being a NoSQL database, should there be the need to change, add or remove the models of

any of the entities saved in the database, said changes would need to be implemented in the platform's backend. This is because the entities are saved in JSON format, which is defined in the *_model.py files of each data models package.

If there is a need to modify the deployment configurations of the frontend and backend containers, their configurations are defined in the dockerfiles located in the root folders of both the frontend and backend. How the three containers making up the platform interact, which ports they use, and so on, is defined in the *docker-compose.yaml* file within the root folder of the project.



(a) File structure of backend



(b) File structure of the frontend

Figure 6.1.: Screenshots of how the project folders are set up

6.2. Creating a fully functional commercial platform

Within this subsection we will discuss the four main changes necessary to the platforms prototype, which would enable its use as a proper commercial platform offering its services on the public market:

- **Split capacity into CPU and RAM**: The first adaptation is a proper split of needed capacity for applications into vCPUs and RAM requirements. While for the purpose of creating a prototype the unit of capacity, where 1 corresponds roughly to 1vCPU and 2GB of RAM, was sufficient, it is not directly applicable to real life applications. Not only are the needs of some applications skewed more towards being memory or CPU intensive, but the capabilities of real life instances offered by CSPs are also measured in the units vCPU and RAM. To accommodate these changes, rather few adaptations to the platform itself are needed, but a major overhaul of the optimizer functions would be necessary, as they currently optimize for one parameter only.

- **Incorporation of CSP instances**: Next up, and dependent on the previous change, is the direct incorporation of instances offered by CSPs. Currently, the instances used by the prototype are selected mocked examples of instances which can be found on the CSPs currently incorporated into the prototype, with their real world capacity converted to our proprietary unit mentioned above, and their actual pricing in US dollars. Having implemented the split of capacity into vCPU and RAM, would enable quite an uncomplicated switch to using instances as offered by all major CSPs. The various CSPs provide APIs that enable querying available instances and their specifications. AWS, for example, allows for a range of queries through direct API calls, not only to query the instances available and their properties, but also a wide range of other operations, such as buying, starting and stopping them[1].

  Therefore, the basic adaptation needed would be implementing the platform to query current instances from CSPs and use them for allocation creation. Another functionality that could be added on top of that would be to allow the customer to use our platform to directly buy instances suggested by allocations via the various CSP APIs. This would require the user to add their account data from the various CSPs to our platform and will result in these accounts being charged for the purchased instances, which hints at the necessity of the next point.

- **Improved security**: Currently the platform uses a simple e-mail + password combination for login and authentication. While the password is stored encrypted with HS256 and the token used for authentication only has a limited validity, the current security measures are not suitable to a public platform. Some basics that would need to be implemented are a password policy for a certain strength of password and validation of the e-mail address used for an account, which currently only checks for the syntax of the e-mail address. Furthermore, a two-factor authentication method

[1] `https://docs.aws.amazon.com/AWSEC2/latest/APIReference/Welcome.html`

via mail or phone for certain sensitive operations like buying instances, could be sensible.

- **User guide**: One feedback received in the UX evaluation in section 5.5 points towards the need for a proper user guide for using the Cloud Portfolio Manager. This could be implemented as a written documentation, with the possible addition of video guides on how to use the platform.
- **Inclusion of payment provider support**: The final additional feature, crucial to turn our prototype into a commercially viable platform, is the incorporation of payment provider support. Without any way to charge a customer for the service provided, the platform can not operate properly. The most convenient way to achieve this, would be the implementing support for widely used online payment system like PayPal, Google Pay and Apple Pay. For other payment methods, like credit card or bank transactions, usage of a third party payment provider like Stripe[2] would be a feasible way to facilitate payment from various sources, without needing to invest much time nor the need for know-how. While services like this charge a fee for payments, implementing proprietary support for services like VISA is not practical for a small business in the authors' eyes.

6.3. Possible further improvements

Unlike the points discussed in the previous subsection, the following changes to the platform are not necessarily required to deploy the platform publicly, but would nonetheless increase its viability on the market:

- **UI improvements**: While the current user interface of the platform is suitable to its functionality and status as a prototype, an overhaul of the UI would increase its marketability as a commercial platform. To that extend, this could either entail developing a custom website design with the help of existing frameworks like Bootstrap or use a website style template, as can be procured online. Another UI related enhancement to the application, would be to extend the visualisation capabilities of the allocation component of the frontend.
- **Additional optimization methods**: Currently our application offers the user two optimization methods, which use different approaches to create allocations for a portfolio. One way to extend a platforms' offer would be the implementation of one, or several, additional optimization methods. While feasible, it has to be taken into account that this new method would not only require significant effort, but would also have to deliver results, at least as good as those GEORG and ERICH can produce, while keeping the run-time at a comparable length.
- **Inclusion of more CSPs**: With the implementation of support for the currently mocked instances from the four biggest providers Amazon AWS, Microsoft Azure,

[2] `https://stripe.com/at`

Google Cloud and Alibaba Cloud, our platform would cover over 70% of the current cloud market volume [32]. Nonetheless, another possible way to extend the platforms' offer, would be to add support for further CSPs offering IaaS services, like IBM Cloud and Oracle Cloud. This could not only extend the platforms reach in terms of customers, but also result in better allocations depending on the added providers instance offers.

- **Additional cloud portfolio management functionalities**: Furthermore, outside the scope of the prototype we developed in this thesis, the platform could also be extended by offering additional services and functionalities related to cloud portfolio management. One example for this would be the creation of a monitoring tool that allows the user to keep track of their entire cloud portfolio spanning over several providers. While every major CSP offers monitoring tools on their platform, they obviously do not support cross platform integration.

7. Conclusion and future work

In this final chapter, a brief summary of this thesis will be given, as well as presenting a discussion on the limitations of this thesis, and possible future work that could be done based upon it.

This thesis focuses on the topic of cloud portfolio management platforms and related business models. Cloud portfolio management primarily concerns itself with finding cost efficient allocations for cloud resources. Therefore, the work starts with discussing the core concepts, beginning with cloud computing and the cloud market in general, as well as business models and cloud intermediaries.

To find an efficient allocation for cloud resources, cooperative research was conducted. For this, we discuss our research into the topic of portfolio optimization, combining concepts already proven useful with new ideas to approach the problem at hand. We present the two optimization strategies chosen and the algorithms developed. One works similarly to a greedy first-fit bin packing technique, while the other employs the framework of a genetic algorithm to approach the problem. The implementation of these algorithms is tested with the help of six test cases using artificial data. The results of the test show that the greedy algorithm achieves slightly better performance, both in terms of tighter packing and speed of execution.

Having discussed the necessary concepts around the thesis is built, we propose our business model for a Cloud Portfolio Manager. This starts with an analysis of various use case scenarios for the platform. We come to the conclusion, that while businesses like those offering SaaS products and incorporating machine learning technology would be a great fit for our value proposition, others, like those facing security requirements not compatible to cloud services, are not. We then present our business model built upon the business model canvas framework by Osterwalder and Pigneur. In this, we describe the various building blocks of our business model including customer segments, value proposition and revenue streams. Next, we classify our business model with the systems of both Timmers, where we place it amongst common internet business models close information brokers with a degree of innovation, but low functional integration, and Osterwalder and Pigneurs patterns. To give a frame of reference, the thesis also looks into similar existing businesses and their models of operation.

With the business model in place, we then implemented a prototype of the Cloud Portfolio Manager, incorporating the two optimization algorithms developed. We go about this in a classical application development process by first doing a requirements analysis, laying down both functional requirements concerning persistence, UI, data visualisation and API support, as well as non-functional requirements like performance, adaptability and scalability. We then define the applications architecture using a 4+1 view containing various UML diagrams such as class and activity diagrams. After discussing why we

have chosen our technology stack, we present the prototype and its functionalities. These include the creation of an account on the platform, mapping the customer's applications and portfolios in our data model and most importantly, the creation of allocations for said portfolios.

To enable maintenance, bug fixing and further development upon the prototype, we also discuss its implementation in some technical detail, as well as how and where it could be expanded code-wise. Furthermore, since the developed prototype cannot be deployed as a public platform to serve our proposed business model as-is, we also discuss the steps that would be necessary to do so. This is split between those points that need to be done, such as the inclusion of payment services and what would be advantageous but not strictly necessary like UI improvements.

As the cloud computing market has been on a meteoric rise over the past years and is still expanding, the topic of cloud portfolio management will most likely keep or expand its relevance in the coming years. As more big IT firms try to break into the market and compete with the three biggest providers, offering customers an easier access to multi-cloud strategies and comparison between the various providers will be a topic of great interest.

With the developing a prototype of a cloud portfolio management platform, and laying out how to expand upon it, chapter 6 already discussed on possible future work based upon this thesis. This would see the prototype expanded to a fully functional public platform operating with our business model or a modification of it. Furthermore, the platform's offered services could be expanded into various monitoring functionalities and direct control of portfolios via the platform.

Further work could also concern itself with improving the existing optimization algorithms and creating new ones. In addition to tighter packing, these could, for example, elaborate on the resource constraints considered, such as network capabilities and storage. Finally, further research could build upon the business model presented in this thesis. Considering that it is likely that new cloud intermediaries will emerge in the future, analysing their business model and comparing it to our model could be of interest.

Bibliography

[1] Amazon ec2 – preise. https://aws.amazon.com/de/ec2/pricing/. Accessed: 2023-03-02.

[2] Annual revenue of amazon web services (aws) from 2013 to 2022. https://www.statista.com/statistics/233725/development-of-amazon-web-services-revenue/. Accessed: 2023-10-09.

[3] Microsoft – preise. https://azure.microsoft.com/de-de/pricing/details/virtual-machines/linux/. Accessed: 2020-04-20.

[4] Worldwide public cloud services revenues surpass 500 billion dollar in 2022, growing 22.9 percent year over year, according to idc tracker. https://www.idc.com/getdoc.jsp?containerId=prUS51009523#:~:text=NEEDHAM%2C%20Mass.%2C%20July%206,increase%20of%2022.9%25%20over%202021. Accessed: 2023-10-09.

[5] Solutions, 2017.

[6] Programming languages popularity. https://insights.stackoverflow.com/survey/2021#most-popular-technologies-language-prof, 2021. Accessed: 09.08.2023.

[7] Aws compute. https://aws.amazon.com/savingsplans/compute-pricing/, 2023. Accessed: 20.06.2023.

[8] Aws on-demand. https://aws.amazon.com/ec2/pricing/on-demand/, 2023. Accessed: 20.06.2023.

[9] Aws pricing. https://aws.amazon.com/ec2/pricing, 2023. Accessed: 20.06.2023.

[10] Aws reserved. https://aws.amazon.com/ec2/pricing/reserved-instances/pricing, 2023. Accessed: 22.06.2023.

[11] Aws spot. https://aws.amazon.com/ec2/spot/pricing/, 2023. Accessed: 23.06.2023.

[12] Aws spot history. https://eu-central-1.console.aws.amazon.com/ec2/home?region=eu-central-1#SpotInstances:, 2023. Accessed: 23.06.2023.

[13] Aws spot interruptions. https://aws.amazon.com/ec2/spot/instance-advisor/#:~:text=Frequency%20of%20interruption%20represents%20the,aws-spot-labs)., 2023. Accessed: 23.06.2023.

[14] Azure pricing. https://spot.io/resources/azure-pricing/the-complete-guide/#a2, 2023. Accessed: 20.06.2023.

[15] Azure reserved. https://azure.microsoft.com/en-us/reservations/, 2023. Accessed: 22.06.2023.

[16] Azure saving plans. https://azure.microsoft.com/en-us/pricing/offers/savings-plan-compute/#how-it-works, 2023. Accessed: 22.06.2023.

[17] Azure spot. https://azure.microsoft.com/en-us/products/virtual-machines/spot/, 2023. Accessed: 23.06.2023.

[18] Bootstrap. https://getbootstrap.com, 2023. Accessed: 13.08.2023.

[19] Canalys cloud 2022q4. https://www.canalys.com/newsroom/global-cloud-services-Q4-2022, 2023. Accessed: 16.06.2023.

[20] Chart.js. https://www.chartjs.org, 2023. Accessed: 13.08.2023.

[21] Crn cloud market q1 2023. https://www.crn.com/news/cloud/aws-microsoft-google-s-cloud-market-share-q1-2023, 2023. Accessed: 16.06.2023.

[22] Docker. https://www.docker.com, 2023. Accessed: 17.08.2023.

[23] Fastapi. https://fastapi.tiangolo.com, 2023. Accessed: 11.08.2023.

[24] Google cloud spot. https://cloud.google.com/spot-vms, 2023. Accessed: 23.06.2023.

[25] Google cud. https://cloud.google.com/compute/docs/instances/signing-up-committed-use-discounts, 2023. Accessed: 22.06.2023.

[26] Google pricing. https://cloud.google.com/pricing, 2023. Accessed: 20.06.2023.

[27] Mongodb. https://www.mongodb.com, 2023. Accessed: 13.08.2023.

[28] Pandas. https://pandas.pydata.org, 2023. Accessed: 11.08.2023.

[29] Python. https://www.python.org/, 2023. Accessed: 09.08.2023.

[30] Python motor. https://motor.readthedocs.io/en/stable/, 2023. Accessed: 14.08.2023.

[31] Pytorch Checkpoints. https://pytorch.org/tutorials/recipes/recipes/saving_and_loading_a_general_checkpoint.html, 2023. Accessed: 12.05.2023.

[32] technology biggest cloud providers 2023. https://technologymagazine.com/top10/top-10-biggest-cloud-providers-in-the-world-in-2023, 2023. Accessed: 16.06.2023.

[33] Tensorflow Checkpoints. `https://www.tensorflow.org/guide/checkpoint`, 2023. Accessed: 12.05.2023.

[34] Typescript. `https://www.typescriptlang.org`, 2023. Accessed: 11.08.2023.

[35] Vue.js. `https://vuejs.org`, 2023. Accessed: 11.08.2023.

[36] Docker. `https://www.nngroup.com/articles/ten-usability-heuristics/`, 2024. Accessed: 01.02.2024.

[37] Mohit Agarwal and Gur Mauj Saran Srivastava. Cloud computing: A paradigm shift in the way of computing. *International Journal of Modern Education & Computer Science*, 9(12), 2017.

[38] Saša Baškarada, Vivian Nguyen, and Andy Koronios. Architecting microservices: Practical opportunities and challenges. *Journal of Computer Information Systems*, 60(5):428–436, 2020.

[39] Sophie C. Boerman, Sanne Kruikemeier, and Frederik J. Zuiderveen Borgesius. Online behavioral advertising: A literature review and research agenda. *Journal of Advertising*, 46(3):363–376, 2017.

[40] Rajkumar Buyya, Rajiv Ranjan, and Rodrigo N Calheiros. Intercloud: Utility-oriented federation of cloud computing environments for scaling of application services. In *International Conference on Algorithms and Architectures for Parallel Processing*, pages 13–31. Springer, 2010.

[41] Chandra Chekuri and Sanjeev Khanna. On multidimensional packing problems. *SIAM journal on computing*, 33(4):837–851, 2004.

[42] Milan De Cauwer, Deepak Mehta, and Barry O'Sullivan. The temporal bin packing problem: an application to workload management in data centres. In *2016 IEEE 28th International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 157–164. IEEE, 2016.

[43] Mauro Dell'Amico, Fabio Furini, and Manuel Iori. A branch-and-price algorithm for the temporal bin packing problem. *Computers & Operations Research*, 114:104825, 2020.

[44] Tonmoy Dey, Kento Sato, Bogdan Nicolae, Jian Guo, Jens Domke, Weikuan Yu, Franck Cappello, and Kathryn Mohror. Optimizing asynchronous multi-level checkpoint/restart configurations with machine learning. In *2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 1036–1043, 2020.

[45] Tinankoria Diaby and Babak Bashari Rad. Cloud computing: a review of the concepts and deployment models. *International Journal of Information Technology and Computer Science*, 9(6):50–58, 2017.

[46] Yucong Duan, Guohua Fu, Nianjun Zhou, Xiaobing Sun, Nanjangud C Narendra, and Bo Hu. Everything as a service (xaas) on the cloud: origins, current and future trends. In *2015 IEEE 8th International Conference on Cloud Computing*, pages 621–628. IEEE, 2015.

[47] Abdessalam Elhabbash, Faiza Samreen, James Hadley, and Yehia Elkhatib. Cloud brokerage: A systematic survey. *ACM Comput. Surv.*, 51(6), January 2019.

[48] Markus Eurich, Andrea Giessmann, Tobias Mettler, and Katarina Stanoevska-Slabeva. Revenue streams of cloud-based platforms: Current state and future directions. In *AMCIS*, 2011.

[49] Emanuel Falkenauer. A hybrid grouping genetic algorithm for bin packing. *Journal of heuristics*, 2(1):5–30, 1996.

[50] Erwin Fielt. Conceptualising business models: Definitions, frameworks and classifications. *Journal of Business Models*, 1(1):85–105, 2013.

[51] Evangelia Filiopoulou, Persefoni Mitropoulou, Christos Michalakelis, and Mara Nikolaidou. The rise of cloud brokerage: Business model, profit making and cost savings. In José Ángel Bañares, Konstantinos Tserpes, and Jörn Altmann, editors, *Economics of Grids, Clouds, Systems, and Services*, pages 19–32, Cham, 2017. Springer International Publishing.

[52] Ines Houidi, Marouen Mechtri, Wajdi Louati, and Djamal Zeghlache. Cloud service delivery across multiple cloud platforms. In *2011 IEEE International Conference on Services Computing*, pages 741–742. IEEE, 2011.

[53] Inkwon Hwang and Massoud Pedram. Portfolio theory-based resource assignment in a cloud computing system. In *2012 IEEE Fifth International Conference on Cloud Computing*, pages 582–589. IEEE, 2012.

[54] Yashpalsinh Jadeja and Kirit Modi. Cloud computing - concepts, architecture and challenges. In *2012 International Conference on Computing, Electronics and Electrical Technologies (ICCEET)*, pages 877–880, 2012.

[55] Itthichok Jangjaimon and Nian-Feng Tzeng. Effective cost reduction for elastic clouds under spot instance pricing through adaptive checkpointing. *IEEE Transactions on Computers*, 64(2):396–409, 2013.

[56] Mark W Johnson and Alan G Lafley. *Seizing the white space: Business model innovation for growth and renewal*. Harvard Business Press, 2010.

[57] Christina Terese Joseph and K. Chandrasekaran. Straddling the crevasse: A review of microservice software architecture foundations and recent advancements. *Software: Practice and Experience*, 49(10):1448–1484, 2019.

[58] Kyungdaw Kang, Ilkyeong Moon, and Hongfeng Wang. A hybrid genetic algorithm with a new packing strategy for the three-dimensional bin packing problem. *Applied Mathematics and Computation*, 219(3):1287–1299, 2012.

[59] Maximilian Kiessler, Valentin Haag, Benedikt Pittl, and Erich Schikuta. Optimization heuristics for cost-efficient long-term cloud portfolio allocations. In *International Conference on Information Integration and Web*, pages 309–323. Springer, 2022.

[60] P. B. Kruchten. The 4+1 view model of architecture. *IEEE Software*, 12(6):42–50, 1995.

[61] Stine Labes, Nicolai Hanner, and Ruediger Zarnekow. Successful business model types of cloud providers. *Business & Information Systems Engineering*, 59(4):223–233, 2017.

[62] Werner Mach and Erich Schikuta. A generic negotiation and re-negotiation framework for consumer-provider contracting of web services. In *Proceedings of the 14th International Conference on Information Integration and Web-based Applications & Services*, pages 348–351, 2012.

[63] EG Co man Jr, MR Garey, and DS Johnson. Approximation algorithms for bin packing: A survey. *Approximation algorithms for NP-hard problems*, pages 46–93, 1996.

[64] Silvano Martello and Paolo Toth. Lower bounds and reduction procedures for the bin packing problem. *Discrete applied mathematics*, 28(1):59–70, 1990.

[65] John Martinovic, Markus Hähnel, Waltenegus Dargie, and Guntram Scheithauer. A stochastic bin packing approach for server consolidation with conflicts. In *Operations Research Proceedings 2019*, pages 159–165. Springer, 2020.

[66] Peter Mell, Tim Grance, et al. The nist definition of cloud computing. 2011.

[67] Christine Miyachi. What is "cloud"? it is time to update the nist definition? *IEEE Annals of the History of Computing*, 5(03):6–11, 2018.

[68] Jakob Nielsen. Enhancing the explanatory power of usability heuristics. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pages 152–158, 1994.

[69] Jakob Nielsen and Rolf Molich. Heuristic evaluation of user interfaces. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 249–256, 1990.

[70] Alexander Osterwalder and Yves Pigneur. *Business model generation: a handbook for visionaries, game changers, and challengers*, volume 1. John Wiley & Sons, 2010.

[71] Benedikt Pittl, Werner Mach, and Erich Schikuta. Cost-evaluation of cloud portfolios: An empirical case study. In *Proceedings of the 9th International Conference on Cloud Computing and Services Science (CLOSER)*, pages 132–143. SciTePress, 2019.

[72] Marcela Quiroz-Castellanos, Laura Cruz-Reyes, Jose Torres-Jimenez, Claudia Gómez, Héctor J Fraire Huacuja, and Adriana CF Alvim. A grouping genetic algorithm with controlled gene transmission for the bin packing problem. *Computers & Operations Research*, 55:52–64, 2015.

[73] Colin Reeves. Hybrid genetic algorithms for bin-packing and related problems. *Annals of Operations Research*, 63(3):371–396, 1996.

[74] Jonas Repschlaeger, Koray Erek, and Ruediger Zarnekow. Cloud computing adoption: an empirical study of customer preferences among start-up companies. *Electronic markets*, 23(2):115–148, 2013.

[75] Elvis Rojas, Albert Njoroge Kahira, Esteban Meneses, Leonardo Bautista Gomez, and Rosa M Badia. A study of checkpointing in large scale training of deep neural networks, 2021.

[76] Subhadip Roy, Sreejesh S., and Sandhya Bhatia. Service quality versus service experience: An empirical examination of the consequential effects in b2b services. *Industrial Marketing Management*, 82:52–69, 2019.

[77] Martín Safe, Jessica Carballido, Ignacio Ponzoni, and Nélida Brignole. On stopping criteria for genetic algorithms. In *Brazilian Symposium on Artificial Intelligence*, pages 405–413. Springer, 2004.

[78] Prateek Sharma, David Irwin, and Prashant Shenoy. Portfolio-driven resource management for transient cloud servers. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 1(1):1–23, 2017.

[79] Deepika Sood, Harneet Kour, and Sumit Kumar. Survey of computing technologies: distributed, utility, cluster, grid and cloud computing. *Journal of Network Communications and Emerging Technologies (JNCET) www. jncet. org*, 6(5), 2016.

[80] Damian A Tamburri, Marco Miglierina, and Elisabetta Di Nitto. Cloud applications monitoring: An industrial study. *Information and Software Technology*, 127:106376, 2020.

[81] Paul Timmers. Business models for electronic markets. *Electronic markets*, 8(2):3–8, 1998.

[82] B Wall, H Jagdev, and J Browne. A review of ebusiness and digital business—applications, models and trends. *Production Planning and Control*, 18(3):239–260, 2007.

[83] Fang Wang and Liwen Vaughan. Firm web visibility and its business value. *Internet Research*, 2014.

[84] Grant Wu, Maolin Tang, Yu-Chu Tian, and Wei Li. Energy-efficient virtual machine placement in data centers by genetic algorithm. In *International conference on neural information processing*, pages 315–323. Springer, 2012.

List of Figures

List of Tables

A. Appendix

A.1. UX questionnaire

Cloud Portfolio Manager UX Questionnaire

Hello,

thank you for partaking in the UX (user experience) evaluation of the Cloud Portfolio Manager, which i have created for my master thesis. This questionnaire will help to evaluate the developed platform and inform possible future improvements upon it. All the data collected will be anonymised and only used for academic purposes.

Regards,
Valentin Haag

* Gibt eine erforderliche Frage an

General knowledge

1. Do you have any IT related experience? *

   Wählen Sie alle zutreffenden Antworten aus.

   - [ ] Work
   - [ ] Education
   - [ ] Neither

2. How would you rate your knowledge of the cloud market? *

   Markieren Sie nur ein Oval.

   - ◯ None
   - ◯ Cursory
   - ◯ Knowledgable
   - ◯ Expert

Visibility of system status

How easily were you able to understand what the system was currently doing? Were you able to infer what next actions would be necessary/possible?

3. The system was able to provide visibility of system status. *

   Markieren Sie nur ein Oval.

	1	2	3	4	5	
Tot:	◯	◯	◯	◯	◯	Totally Agree

4. Do you have any comments on the visibility of the systems status or recommendation to improve it?

Match between the system and the real world

The terms used by the system match those used in the real world. No new words and concepts have to be learned.

5. The system was able to provide a match between its terminology and the real world. *

Markieren Sie nur ein Oval.

	1	2	3	4	5	
Tot	◯	◯	◯	◯	◯	Totally Agree

6. Do you have any comments or recommendation on the terminologies used within the system?

User control and freedom

As users like to try various actions and mistakenly perform other, there should always be a clear way to go back a step and cancel any action.

7. There was always a clear, easily visible way to cancel any action. *

Markieren Sie nur ein Oval.

	1	2	3	4	5	
Tot	◯	◯	◯	◯	◯	Totally Agree

8. Did you find instances in which you found yourself stuck and could not find a clear way back?

Consistency and standards

To offer a good user experience, an interface should be consistent.

9. The user interface felt consistent through its various components. *

Markieren Sie nur ein Oval.

	1	2	3	4	5	
Tota	◯	◯	◯	◯	◯	Totally Agree

10. Did you find any inconsistencies within the interface?

Error prevention

An interface should be designed to prevent errors. Errors can be slips or mistakes. Slips are unconscious errors caused by inattentiveness, and can be combated by setting helpful constraints and defaults. Mistakes happen consciously and are the result of badly communication by the system. They can be alleviated by enabling the undoing errors and good warnings.

11. Errors did not occur often. *

Markieren Sie nur ein Oval.

	1	2	3	4	5	
Tota	◯	◯	◯	◯	◯	Totally Agree

12. Do you have any suggestions on how to prevent certain errors you found from occuring?

Recognition rather than recall

A good design does not rely on recall of elements, actions and options but encourage recognition. In case of additional information being needed, it should be easily accessible.

13. I was able to recognise design elements in the various components of the platform. *

Markieren Sie nur ein Oval.

	1	2	3	4	5	
Tota	◯	◯	◯	◯	◯	Totally Agree

14. Do you have any comments or recommendation on symbols, elements and actions that could be made more easily recognisable?

Flexibility and efficiency of use

Offer experienced users shortcuts to speed up processes. Flexible processes can be executed in various ways.

15. The system offers several ways and shortcuts for processes. *

Markieren Sie nur ein Oval.

	1	2	3	4	5	
Tota	◯	◯	◯	◯	◯	Totally Agree

16. Do you have suggestions for possible further shortcuts to processes?

Aesthetic and minimalist design

The design of interface elements should only contain essential information required for their functionality.

17. The design is in a minimalist style and does not display irrelevant information. *

Markieren Sie nur ein Oval.

	1	2	3	4	5	
Tota	◯	◯	◯	◯	◯	Totally Agree

18. Did you find any elements that had any irrelevant visual information?

Help users recognize, diagnose, and recover from errors

If errors occur users should be informed using plain language, providing an accurate description and, when possible, a suggested solution. Avoid using technical terms such as error codes and unusual visual design for error messages.

19. If errors occurred they were well communicated and understandable. *

Markieren Sie nur ein Oval.

	1	2	3	4	5	
Tota	◯	◯	◯	◯	◯	Totally Agree

20. Were there any error that left you unsure of what happened?

Help and documentation

While in the best case a system does not need any further explanation, it may be necessary to offer documentation to enable the completion of some tasks. Said documentation should be concise, easily searchable and consist of concrete steps to be carried out.

21. Help and documentation was easily found and understood. *

Markieren Sie nur ein Oval.

	1	2	3	4	5	
Tota	◯	◯	◯	◯	◯	Totally Agree

22. What additional documentation do you think could improve the system?

Cloud Portfolio Manager UX questionnaire

Thank you for taking part in the Cloud Portfolio Manager UX questionnaire!