Conceptualizing Supply Chain Management and Supplier Relationship Management as a Service

Marco Lindgren

Master's Thesis in Computer Engineering
Åbo Akademi University
Faculty of Science and Engineering
Information Technologies
October 2021
Supervisor: Dragos Truscan

Abstract

Supply chain management and supplier relationship management processes are becoming increasingly important in the modern world, which has introduced a need for relevant software solutions. To handle these processes, Cerion Solutions Oy wishes to create a multi-tenant Software as a Service solution with the ability to connect to any customer backend systems. The problem is, how should a multi-tenant software solution of this kind be designed, and what must be taken into consideration? In this thesis, a technical design is produced, and a proof of concept for the application is developed. The proof of concept consists of a UI layer and a backend, both of which are written in JavaScript. The technical design will be used as a foundation for future developments.

Keywords: supply chain, SaaS, multi-tenancy, web service, cloud

Table of Contents

Preface

This Master's Thesis was made at Åbo Akademi University for Cerion Solutions Oy. The goal of the work was to design a multi-tenant web service for handling supply chain management and supplier relationship management tasks, and to implement a proof of concept.

I would like to thank my supervisor Dragos Truscan from Åbo Akademi University and Jonas Liinamaa from Cerion Solutions Oy for their support, assistance and counseling.

Additionally, I would like to express my thanks to my co-workers Esa Loukkola, Ilkka Tukeva and Johan Östman for their guidance on the design of the solution.

Finally, a big thank you to my mother and my brother for their never-ending support and encouragement.

Turku, 24 October 2021

Marco Lindgren

List of abbreviations and terms

CRM	Customer relationship management
ERP	Enterprise resource planning
IaaS	Infrastructure as a Service
JWT	JSON Web Token
PaaS	Platform as a Service
POC	Proof of concept
SaaS	Software as a Service
SCM	Supply chain management
SRM	Supplier relationship management
UI	User interface
VM	Virtual machine

1. Introduction

Cerion Solutions Oy, from now on referred to simply as Cerion, has recognized a need for supply chain management and supplier relationship management solutions. According to them, several companies have no systems in place to handle such tasks and instead do everything by email and spreadsheets. Cerion has developed supplier portal solutions before, but they were custom-built on the customers' pre-existing software foundations, which severely limits the re-usability and expansion of the code base for other, new customers. Their wish is to offer these functionalities as a multi-tenant Software as a Service solution that can be connected to any customer backend or enterprise resource planning (ERP) system. The problem is: how should a solution like this be designed, and what must be taken into consideration because of the multi-tenancy requirement? A basis for this concept needs to be designed to develop it further.

The scope of this work is a technical design and a proof of concept (POC) for a working supply chain management and supplier relationship management web portal. The portal should have separate front- and backends that communicate with each other through a REST API.

The project is intended to keep growing after the thesis is complete and, as such, the POC will be limited to a few core functionalities. The POC is done when its requirements are reached, which entails the implementation of a select few supply chain management and supplier relationship management tasks, as well as requirements on the usability and accessibility of the frontend part. The technical design is centered on the POC, but can go beyond its scope for the sake of future development.

2. Background

To fully understand what is needed from a web service of this kind, the relevant concepts must first be introduced. This chapter serves as an introduction to supply chain management and its processes. The different processes are analyzed and their relevancy for the POC will be defined. The supplier relationship management process is given a separate in-depth analysis because of its larger role within the project scope.

Finally, this chapter includes an introduction to cloud-based web services and an overview of different multi-tenant software architectures.

2.1. Supply chain management

In the modern world, an increasing number of businesses are no longer competing independently. Instead, the competition happens within the supply chain [1]. Figure 1 depicts a simplified model of a supply chain from the perspective of a manufacturer. The manufacturer has three Tier 1 suppliers. Each of these have one or more of their own suppliers, which are referred to as Tier 2 suppliers in relation to the manufacturer. Also depicted are customer entities at the bottom who buy products and/or services from their own suppliers and are, thus, part of the supply chain as well. Like the suppliers, customers can also appear in several tiers.

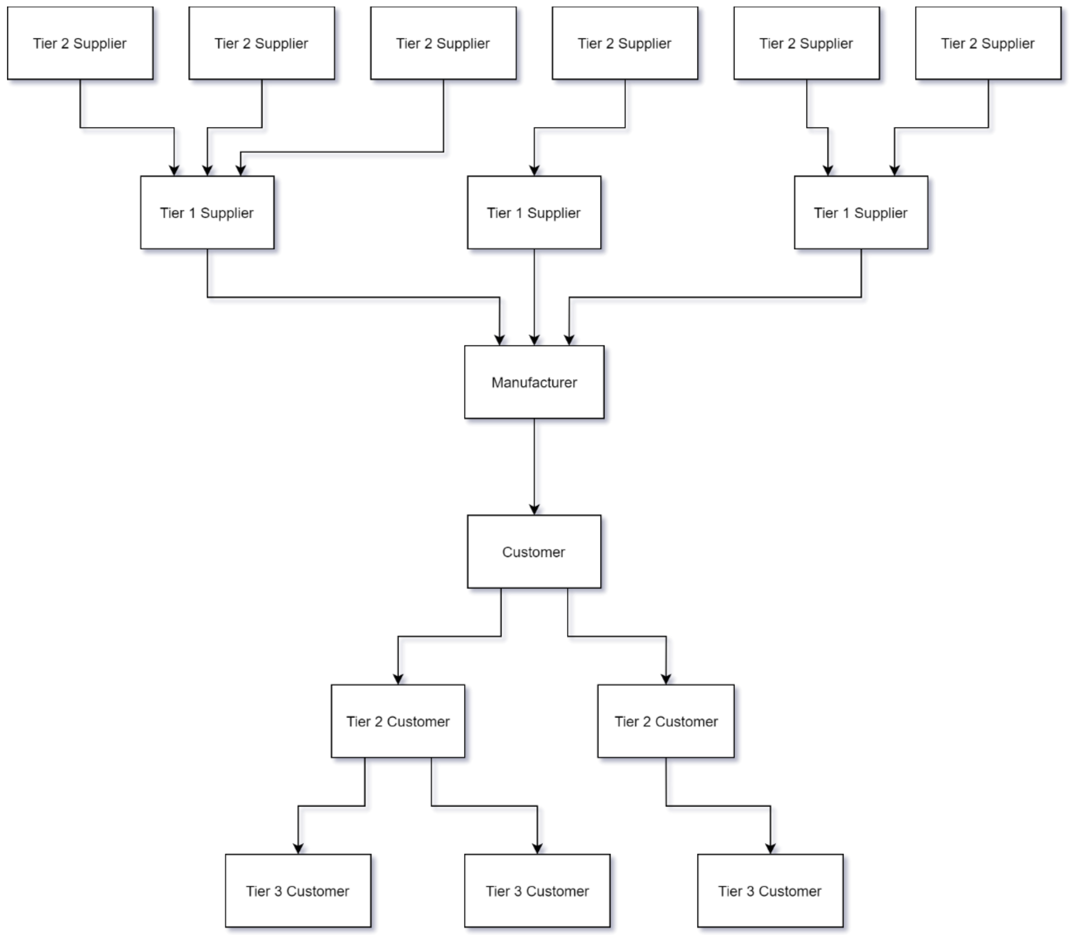


Figure 1: Example of a supply chain with multiple tiers of suppliers and customers

As is apparent from Figure 1, a supply chain is not really a straightforward chain as such, but more of a network. However, it is not only the network of businesses, but also their relationships to each other that make up the supply chain [1]. Therefore, a successful business depends on the integration of these business processes and relationships. This is where supply chain management comes in.

Supply chain management (SCM) is about integrating and managing the business processes in all parts of the supply chain. While SCM is often equated to logistics, the latter is only a part of the former [1]. In fact, SCM requires at least as many business functions as managing a regular company does. The focus does not lie on optimizing each business unit locally, however, but rather on improving the full end-to-end supply chain process [2].

SCM entails eight different processes, as identified by The Global Supply Chain Forum [1]. They are customer relationship management, customer service management, demand management, order fulfillment, manufacturing flow management, product development and commercialization, returns management, and supplier relationship management. Out of these eight, only a few are relevant to this thesis, as the customer organization is expected to have implemented the rest in its business practices already.

Customer relationship management (CRM) is the basis for customer relationship development and maintenance. The focus of CRM is to acquire new customers through marketing and retain existing ones by meeting their needs [3]. In this thesis, it is assumed that the users of the web portal have their own CRM systems already in place, as they are meant to be one possible source of the portal’s data. Therefore, CRM functionality will not be handled further in this thesis.

Customer service management is the method by which the company represents itself to its customers and provides them with a singular source of information for business matters [4]. A key function is customer inquiry responses and making order placement easier for the customer. Problem handling and prevention is also a part of this process. Customer service managers detect problems proactively and try to solve them before customers are affected [1]. This process is not in scope for the thesis or the POC, as it is closely related to CRM actions and should already be handled by the company.

Demand management entails adjusting supply to meet customer demand in a way that retains quality of product and service, as well as forecasting future demand [1] [5]. The goal is to increase the value of the service for the customer while keeping it as profitable as possible for the company. Demand managers try to synchronize the process so that as little variability exists as possible. Demand forecasting is one of the planned functions of the web portal, but it will not be included in the POC.

Order fulfillment is a process that is practiced whenever an order is placed by a customer up until the product is delivered [6]. From the business's point of view, orders should be fulfilled in a way that maximizes profitability [1]. The process is divided into two different but linked flows: order processing and order realization [6]. Order processing is a key part of the thesis and the web portal. It is in scope for the POC where suppliers can process new or updated purchase orders and fill in details on them. As the name suggests, order realization is the last step of order fulfillment. It is the point where every dependency has been received from higher tier suppliers and the product is ready for delivery. This, of course, cannot be executed in a web portal environment.

The manufacturing flow management process is mostly physical and cannot be solved using a web portal alone, which is why it is not included in the thesis or the POC. It is largely about managing manufacturing flexibility, i.e., making products quickly at a low cost [7]. One company alone cannot practice manufacturing flow management; every part of the supply chain needs to participate to reach the desired flexibility and to make the flow of the product as problem-free as possible.

Product development and commercialization helps companies in a joint effort with their suppliers and customers with developing new products and ultimately bringing them to the market [8]. The web portal is not designed for such tasks and, as a result, this process is not a part of this thesis or the POC.

The returns management process handles product returns, claims and reverse logistics, among other similar matters. It aims to minimize undesirable returns altogether which, in turn, would reduce costs [1]. There are five general categories of returns, four of which are undesirable [9]. The most significant category is consumer returns due to defects or buyer's remorse. Other undesirables include marketing returns that can happen e.g., when a product does not sell well enough, product recalls that are usually the result of safety issues or poor quality, and environmental returns in case of hazardous materials etc. The desirable category is asset returns, e.g., containers that can be reused or pieces of equipment of some kind. The plan for the web application includes functionality for claim management, which falls into this category. It will not, however, be included in the POC.

Finally, supplier relationship management (SRM) is one of the cornerstones of the proposed web portal. It will be handled as a standalone entity in the following part.

2.2. Supplier relationship management

Supplier relationship management, or SRM, is one of the business processes of SCM. It is an organized process of a buyer organization developing and maintaining relationships with its suppliers and gauging their performance [10] [11]. In this case, performance can mean very different things, depending on the context and the needs of the buyer. Common performance benchmarks are price, reliable delivery, ability to scale, consistently good quality, and efficiency [10]. Therefore, strategies need to be tailored differently to every separate relationship [1] [12]. SRM is also used for finding new potential suppliers. The process is mutually beneficial to all parties involved, as it strengthens the supply chain as a whole and can improve performance within the company [13] [14] [15].

At its core, SRM focuses on reduced costs, innovation in product development and shared success by the way of mutual commitment and collaboration between supplier and buyer [11]. A successful SRM process is measured by the impact on profitability for both parties. However, not all suppliers contribute to the buyer's success equally, and therefore, the type of relationship to maintain and the level of integration must be considered for each supplier separately.

While SRM is a process within SCM, it contains two sub-processes of its own: the strategic process and the operational process. The strategic process entails establishing the formal SRM process and managing the level of integration with the suppliers, e.g., analyzing and deciding which suppliers are critical for the company to succeed, and how the relationships are to be developed with each supplier. There are five steps to this process [11]: categorizing suppliers into segments by the value provided to the buyer company [16]; further categorizing by additional criteria, e.g., how well the needs for both parties align; analyzing the costs for how much customization and which guidelines can be provided for each segment of differing importance; aligning profitability with the supplier [17] [18] and synchronizing performance measures [19], which leads to shared risks and rewards [11]; and, finally, sharing guidelines for improvements with the suppliers.

The operational process implements what has been established in the strategic process [11]. This sub-process is also split up into steps. First, suppliers are split up into segments as per the directions from the strategic process. This results in an identification of which supplier companies are key suppliers and which ones are not. Next, the buyer company assigns teams to handle relationship management with either key suppliers individually or with non-key suppliers within their segments. When these teams are formed, they review their assigned suppliers or segments internally to identify ways to improve the relationships and processes with the suppliers in such a way that every party is incentivized to participate in the improvements. The improvement suggestions are then presented to the suppliers or segments, and processed in collaboration until all parties agree on the terms. Next, the suppliers begin the implementation of the improvements, followed by regular meetings. Finally, the performance and profitability of the improvements are measured. The measured values must be meaningful in a way that clearly displays the relationship's benefits, which include costs, return of investment and sales impact [19].

While CRM is essentially the opposite process, to benefit most from SRM, it is critical that they work hand in hand [11]. By using the SRM process effectively, the supply chain can be optimized so that there is no need to keep large inventories of goods and materials.

The scope of SRM in the POC is basic supplier management in the web portal. Every supplier has a supplier page, which lists the relationship status with the buyer, and relevant information about the company. The buyer can add notes about the supplier, attach files, and communicate with the contact persons through this function, although some of these functionalities will not be fully implemented in the POC. A user of a supplier company can only view the company's own supplier page and cannot access the pages of competitors. The idea of the complete project is to implement the supplier performance page, a supplier database that lists all suppliers for a tenant, a supplier assessment function, and supplier onboarding functionality, for finding and creating relationships with new suppliers.

2.3. Cloud-based web services

Cloud services have become very popular lately, because they enable high-speed internet hosting and eliminate the need to invest in personal server hardware and software [20]. There are a few service types to choose from, but from the perspective of this thesis, Infrastructure as a Service (IaaS) and Platform as a Service (PaaS) providers are particularly interesting, as the intention is to host the POC which, in turn, is a Software as a Service (SaaS) solution.

The term IaaS entails the physical data centers, along with the servers, storage, and firewalls that are rented for cloud hosting. This is the bare minimum needed for hosting a service on the cloud. PaaS, in contrast, refers to everything IaaS offers, as well as the underlying operating system. The management and development tools that developers can use to manage cloud instances and host applications are often included. Finally, SaaS can be an application that is completely managed and updated by the service provider, and typically accessed by end-users through a web browser. In this thesis, a SaaS product is built in the form of the POC, and an IaaS platform is required at the very least to host it. A visualization of the relationships between these service types is depicted in Figure 2.

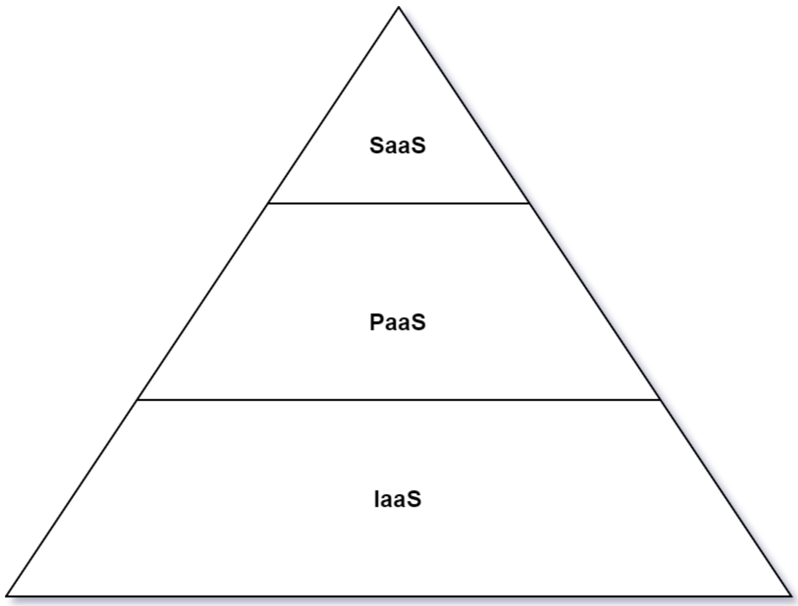


Figure 2: Relationship between IaaS, PaaS and SaaS

2.4. Multi-tenant software architectures

In the software context, a tenant is an organization or a group of users who share data access rights on the group or organization level [21].

Multi-tenancy is a type of software architecture where multiple tenants use the same software instance running on the same hardware [22]. While the software instance is shared, tenants have separate privileges and cannot access each other's data. This contrasts with single-tenant architectures, where each tenant has separate software instances, hardware, and databases, as seen in Figure 3. Furthermore, multi-tenant architectures are generally cheaper and easier to scale.

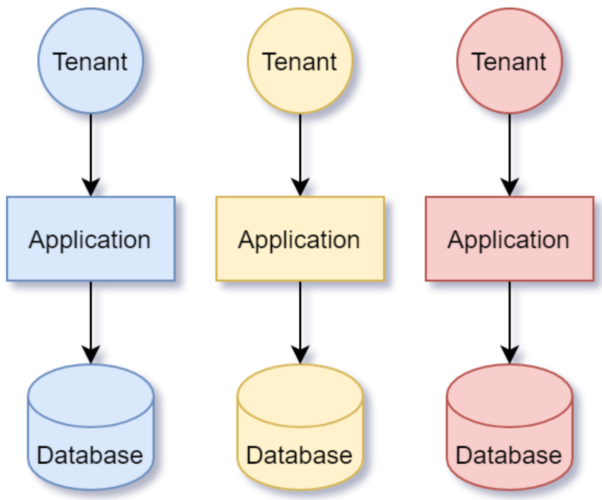


Figure 3: Single-tenant architecture

Multi-tenant architectures benefit SaaS platforms in a few ways. Because the underlying code and application infrastructure is shared among tenants, scaling the application, maintaining and updating the code base, as well as configuring the environment are considerably less complicated than in single-tenant architectures, where these operations must be done separately for every environment [23]. This also leads to lower operating costs, for both the SaaS provider and the customer. The weaknesses in multi-tenancy lie primarily in the limitations of any tenant-specific customizations on the application level. These customizations must be part of the application that is used by everyone, with only certain permissions allowing access.

There are three basic multi-tenancy database architectures [24], as is depicted in Figure 4. The first picture shows the “dedicated database” model, where each tenant has a separate database, and only the database that belongs to the tenant can be accessed. In other words, this model provides tenants with the highest degree of data isolation out of the three displayed. However, higher data isolation leads to lower scalability [22]. The second model depicts an architecture in which one database is shared by every tenant. Every table is shared, and access is only restricted with user rights and unique identifiers for each tenant. The third model is similar, but every tenant has access to the database tables through tenant-specific database schemas, which provide additional data isolation compared to the second model.

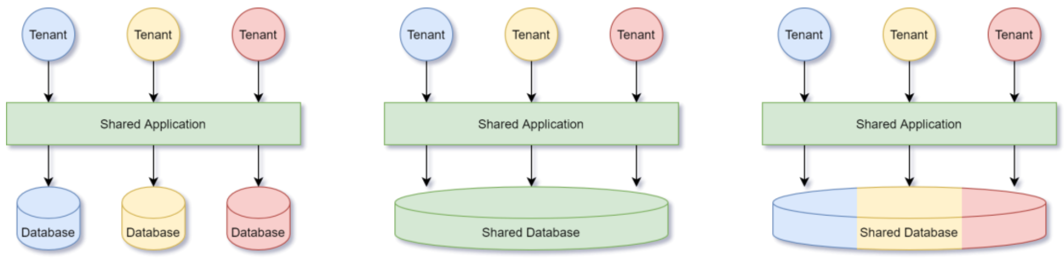


Figure 4: Three common multi-tenancy database models

The dedicated database model allows for customizations for each individual tenant without affecting the other tenants. However, the process of introducing changes to the schemas that should affect all databases becomes more delicate and requires careful management [25]. Running multiple databases that may accommodate high usage can be an expensive endeavor, but some cloud providers offer elastic resource pools for databases [26]. This means that the available resources for every database change dynamically, based on performance demand, which is more cost-effective. This model scales better than a single database for every tenant, because the number of available indices for each tenant is smaller the more tenants the database contains. Operations, such as creating backups and recovering past tenant data, are also easier and less risky, since every change only affects a single tenant's database, leaving the others untouched.

The data in the shared-database-shared-schema model is separated by assigning an identifier to each tenant and selecting data with that identifier. The shared schemas are easier to mass update for every tenant. However, this comes at the cost of customization. For example, columns that are created to be used by a single tenant also exist for other tenants. A shared database with shared schemas offers no inherent data isolation. Making sure only the correct data are accessed becomes the responsibility of the developers [25]. Some database software products support tenant scope enforcement for database rows, alleviating some of these concerns. Shared database models are cheaper than per-tenant databases with an equivalent number of tenants, and have a higher potential maximum number of tenants. However, if some tenants increase the workload by an abnormal amount, it may impact the performance of every other tenant. Backing up and recovering data without affecting all tenants also becomes very complicated.

The shared-database-separate-schema model is like the previous model, but all tenants have separate database schemas. This results in partial data isolation, where tenants can have customized tables without affecting the other tenants' data [27]. This has the same weakness as the separate database model, as making global schema changes requires modifications for every single schema. Additionally, in the case that the SaaS provider wants to gather statistics from all tenants, having separate schemas may result in duplicate identifiers, especially if identifiers are generated as numbers in running order [28]. This makes it difficult to merge queries. Backing up and recovering data is still similarly complicated as in the shared schema model [27].

Using one code base comes with its own challenges. According to Krebs et al. [21], “If one single code base is used, the application has to be widely configurable to be adapted for customer-specific needs.”

3. Design

This chapter details the proposed architecture of the solution. This includes the proposed multi-tenancy model that will be used, as well as a comparison of cloud providers, namely IaaS and PaaS providers. Finally, a decision on how to package and deploy the project will be reached.

3.1. Creating a multi-tenancy model

As an SaaS project, the built solution will logically be multi-tenant [22]. The POC requires that it can communicate with the customers’ own ERP systems. Because some customer-specific data must be stored somewhere in the application itself, the resulting multi-tenancy architecture will approximately be a hybrid between the dedicated database and the shared-database-shared-schema models. This is depicted in Figure 5 below, where all tenants use the same application, but communicate both with their own databases and the unified database of the solution. The purpose of the unified database is to store user information and operational data specific to the application.

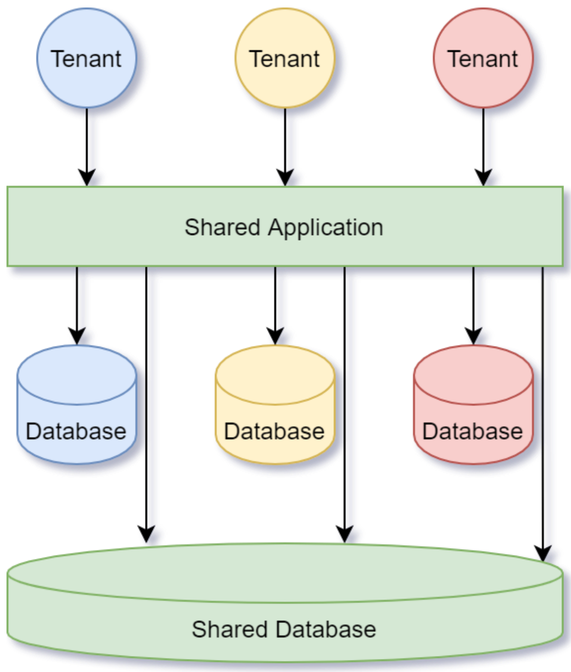


Figure 5: Proposed multi-tenant database architecture of POC.

At this stage, the shared-database-shared-schema approach was selected purely for the POC. At later stages, a “sharded” variation of the model might have to be implemented. “Sharding” means that all tenants are split into different databases depending on the tenants’ individual sizes. An additional catalog database instance would keep track of the locations of each tenant and their data. This would allow for the possibility to organize and move around tenants, e.g., based on their activity, their impact on the database workload, or their subscription plan [25]. This would be a reliable way to scale up the architecture seamlessly in the future.

3.2. Cloud provider comparison

Because of the POC’s modular nature, it does not matter much where both the frontend and the backend are hosted. Nevertheless, a cloud platform that both fits the solution and is relevant to Cerion’s interests is preferable. The three chosen platforms are Amazon Web Services, Microsoft Azure and Google Cloud Platform. These were selected as candidates because of their popularity and prominence in the cloud provider market. The cloud providers’ prices, relevant features and services will be compared in this chapter.

3.2.1. Amazon Web Services

Amazon, formally Amazon.com, Inc., is as American technology company. While Amazon started as an online bookseller, they expanded their business to other goods as well and became one of the largest online retailers in the world [29]. They entered the IaaS market as early as 2006 with Amazon Web Services, or AWS for short. The AWS brand had already existed since 2002, but back then it was simply a service that provided data about website traffic and marketing statistics. At launch, AWS offered data storage and processing over the internet with their Simple Storage Service (S3) and Elastic Compute Cloud (EC2) services, respectively. It has since grown to include a wide selection of different technologies, including artificial intelligence and internet of things (IoT) -related endeavors.

Because Amazon has been a long-time provider of cloud services, their AWS platform is very mature and has wide coverage of data centers all over the world, with documentation and customer support available in multiple different languages [30]. According to the Synergy Research Group [31], Amazon controls approximately a 33% share of the worldwide cloud infrastructure market.

With security as Amazon’s top priority [32], it is fully enterprise-ready but is also a good fit for many other types of cloud platform customers, as deployment of solutions is described as fast and easy [33]. AWS offers a large selection of different services including virtual machines that run common operating systems such as Linux or Windows, file storage, databases, serverless cloud functions, virtual reality tools, robotics services and many more [34]. The platform is ideal for solutions designed to work in virtualized environments, with the focus lying on the public cloud. Private cloud and hybrid solutions can be built using the platform, but are not a priority for Amazon. AWS has a cost-effective pricing model that even includes a free tier for small computing tasks, though the cost structure itself has been criticized for being confusing, making it hard for enterprises to manage their costs efficiently.

AWS has all the tools needed for deploying the practical part of this thesis. With such a strong set of features, Amazon’s IaaS platform is a strong candidate. However, because hybrid cloud solutions are not a priority for Amazon, this raises the question whether there is enough support available for cases such as this.

3.2.2. Microsoft Azure

Microsoft Corporation is a multinational technology company based in the United States. While they are known for developing software such as their popular Windows operating system and the Office family of business applications, they also produce hardware in the form of laptops, computer peripherals and Xbox-brand video game systems [35]. Microsoft entered the cloud computing market when their integrated PaaS and IaaS platform Azure was announced in 2008 and finally brought to public use in 2010.

Like AWS, Azure has a wide array of different services for cloud solutions, but its strengths lie particularly in data lakes, machine learning and IoT. It is the second-largest cloud provider and claims 18% of the cloud infrastructure market share [31]. Azure is applauded for its high availability and scalability, as well as their strong security [33], but it is also considered to be a platform that requires know-how and a lot of management to run things on it. Azure has a similar service offering as AWS, with virtual machines, serverless functions and databases, but also offers an insight into experimental features such as quantum computing [azure docs]. It also supports many programming languages and tools, with a robust support for open-source software [33]. However, its pricing is as confusing as Amazon's, with multiple licensing options [30].

Microsoft has a long history of working with enterprise customers and knows their needs: not all companies rely on purely cloud-based architectures and instead run their own data centers. Therefore, Azure is built to be a good fit for hybrid cloud solutions, and a lot of Microsoft's own enterprise software is also built into Azure, such as SQL Server and SharePoint [33]. Enterprises that already use Microsoft products tend to favor Azure for this reason. However, because of issues with documentation and customer support, Azure has been criticized for not feeling as enterprise ready as expected [30]. Regardless, customer support is available in several languages, and Azure data centers exist in several locations across the U.S., Europe and Asia.

With a good set of features, hybrid cloud support and integration with existing Microsoft products, Azure seems like a good candidate for hosting the practical part of this thesis. The only potential concern lies with the quality of the support, but it is not a showstopper by any means.

3.2.3. Google Cloud Platform

Google LLC is an American company owned by Alphabet Inc. that started as an online search engine company and has since expanded into other products and services, such as the Chrome web browser, the email service Gmail, the video sharing site YouTube and the mobile operating system Android. The Google search engine is still very popular, with an estimated 70% of all search requests in the world being handled by it

[36]. Google had been dabbling in cloud computing for several years before launching their IaaS platform Google Cloud Platform, or GCP, for the public in 2013.

While GCP has the same basic services, its number of features and services is not as high as AWS or Azure. It is, however, strong in analytics, machine leaning, big data and technologies for building artificial intelligence as well as native cloud applications [30]. While its strengths lie in the public cloud, it is possible to build hybrid or private cloud solutions, though GCP is often selected as a secondary cloud provider in these cases [33]. Its fast I/O operations and strong storage have been applauded, but the limited set of compatible programming languages and tools have been noted as a particular weakness.

Like its competition, GCP is available in multiple regions in many languages but has fewer global data centers. It integrates well both with Google's own and open-source services, but its technology is criticized for being almost completely proprietary, making it difficult to have any control over some of their services, e.g., virtual machines. As a result, migrating away from GCP is a complicated process [33]. The development process in GCP heavily relies on DevOps, making it ideal for agile development practices.

Since GCP is mostly designed for the public cloud, it might not be a perfect fit for hosting the practical part of this thesis. However, the intention of this comparison is not to find the IaaS platform with the most features or a general best cloud provider, but to find the best fit for the practical solution. Because hybrid cloud solutions are possible to build with GCP, it is still worth considering as an option.

3.2.4. Comparison

As can be seen from the descriptions above, the three chosen cloud providers have similar offerings. Despite this, different providers can have vastly different performance metrics [37]. As such, comparing their features directly would be unproductive, and other metrics such as performance and cost, as well as the performance-cost ratio need to be examined instead.

Comparing performance of virtual machines (VMs) is challenging, as traditional benchmarking tools cannot be directly applied to virtual environments. Particularly, Transaction Processing Performance Council (TPC) benchmarks have been proven to be ineffective for this purpose [38]. However, according to a number of performance comparisons of Azure, Google Cloud Platform and AWS conducted by Muhammad-Bello and Aritsugi [37] [39], it was found that some high-performance computing (HPC) tests can be applied to the cloud as well, though some fluctuation is still present [40]. Below are some of their findings.

The results of the different analyses showed that while Azure generally has the highest VM costs of the three, it also has the most performant CPU in the medium and large VM instance sizes, while AWS had the worst across the board. However, AWS had the most consistent CPU performance. In memory tests, each provider had their own strengths and weaknesses, with Google Cloud Platform having generally better memory performance with multicore CPUs and AWS having an advantage with single core ones. Reportedly, all three platforms had similar disk I/O performance, but the size and type of storage medium played a bigger role here. Generally, hard disks performed better in sequential I/O tests, while solid-state disks performed better in random I/O tests. It was also found that out of the three IaaS providers, Google provided the most customer value on small and large VMs, with Microsoft claiming victory on medium-tier VMs. Amazon’s offerings provided the least performance-cost value, sometimes with large differences compared to the other two.

Muhammad-Bello and Aritsugi proceeded to benchmark each cloud provider in real-world applications by running data, graph and in-memory analytics algorithms on VMs of different sizes. In these cases, Azure was the best in almost every single one, except in performing data analytics on large-tier VMs. However, in that category, all platforms performed similarly.

From these results, it seems that every cloud platform is competent enough in their own way. However, any results that purely rely on high performance can be considered irrelevant for the purposes of the project and should not be taken into consideration. Nevertheless, the price-performance ratio is important, and it seems like Microsoft’s Azure has the best offering of the three in a general sense.

Another significant factor for selecting a cloud platform that fits the needs of the project is the opinion of Cerion. From Cerion’s point of view, choosing Microsoft Azure would be beneficial for a couple of reasons: Cerion is already partnered with Microsoft and using more of their services would benefit both parties in the long run. Furthermore, Cerion has previous know-how in cloud computing with both Azure and

AWS. From a pure logistics standpoint, it would make sense that Azure could be the platform used for hosting the project.

Considering performance, cost, value, and Cerion’s opinion, Microsoft’s cloud platform Azure is the best candidate for the job. It will be used to host the project in its entirety.

3.3. Packaging and deploying the solution

To get the project up and running on the internet, the methods by which both the frontend and backend should be packaged and deployed must be considered.

Because the backend is designed for a growing number of tenants, it must be packaged and deployed in a way that allows for easy scaling. While this can be done by simply pushing the code to an Azure App Service that handles the build- and deployment process automatically, a Docker image will be built and packaged into a container instead.

Docker is a program that allows the user to create containers, which is a way to isolate an application from the environment, as well as deploy and run them. This makes the build and deploy process easier for the developers, and any ambiguity with whether the application will run on a specific machine is abolished. A container challenges the traditional virtual machine model by not virtualizing the whole machine, but only parts of the operative system, effectively removing a performance-heavy layer from the picture [41]. The difference is detailed in Figure 6 below. The resulting container is much smaller than a full virtual machine instance because of this, and can be deployed to an environment quickly. Furthermore, Docker containers perform better than similar virtual machine setups according to a study made by Yadav et al. [42].

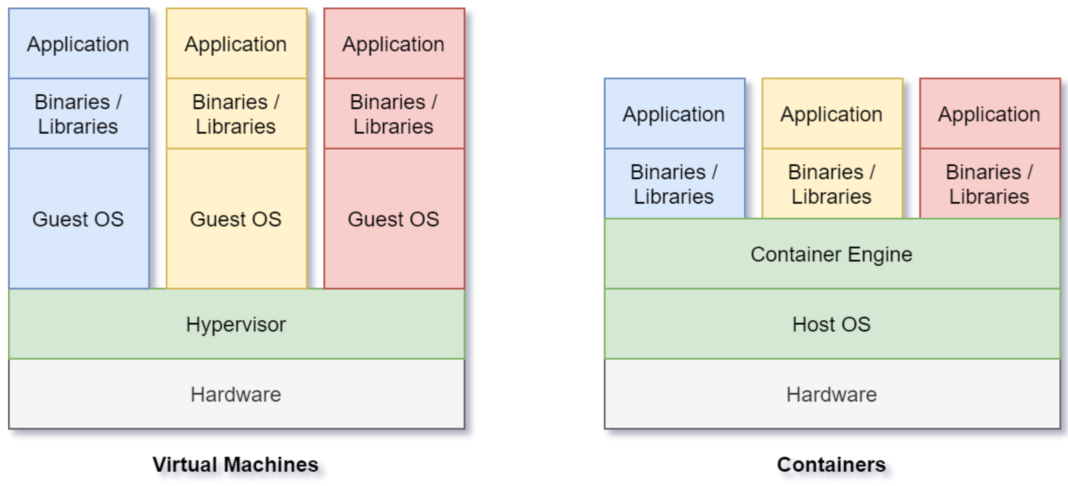


Figure 6: Architectural difference between a virtual machine and a container

The next question is how to deploy the container. In Azure, the two available services that fit the solution the best are Azure App Service and Azure Kubernetes Service. Both services can host containers and have a robust continuous integration and continuous delivery (CI/CD) pipelines and good security [43] [44]. They can also be scaled up when needed. However, there are major differences in the underlying technology. App Service is a PaaS solution with a cheap pricing model that is meant for straightforward web applications that uses one of the natively supported languages [43]. Kubernetes Service, in contrast, is a language-agnostic, managed container cluster that can be easily migrated from on-site use to the cloud or from one provider to another if necessary. It is also ideal for services that require multi-tenancy and complex networking [45].

Based on this information and the need for multi-tenancy, it would make sense to host the backend in a Kubernetes cluster. With it, it would be possible to deploy the development environment to an on-site cluster and only have the production environment in the cloud. The frontend is a slightly different matter. Because the frontend will only ever communicate with the one backend, it would be enough to run it in a container on an Azure App Service instance.

4. Practical implementation

The practical implementation consists of the technical design and the POC itself, which has been split into the user interface (UI) layer, the backend layer and a chapter that discusses the authentication and security logic across the application.

4.1. Technical design

This chapter includes the functional and non-functional requirements for both the front and backends, the in-depth architecture of the system, and the database design for the shared database. Some details may go beyond the scope of the POC.

4.1.1. Functional requirements

The portal should be able to perform several SRM and SCM tasks. SRM tasks include a supplier database where a buyer can view a list of all suppliers, a way to gauge and analyze the performance of each individual supplier, supplier assessment functionality, and supplier onboarding for acquiring new suppliers. SCM functionality entails quotation requests, forecasted demand, claim management, certificates, declarations, collaboration, dispatch and inbound logistics, as well as order management. Because of the limited scope of the POC, it will handle all abovementioned SRM tasks except for supplier onboarding and supplier assessment, and only order management from the SCM portion.

The supplier database should be a list of all the suppliers that are related to the user's company in the buyer role. The rows in the list can be filtered by name, number, status, and supplier category. Each row contains a button that will take the user to a supplier detail page. This page contains the contact information of the selected supplier, including a list of all contacts associated with it, and their assessment status. This page will eventually have more performance metrics, document collaboration and comment functionality as well, but it is out of scope for the POC.

Order management entails being able to manage orders from both the supplier and the buyer perspectives. The main page should have a list of every open and closed purchase order, which can be filtered based on status, and a free text search field. Selecting a purchase order from the list opens a page that lists all purchase order items, along with their prices, quantities etc. The details of each item are editable when the order's status is “Received” or “Changed”, otherwise the orders are strictly read-only. This page should also display an audit trail that shows which user modified the order and when that change took place, but this functionality will not make it to the POC in its entirety.

When the user logs in to the application, a dashboard-like overview of the current statistics should be shown, with the number of open purchase orders and other, similar metrics. It should be possible to navigate to every main view from every page, with a navigation bar. The individual pages should have a breadcrumb-navigation component as well to make it easier to understand the navigation paths. The user should also be able to log out and access personal settings easily from any view.

The frontend layout should be built to be responsive so that the application is usable and will not break or overflow on small screens, e.g., on mobile phones. It should also be accessible to people with disabilities who might rely on screen readers to read and navigate web pages. For example, forms should clearly mark required fields and invalid inputs, as well as provide feedback on what kind of data was expected. Preferably, every field should have a tooltip that explains its purpose. These tooltips can be edited by users with the administrator role. Every action taken in the front end should provide immediate feedback. If the user opens a different page, that page should open immediately. If something relies on asynchronous operations, such as sending data to the back end and awaiting a result, a loading spinner should be displayed to indicate that something is happening. Additionally, successful CRUD operations (Create, Read, Update, Delete) should display a toast as feedback regardless of whether the operation was successful or not.

While tenants generally have either a supplier or a buyer role, one of the core concepts of this project is to enable many-to-many compatibility within the roles, e.g., enabling a supplier to take on a buyer role towards the company's own suppliers. In essence, this gives tenants access to both SRM and SCM functionality toward their respective tiers of buyers or suppliers. The objects used for tenants also contain the access information to different functionalities within the application for its users. The tenant object's access right can only be edited by a user in the super admin role within Cerion. There are three types of access rights individual users can have: read-only access, edit access and administrator rights. Users with read-only access can only view data he or she has access to, while users with edit access can also edit it. By default, administrators have access to all settings and data within the tenant context, except for the super admin that can manage other tenants' access settings as well.

In addition to these requirements, the portal should use REST APIs both for communication between the front- and the backends, as well as between the backend and the customers' ERP systems.

4.1.2. Non-functional requirements

The portal has a few non-functional requirements. For every change that is made to business-critical objects, for example purchase orders, the action should be recorded in an audit trail for traceability. Using the application should be fast, responsive and should not freeze or crash in either the frontend or the backend. The application should also be built in a way that no piece of data leaks to a tenant that should not see it. Both the frontend and the backend should be coded so that maintenance is easier. This can be achieved in part by prioritizing reusable components and by reducing duplicate code.

4.1.3. Portal architecture

Figure 7 below depicts the high-level architecture of the portal and how it communicates with its own parts and external systems. The portal itself consists of a separate frontend and backend, which communicate with each other through a RESTful API. When the user performs an action in the frontend layer, it calls the backend API, which in turn returns the desired data or message. Calls that require the customer's data to change are forwarded to the customer's ERP system through a connector that utilizes its API. If a data change is detected in the customer's own backend, the portal's backend is notified, and the stored records are updated.

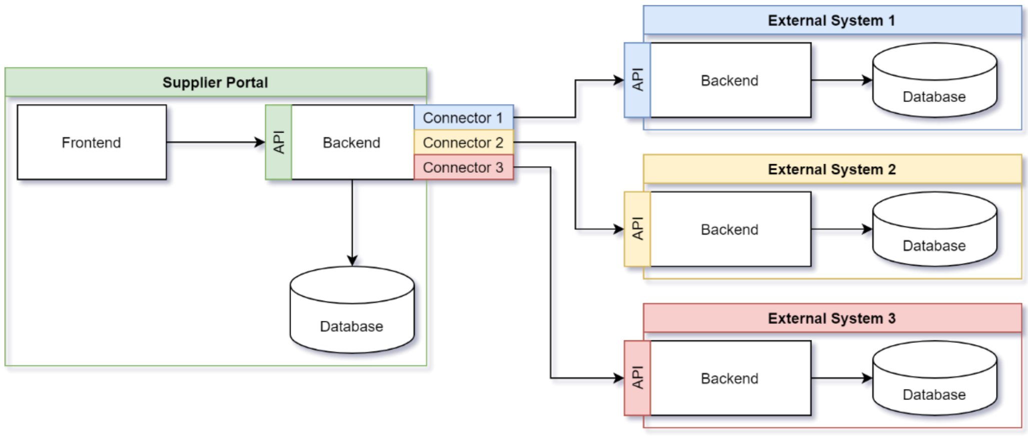


Figure 7: High-level portal architecture

One of the core ideas is that the backend could be extended in the future with functional modules that would offer new functionality. The availability of different modules for each tenant would be dependent on what the customer is willing to purchase for their needs.

Some work is required on the customer's end as well to accommodate the incoming API calls and outgoing notifications before the users can properly start using the portal. Connectivity with at least SAP and Microsoft Dynamics is planned, with support for more ERP software in the future, but these are not included in the POC.

4.1.4. Database design

As mentioned earlier, the database architecture is a hybrid between two multi-tenant database architectures. The focus in this part lies on the shared database that is connected to the backend of the solution. Since Azure functions as the cloud provider of the solution, the best-suited database software is Azure SQL Database. Azure SQL Database uses the Microsoft SQL Server database engine, which has built-in support for multi-tenancy designs [25]. In the POC, however, a local SQLite 3 database is used for simplicity. The SQLite database will not be used after the POC has been delivered, as an embedded database is neither practical nor scalable when more developers join the project.

The planned database tables on the portal side for the POC are User, Contact, Company, Company Relation, Associated Company, PO Header and PO Item. They are set up in the following way: each individual user has their information stored in two tables, namely User and Contact. Contact contains the personal information such as name and email, while User contains portal-specific information such as password and user role. The two tables have a one-to-one relation, and a User record requires a corresponding Contact record. However, the opposite is not true. Contact records can be stored individually, which means that those people do not have login access rights to the portal. A Contact belongs to one primary Company, and one Company record can have many Contacts. A Company record contains information about a company, whether it is a customer of the tenant or the tenant's company itself. To enable many-to-many buyer-supplier links between different Company records, the Company Relation table is used. It simply specifies the buyer and the supplier in its records. It is used as a junction object to, for example, populate a list of a tenant's suppliers or buyers. The Associated Company table enables a Contact to be associated with multiple Company records that are not a Contact's primary Company. This would enable a user to act on behalf of another Company in the case of partnerships or hierarchical enterprise structures.

The PO Header model contains a purchase order. It includes the header details of a purchase order and the buyer and supplier Company records. These details primarily come from the tenants' own ERP systems. PO Item, in turn, includes the details of each individual item in the purchase order, such as item information and price. PO Headers must belong to a buyer Company record and a supplier Company record. PO Items belong to one PO Header, but a PO Header can have as many PO Items as necessary. The complete database schema can be seen in Figure 8.

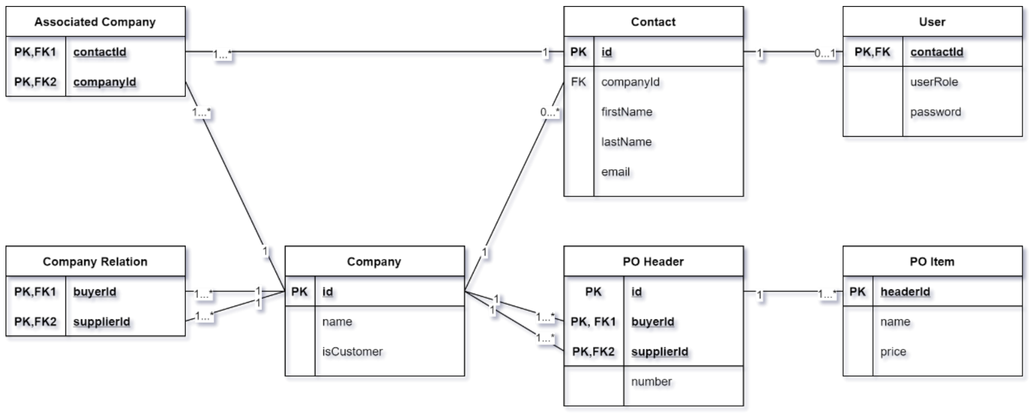


Figure 8: Diagram of database schemas

Of course, once more features for the portal are planned and implemented, the database design will grow as well. These are merely the bare minimum schema required for the core POC functionality.

4.2. UI layer

Because of the requirements that the UI layer should be responsive, accessible and easily maintainable, it was decided early in the project that React would be used as its core technology. React is a JavaScript framework that specializes in building interactive user interfaces with component-based design patterns. While other, similar libraries exist, e.g., Angular or Vue, React was chosen because of Cerion's interest in it and the author's experience with the framework. With React, web pages are built inside the JavaScript code using an XML-like syntax called JSX. A quick way to get started with a React project is to use Create React App (CRA), which that includes developer and build tools for React development. The JavaScript code in this project is written using the ECMAScript 6 (ES6) standard, which the tools included in CRA can compile down to ECMAScript 5 (ES5) for compatibility with older browsers, though this is not a requirement for the POC.

Using a layout template eliminates the need for a dedicated UX designer, and makes the start of the development process smoother for the programmers, so the UI layer for the project was built on top of a premium layout template called CoreUI Pro. There are multiple versions of CoreUI Pro available for different frameworks, including a React version that was used in this implementation. The React version is built on CRA and includes numerous third-party UI components.

The UI layer is built to be fully responsive, which means that the content scales properly to fit smaller screens, e.g., mobile phones, without breaking the layout, therefore making the website easier to use on such devices. This is achieved partially with the CSS classes supplied by CoreUI, but also with strategic use of custom CSS and layout ordering. Many of the pre-made CoreUI components can be used as-is for the layout and content containers, and have been used as the base for most of the functional components in the project. Much effort has been put into creating reusable components to reduce unnecessary duplicate code and help with maintenance.

The aim is to conform to the Web Content Accessibility Guidelines (WCAG) as closely as possible so that the portal is also accessible to people with disabilities who rely on screen readers. To reach these standards, everything from text size and contrast to appropriately used HTML 5 and Web Accessibility Initiative – Accessible Rich Internet Applications (WAI-ARIA) tags have been carefully thought through and implemented during the development of the POC. The previously mentioned responsive design also contributes to accessibility.

The user has access to different views depending on their role. Currently, the system supports two roles: buyer and supplier. Buyers can create and view purchase orders, as well as view and interact with suppliers, while a supplier can manage the purchase orders made by buyers. In some cases, the user can switch between the roles to manage both buyer and supplier-related tasks.

Forms will be handled in a way that is as user-friendly as possible. Required fields will be marked with an asterisk (*), and before a form is submitted, validation messages will appear on fields with invalid input values, e.g., empty required fields, or letters in number fields etc. For even more clarity, there could even be a summary of all the invalid fields near the submit button.

Toasts, a type of pop-up notification, are used for notifying the user about results from REST requests and other operations, regardless of whether they are successful or not. The toasts are color-coded to inform the user of the severity of the notification by simply looking at it. For example, successful REST requests display a green toast, while unsuccessful ones display a red one. A descriptive icon could be added to these toasts to make them immediately obvious for colorblind people as well. For people who use assistive technologies, such as screen readers, the toasts are announced immediately as they appear.

4.3. Backend layer

Because of the API architecture and the fact that the UI layer is backend-agnostic, almost any backend would have been able to get the job done. For the purposes of the POC, a Node.js -based backend was chosen to keep the stack consistent in compatibility and language, i.e., JavaScript written in the ES6 standard. Node.js is modern and asynchronous, and has fast, non-blocking I/O. It uses a lot of CPU power since traditionally, it only uses one CPU thread, but it does so very efficiently. Newer versions of the runtime environment also support multiple threads aimed at CPU-heavy tasks. Because Node.js currently only understands the older ES5 standard, the ES6 code is compiled using Babel. Third-party packages are fetched through the Node Package Manager (NPM), which has a large selection of helpful and functional component libraries.

The main framework used in the backend is Express, which is a tool for building web applications, but also for creating REST APIs. For Node-based servers, Express is the de facto standard because it is light and compatible with a vast selection of supporting libraries and middleware. Some of the packages that were used in the solution are “morgan”, “cookie-parser” and “passport”, which handle request logging, cookie management and authentication, respectively.

Because one of the main points of this project is to connect with multiple different customer ERP systems via their APIs, the database management of the backend should be as straightforward and effortless as possible. To accomplish this, an Object-Relational Mapping (ORM) tool was used, which enables the developer to do database operations using the JavaScript language and objects. This would make it possible to map the received data from the customer’s system into the portal’s own JSON structure, which could then be handled as necessary.

Because this project uses an SQL database, the library Sequelize was chosen as its ORM tool. Sequelize supports multiple SQL dialects, for example, MySQL, MariaDB and SQLite. Because the same syntax is used to manage each of them on the code level, it makes local development with an SQLite database easier, since it would have only minor configuration differences with a production build.

4.4. Authentication and security

Security is handled differently on each layer of the project. Most of the security measures are placed in the backend, but the frontend must also pre-emptively stop misuse of the system. This chapter intends to explain the implemented and planned security measures and why they are needed.

4.4.1. UI layer

Every piece of data that is sent from the UI layer is validated. The user is first informed on the form level if the inputted data is faulty. Then, when the form is submitted, the data is once again sanity checked and then sent to the backend. Ideally, the frontend should contain as little business logic as possible, but a little is required since the portal is backend-agnostic and operates purely on API requests. To avoid hard-coded duplicate validation data, a master copy of every form field and their requirements could be stored in the backend and sent to the frontend when the form is loaded. This data could then be adapted to the validation library in the frontend.

At login, if the user inputs the correct login details (email and password), the UI layer receives a JSON Web Token, which contains authorization information. This information is stored in a Redux store, and is used for determining which pages and features the user can access. Pages that should not be visible will not be loaded on the client side. This is achieved by lazy loading the page data on demand. This is of course not foolproof, but frontend authorization rarely is. The following backend layer section contains more details about this.

Every HTTP request made towards the backend (GET, POST, PUT, PATCH, DELETE) is done using a custom REST request class that is built using JavaScript’s Fetch API. All requests, except for the login request, must include the “Authorization” header with the user’s JSON Web Token as the value.

4.4.2. Backend layer

All requests that are received in the backend first go through an authorization phase. The main library used for all the heavy lifting with authentication and authorization in this project is Passport. It is a middleware that can be plugged in to Express and can be configured with different authentication strategies, for example JSON Web Tokens, Google OAuth, OpenID, different social media platform logins and many more. According to their own documentation, the library supports over 500 authentication strategies.

A JSON Web Token, or JWT, is an encoded token commonly used for authentication. It consists of a header, a payload, and a signature. The header contains information about the signing algorithm. The payload includes all the data that wants to be sent, and the signature is generated by using the header information, the payload data and a secret key. Because the header and the payload are used to generate the signature, if the backend receives a valid JWT that decodes correctly, it can safely be assumed that the payload has not been tampered with.

In this concept, JWTs were chosen as the main authentication method because of its easy syntax and its independence from external services. The token payload will contain information about the user making the requests, along with their authorization information (roles). The timeout for the generated JWT is short, about 15 minutes, after which it is expired and is no longer valid. The expiry time is kept short to mitigate the risk of a third party using the token to make requests with someone else's identity. The JWT is only stored in the memory while the application is running and will disappear when the page is closed or refreshed as a result. This is a conscious decision because, for example, if the token is stored in the local storage, it can be accessed with JavaScript from another application and can be used in cross-site scripting (XSS) attacks [46].

When the user logs out, the token is removed from memory and the user is redirected to the login page. Additional logic with a refresh token is implemented so that the user is not forcibly logged out every 15 minutes. The idea is that for persisting the login, a long-lived refresh token is given to the frontend at login, and it is used for requesting a new JWT in regular intervals based on the JWT's expiry value. Every time a new JWT is requested, the refresh token changes and is saved in the database and associated with the user. The refresh token is stored in an HttpOnly, SameSite cookie that cannot be accessed by JavaScript or external domains. This prevents cross-site request forgery (CSRF) attacks [46]. To automatically log in the user on the next site visit or refresh, the same refresh token can be used to check if a matching token is found in the database. If there is a match, the user gets a JWT. Otherwise, they are taken to the login page. Manually logging out of the service would invalidate the refresh token in the database. The complete flow is detailed in Figure 9, Figure 10 and Figure 11 below.

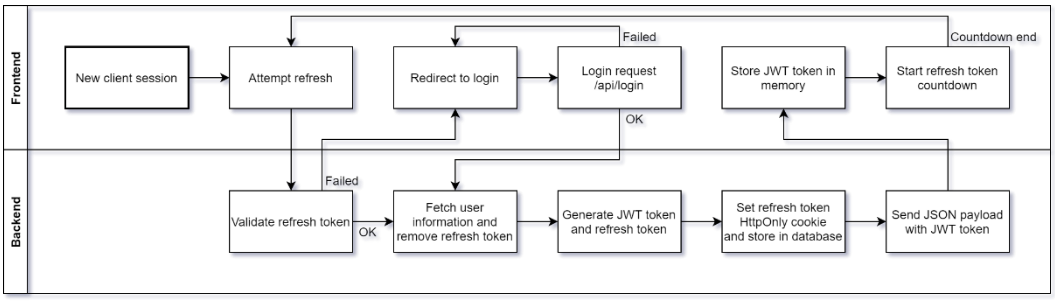

Frontend
Backend
New client session
Attempt refresh
Redirect to login
Login request /api/login
Store JWT token in memory
Start refresh token countdown
Failed
Countdown end
OK
Failed
Validate refresh token
OK
Fetch user information and remove refresh token
Generate JWT token and refresh token
Set refresh token HttpOnly cookie and store in database
Send JSON payload with JWT token

Figure 9: Authentication and refresh token flow

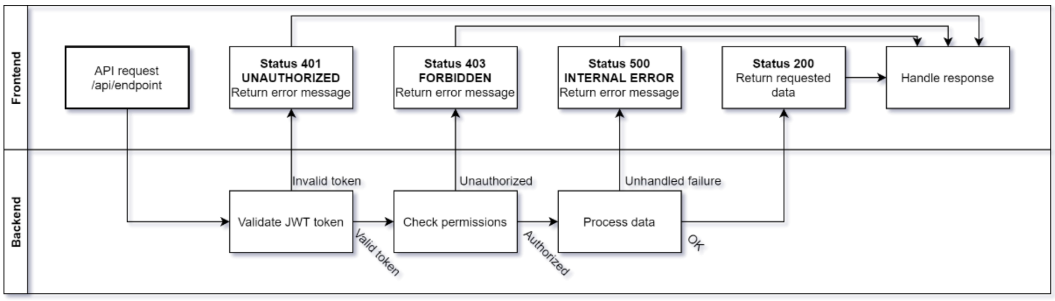

Frontend
Backend
API request /api/endpoint
Status 401 UNAUTHORIZED Return error message
Status 403 FORBIDDEN Return error message
Status 500 INTERNAL ERROR Return error message
Status 200 Return requested data
Handle response
Invalid token
Unauthorized
Unhandled failure
Validate JWT token
Check permissions
Process data
Valid token
Authorized
OK

Figure 10: Request authorization flow

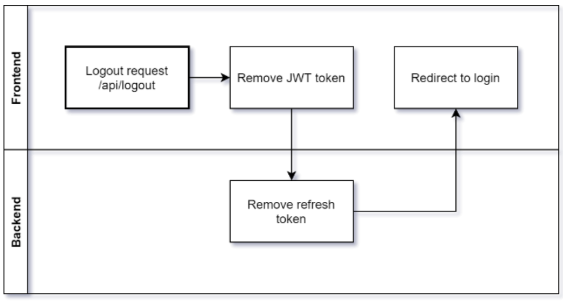


Figure 11: Logout flow

Passport is used to make sure the user who makes a REST request to the backend has the proper authorizations to do so. For example, if a supplier somehow fooled the UI layer into thinking that they have the Buyer role, which would enable the user to view a full list of their competitors, the backend would block the request and return an error message instead of exposing sensitive data to a potentially malicious party. Of course, this is the worst-case scenario where such requests could return anything.

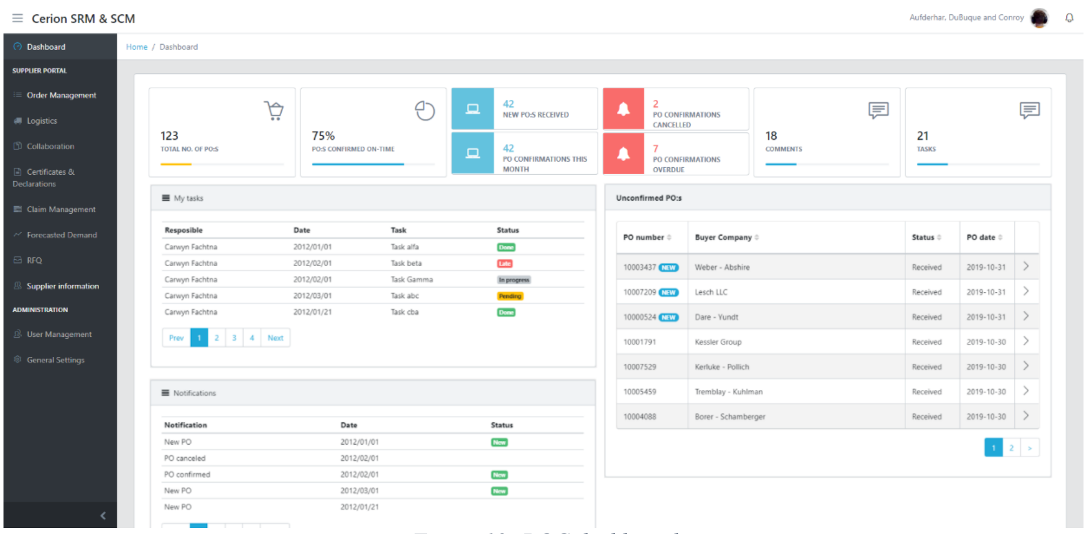
5. Evaluation and results

This chapter contains a presentation of the finished POC, and an evaluation of the work.

The finished POC consists of a front- and a backend project. The production-built, minified frontend code has the size of 18.6 MB, while the backend code resulted in 38.4 KB of files. The SQLite 3 database used for the POC was small and filled with 136 KB of generated data.

The frontend of the POC has the following layout: the top of the page contains a header section, and a navigation bar can be found on the left side of the page. The header section contains the application name, or "Cerion SRM & SCM" for the POC, as well as a user menu. The user menu is labelled by the user's company name and a user icon. In the POC, the user menu contains only a link for signing out of the site. The navigation bar contains links to different sections of the website. These links are different depending on the user's role. For example, suppliers are unable to see the SRM-related links.

The first thing the user sees when accessing the supplier portal is the dashboard page, which is depicted in Figure 12. The purpose of the dashboard is to give the user information about the state of the SCM or SRM process and any tasks that need attention. The state of the process is mainly represented by key performance indicators (KPIs), while notifications and tasks are grouped into their own lists. The KPIs are different for users with supplier and buyer roles, as the relevant numbers for each are different.



My tasks

Resposible	Date	Task	Status
Carwyn Fachtna	2012/01/01	Task alfa	Done
Carwyn Fachtna	2012/02/01	Task beta	Late
Carwyn Fachtna	2012/02/01	Task Gamma	In progress
Carwyn Fachtna	2012/03/01	Task abc	Pending
Carwyn Fachtna	2012/01/21	Task cba	Done

Notifications

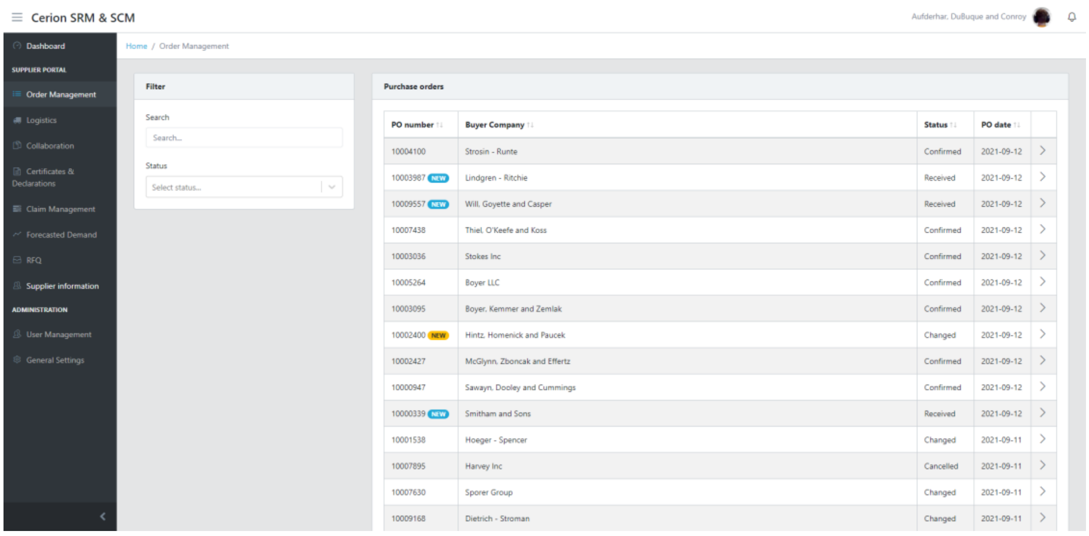
Notification	Date	Status
New PO	2012/01/01	New
PO canceled	2012/02/01	
PO confirmed	2012/02/01	New
New PO	2012/03/01	New
New PO	2012/01/21	

Unconfirmed PO:s

PO number	Buyer Company	Status	PO date	
10003437 NEW	Weber - Abshire	Received	2019-10-31	>
10007209 NEW	Lesch LLC	Received	2019-10-31	>
10000524 NEW	Dare - Yundt	Received	2019-10-31	>
10001791	Kessler Group	Received	2019-10-30	>
10007529	Kerluke - Pollich	Received	2019-10-30	>
10005459	Tremblay - Kuhlman	Received	2019-10-30	>
10004088	Borer - Schamberger	Received	2019-10-30	>

Figure 12: POC dashboard

Both suppliers and buyers can access the order management page, which contains a list of all purchase orders. The view differs slightly for each role: buyers see a list of purchase orders toward their suppliers, while suppliers see a list of purchase orders made by their buyers. This list can be filtered by free text search and status. Some purchase orders are marked with a "NEW"-label next to the PO number value to communicate that the purchase order was created recently or contains new information. This list can be seen in Figure 13 from a supplier's point of view.

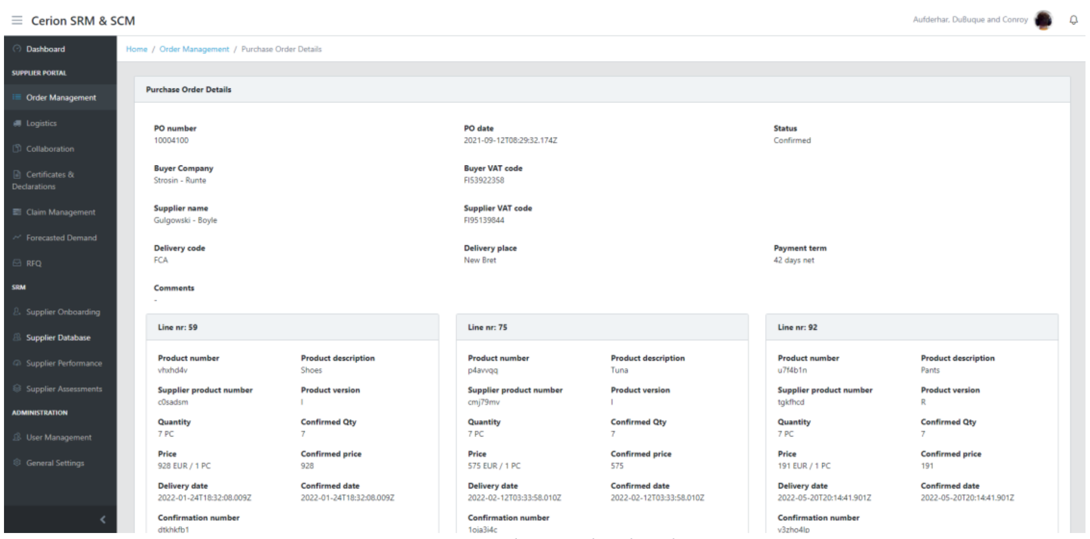


Purchase orders

PO number	Buyer Company	Status	PO date	
10004100	Strosin - Runte	Confirmed	2021-09-12	>
10003987 NEW	Lindgren - Ritchie	Received	2021-09-12	>
10009557 NEW	Will, Goyette and Casper	Received	2021-09-12	>
10007438	Thiel, O'Keefe and Koss	Confirmed	2021-09-12	>
10003036	Stokes Inc	Confirmed	2021-09-12	>
10005264	Boyer LLC	Confirmed	2021-09-12	>
10003095	Boyer, Kemmer and Zemlak	Confirmed	2021-09-12	>
10002400 NEW	Hintz, Homenick and Paucek	Changed	2021-09-12	>
10002427	McGlynn, Zboncak and Effertz	Confirmed	2021-09-12	>
10000947	Sawayn, Dooley and Cummings	Confirmed	2021-09-12	>
10000339 NEW	Smitham and Sons	Received	2021-09-12	>
10001538	Hoeger - Spencer	Changed	2021-09-11	>
10007895	Harvey Inc	Cancelled	2021-09-11	>
10007630	Sporer Group	Changed	2021-09-11	>
10009168	Dietrich - Stroman	Changed	2021-09-11	>

Figure 13: Order management page

Clicking a purchase order in the order management list takes the user to the purchase order details page. This page, which is shown in Figure 14 below, displays the full details of the purchase order, i.e., the header information and each individual item in the purchase order. If a purchase order has not been confirmed by the supplier, the quantity, and price details can be edited by either party.



Purchase Order Details

PO number
10004100

PO date
2021-09-12T08:29:32.174Z

Status
Confirmed

Buyer Company
Strosin - Runte

Buyer VAT code
FI53922358

Supplier name
Gulgowski - Boyle

Supplier VAT code
FI95139844

Delivery code
FCA

Delivery place
New Bret

Payment term
42 days net

Comments
-

Line nr: 59

Product number	Product description
vhxhd4v	Shoes
Supplier product number	**Product version**
c0sadsm	I
Quantity	**Confirmed Qty**
7 PC	7
Price	**Confirmed price**
928 EUR / 1 PC	928
Delivery date	**Confirmed date**
2022-01-24T18:32:08.009Z	2022-01-24T18:32:08.009Z
Confirmation number	
dtkhkfb1	

Line nr: 75

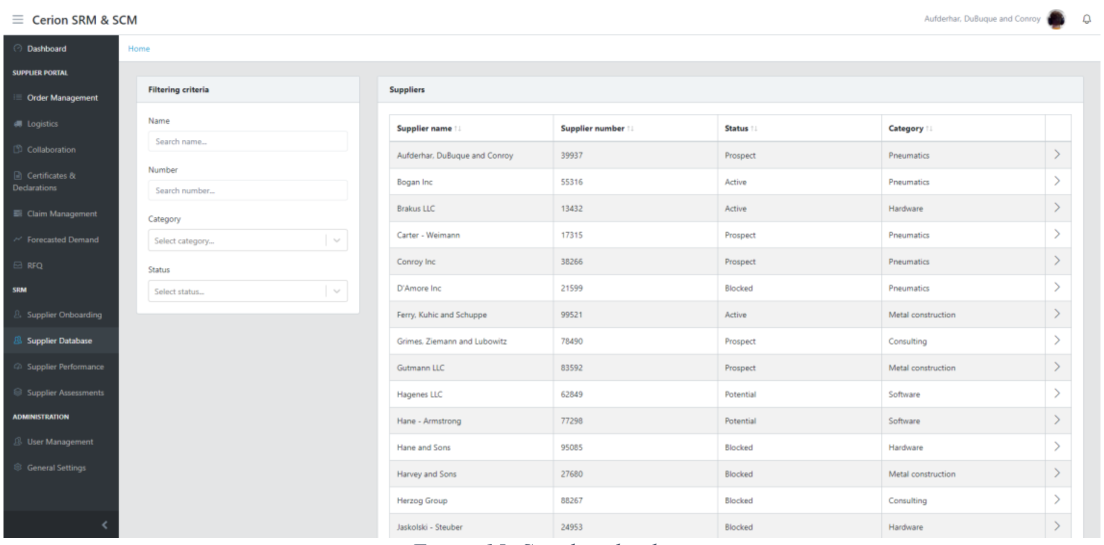
Product number	Product description
p4avvqq	Tuna
Supplier product number	**Product version**
cmj79mv	I
Quantity	**Confirmed Qty**
7 PC	7
Price	**Confirmed price**
575 EUR / 1 PC	575
Delivery date	**Confirmed date**
2022-02-12T03:33:58.010Z	2022-02-12T03:33:58.010Z
Confirmation number	
1oia3i4c	

Line nr: 92

Product number	Product description
u7f4b1n	Pants
Supplier product number	**Product version**
tgkfhcd	R
Quantity	**Confirmed Qty**
7 PC	7
Price	**Confirmed price**
191 EUR / 1 PC	191
Delivery date	**Confirmed date**
2022-05-20T20:14:41.901Z	2022-05-20T20:14:41.901Z
Confirmation number	
v3zho4lp	

Figure 14: Purchase order details page

Users with the buyer role can access the supplier database in Figure 15, which is a key feature in the SRM part of the solution. As the name suggests, this page lists all suppliers for the buyer, along with their relationship status and other details. This list can be filtered by status and category, as well as by searching for a specific name or supplier number.



Filtering criteria

Name: Search name...

Number: Search number...

Category: Select category...

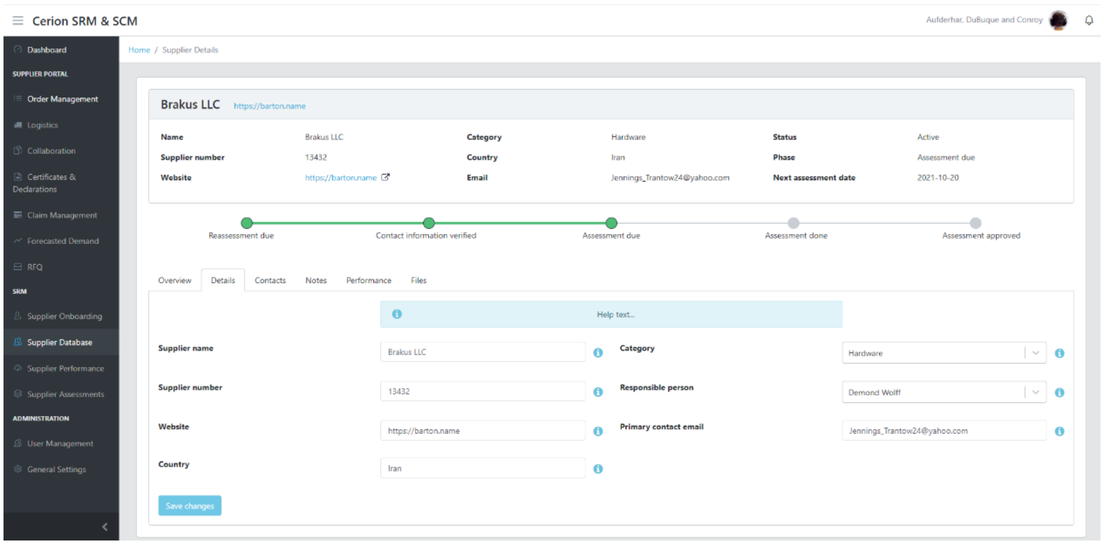
Status: Select status...

Suppliers

Supplier name	Supplier number	Status	Category	
Aufderhar, DuBuque and Conroy	39937	Prospect	Pneumatics	>
Bogan Inc	55316	Active	Pneumatics	>
Brakus LLC	13432	Active	Hardware	>
Carter - Weimann	17315	Prospect	Pneumatics	>
Conroy Inc	38266	Prospect	Pneumatics	>
D'Amore Inc	21599	Blocked	Pneumatics	>
Ferry, Kuhic and Schuppe	99521	Active	Metal construction	>
Grimes, Ziemann and Lubowitz	78490	Prospect	Consulting	>
Gutmann LLC	83592	Prospect	Metal construction	>
Hagenes LLC	62849	Potential	Software	>
Hane - Armstrong	77298	Potential	Software	>
Hane and Sons	95085	Blocked	Hardware	>
Harvey and Sons	27680	Blocked	Metal construction	>
Herzog Group	88267	Blocked	Consulting	>
Jaskolski - Steuber	24953	Blocked	Hardware	>

Figure 15: Supplier database

Figure 16 depicts the supplier details page. This page can be accessed by users with the buyer role by clicking a supplier in the supplier database, and by users with the supplier role, though they are only able to view their own supplier details page. This page lists the details of the supplier in question and displays SRM-related assessment information as a progress bar. The buyer can use this page to see an overview of the supplier's status and interact with the contact persons by leaving notes or sharing files. The supplier's details can be edited at any time, if corrections are needed.



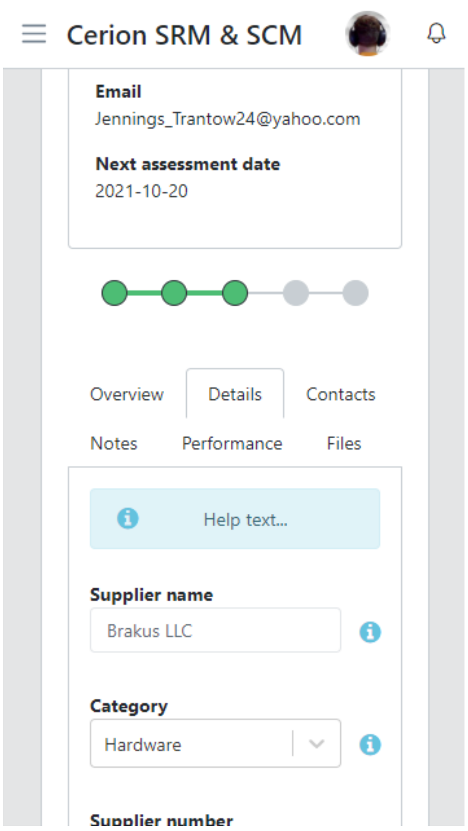
Brakus LLC https://barton.name

Name	Brakus LLC	Category	Hardware	Status	Active
Supplier number	13432	Country	Iran	Phase	Assessment due
Website	https://barton.name	Email	Jennings_Trantow24@yahoo.com	Next assessment date	2021-10-20

Supplier name	Brakus LLC	Category	Hardware
Supplier number	13432	Responsible person	Demond Wolff
Website	https://barton.name	Primary contact email	Jennings_Trantow24@yahoo.com
Country	Iran		

Figure 16: Supplier details page

Finally, the POC is built to be fully responsive so that it can scale down gracefully on displays with lower resolutions and on devices with narrower screens, such as mobile phones. Figure 17 was included as an example of this. It shows the supplier details page on a simulated mobile phone screen, at a resolution of 360×640 pixels. As can be seen, the layout remains unbroken, and the navigation bar has been hidden by default to save screen space.


Cerion SRM & SCM

Email
Jennings_Trantow24@yahoo.com

Next assessment date
2021-10-20

Supplier name
Brakus LLC

Category
Hardware

Supplier number

Figure 17: Supplier details page viewed on a mobile phone screen

The POC has been evaluated by verifying that each of the functional and non-functional requirements that were in scope for the POC worked as expected. This was done by end-to-end testing the front- and backends and verifying that the data were correctly handled and that actions had expected outcomes.

The technical design will serve as the starting point for any further development of the concept and, as such, will keep growing alongside the POC. The POC itself has been showcased by Cerion to a small number of customers who have expressed interest in the project.

6. Discussion

The main purpose of this thesis was to establish a basis on which to develop the concept of the "SCM and SRM as a Service" idea further. In that sense, this thesis has contributed exactly that to Cerion. However, there are a few things that could have been done differently in retrospect. These ideas can also serve as possible future developments within the project.

Because of its dynamically typed nature, JavaScript lets mistakes slip by easily. This has been unpleasant to handle at times in the frontend layer because of its many React components and functions that pass data to each other. Static typing would have assisted in catching a few mistakes much earlier on account of the type checking. Therefore, TypeScript could have been a better choice in the frontend layer. Working with the CoreUI Pro template went well and by using it, it became possible to focus almost entirely on the functionality instead of the user interface and experience, though it generated much additional code for the build because of its many package dependencies.

For the backend layer, however, Node.js and JavaScript were good choices, though selecting a more performant backend technology would likely be preferable when the scope of the project grows further.

Summary in Swedish – Svensk sammanfattning

Hantering av försörjningskedjan och leverantörsrelationer som ett molntjänstkoncept

Introduktion

Cerion Solutions Oy – eller Cerion – har identifierat ett behov av mjukvarulösningar inom försörjningskedjan, nämligen hantering av försörjningskedjan (SCM, Supply Chain Management) och hantering av leverantörsrelationer (SRM, Supplier Relationship Management). Enligt dem sköter flera företag dessa processer med hjälp av e-post och kalkylprogram. Cerion vill erbjuda dessa funktionaliteter som en molntjänst där flera företag använder samma programvaruinstans. Dessutom ska det vara möjligt att ansluta tjänsten till kundens egna resursplaneringssystem eller servrar. Problemet ligger i hur en design för en sådan lösning ska utvecklas, samt vad som måste beaktas för att fylla kravet om många företagsanvändare i samma instans. För att vidareutveckling ska vara möjligt måste en grund läggas för detta koncept.

Detta arbete omfattar en teknisk design samt en experimentell teknisk lösning – en ”proof of concept”-lösning (POC) – för en fungerande SCM- och SRM-webbportal med skilda användargränssnitt och serverdelar. Funktionaliteten i POC-lösningen kommer att vara begränsad eftersom avsikten är att projektet fortsätter långt efter att avhandlingen är klar. I avhandlingens kontext är den tekniska lösningen klar när alla kraven har fyllts. Den tekniska designen fokuserar på POC-lösningen, men kan innehålla detaljer för vidareutveckling som inte ingår i detta arbete.

Bakgrund

I detta kapitel introduceras grundkoncepten för en mjukvarulösning av detta slag.

Hantering av försörjningskedjan

I dagens värld är det allt vanligare att företag konkurrerar inom försörjningskedjan i stället för direkt konkurrens [1]. En förenklad modell av försörjningskedjan presenteras i Figure 1 från en tillverkares synvinkel med leverantörer och köpare på flera nivåer.

SCM handlar om att integrera och hantera affärsprocesserna i varje del av försörjningskedjan. Poängen är inte att optimera varje företag skilt för sig, men i stället att förbättra processen genom hela försörjningskedjan [2]. SCM omfattar åtta olika processer, men de relevanta processerna för POC-lösningen är orderbehandling och hantering av leverantörsrelationer, eller SRM. Orderbehandling innebär att en leverantör som använder lösningen kan behandla beställningar och fylla i detaljer vid behov. SRM behandlas som en självständig helhet i följande kapitel. De återstående processerna behandlas inte på grund av att de antingen är fysiska processer som inte kan behandlas genom en webbportal eller för att den tillhörande funktionaliteten ligger utanför projektets ramar.

Hantering av leverantörsrelationer

SRM är en process inom SCM som innebär att en köpare utvecklar och underhåller sina relationer till olika leverantörer samt mäter deras prestanda [10] [11]. Prestandan kan mätas baserat på inköpspris, pålitlig leverans, kvalité och effektivitet [10]. Olika leverantörer kräver dock olika SRM-strategier som är anpassade till sammanhanget och köparens behov [1] [12]. Processen är ömsesidigt fördelaktig för alla parter eftersom den stärker försörjningskedjan som en helhet och kan därmed förbättra företagens prestanda.

I POC-lösningen omfattar SRM att en köpare kan se sina leverantörer och förhållandets status, samt kommunicera med dem individuellt.

Molnbaserade webbtjänster

Molntjänster har blivit allt populärare eftersom de eliminerar behovet av att investera i egen serverhårdvara och mjukvara [20]. Ur avhandlingens synvinkel är tjänsterna ”Infrastructure as a Service” (IaaS) och ”Platform as a Service” (PaaS) speciellt intressanta eftersom avsikten är att köra POC-lösningen som i sin tur är en ”Software as a Service”-tjänst (SaaS).

Med IaaS menas att den fysiska hårdvaran, tillsammans med servrar, lagringsutrymme och brandväggar hyrs ut åt en molntjänstanvändare. PaaS betyder däremot att i stället för att den råa hårdvaran hyrs operativsystemet och utvecklingsverktyg ut för att hantera molninstanser. Slutligen, är SaaS ett program på molnet som en slutanvändare använder, ofta genom en webbläsare. Förhållandet mellan dessa tjänsttyper avbildas i Figure 2.

Mjukvaruarkitektur för många företagsanvändare

En arkitektur där många användare utnyttjar samma mjukvaruinstans oberoende av varandra kallas ”multi-tenant architecture” på engelska [22], eller förkortat MTA. Detta skiljer sig från program vars användare har egna mjukvaruinstanser, samt egna databaser och hårdvara. I denna kontext betyder ”användare” en organisation eller grupp av personer som har rätt att se samma data [21].

Det finns tre grundläggande MTA-modeller [24] som kan ses i Figure 4. Den första bilden visar en modell vars användare har egna, separata databaser. Denna typs modell är dyr att underhålla [26], men alla data är väl isolerade. Den andra modellen visar en arkitektur med en databas som delas av flera användare och data måste hämtas med filter så att användare endast kommer åt data som tillhör dem. Ansvaret att isolera data ligger i detta fall hos mjukvaruutvecklaren [25], men modellen är billigare att underhålla. Den tredje modellen har också endast en delad databas, men användarna kan endast komma åt tabeller från scheman som tillhör dem. Modeller med bättre isolerad data är huvudsakligen säkrare, men storleken är svårare att trappa upp på dem [22].

Design

I detta kapitel föreslås en arkitektur för POC-lösningen. En jämförelse av olika molntjänster presenteras, och lösningens distribueringsplan beslutas.

Skapandet av arkitekturen

Eftersom lösningen är ett SaaS-projekt kommer arkitekturen logiskt att vara av typen MTA [22]. Eftersom det ska vara möjligt att kommunicera både med kundens egna resursplaneringssystem och lösningens egna serverprogram kommer MTA-modellen i POC-lösningen att vara en kombination av den första och den andra MTA-modellen i Figure 4. Resultatet visas i Figure 5, där varje användare är ansluten till båda datakällorna. Märk väl att den andra modellen i Figure 4 används enbart för POC-lösningen, och en mer säker implementering används när projektet växer i storlek.

Jämförelse av molntjänstleverantörer

Eftersom POC-lösningen är delad i användargränssnittet och serverdelen, måste en molntjänst som kan serva båda delarna hittas. Molntjänsten måste vara lämplig till lösningens ändamål samt höra till Cerions intresseområde. De tre molntjänstleverantörerna som jämförs är Amazon Web Services, Microsoft Azure och Google Cloud Platform, som valdes på grund av att de är mycket populära inom molntjänstmarknaden.

Alla tre molntjänstleverantörer erbjuder mycket liknande tjänster som är lämpliga för POC-lösningen. Trots det kan prestandan för de olika leverantörerna vara mycket olika [37]. Enligt ett antal prestandajämförelser av Muhammad-Bello och Aritsugi [37] [39] framkommer det att prestandan på alla tre molntjänstleverantörer är mycket liknande, men Microsoft Azure har det bästa förhållandet mellan kostnad och prestanda av dem i allmänhet. En annan viktig faktor är Cerions åsikt. Cerion och Microsoft har redan ett partnerskap, och att använda Microsoft Azure skulle gynna båda parterna på lång sikt. Med tanke på alla dessa faktorer är Microsoft Azure det bästa valet av de tre alternativen. Den kommer att användas för lösningens båda delar.

Distribuering av lösningen

För att få projektet på internet måste distribueringsprocessen för användargränssnittet respektive serverdelen beslutas.

Eftersom det ska vara möjligt att öka prestandan på serverdelen tillsammans med ett växande antal användare måste också distribueringssättet tillåta det. En Azure App Service -instans kan användas för att lätt hantera bygg- och distribueringsprocessen med att överföra koden till den. Detta är idealt för distribueringen av användargränssnittet, men eftersom serverdelen måste kunna kommunicera med olika servrar från kundernas sida måste en alternativ teknologi hittas.

Docker är ett program som låter användaren skapa så kallade containrar som isolerar ett program från omgivningen den körs i. De är effektivare än motsvarande virtuella maskiner [42], och skillnaden mellan dessa teknologier klargörs i Figure 6. Docker-containrar kan köras på tjänsten Azure Kubernetes Service. Den bygger på Kubernetes-teknologi som kan beskrivas som ett kluster av containrar. Sådana kluster är ideala för MTA-baserade lösningar med komplexa nätverk [45] eftersom det är lätt att koppla den till olika omgivningar. Detta lämpar sig väl till serverdelens ändamål, och den kommer därmed att distribueras på Azure Kubernetes Service i en Docker-container.

Praktisk implementering

Den praktiska implementeringen består av den tekniska designen och POC-lösningen.

Teknisk design

Detta kapitel innehåller projektets funktionella och icke-funktionella krav, samt lösningens arkitektur och databasdesign för den delade databasen. En del av den tekniska designen implementeras inte i POC-lösningen.

Till de funktionella kraven hör ett antal funktioner för SCM och SRM. Till SRM-uppgifterna hör en leverantörsdatabas, ett sätt att mäta prestandan på enskilda leverantörer, och funktionalitet för att möjliggöra förvärvande av nya leverantörer. SCM-funktionaliteten omfattar offertförfrågningar, prognos på efterfrågan, kravhantering, certifikat, deklarationer, logistik och orderbehandling. Till webbportalens funktionella krav hör bland annat en startsida med statistik, ett navigeringsfält som tillåter användaren att navigera till olika sidor, samt möjligheten att logga ut från webbportalen i varje vy. Sidan måste också vara användbar på enheter med små skärmar, till exempel mobiltelefoner. Dessutom ska hela användargränssnittet vara tillgängligt för personer med funktionsnedsättning. Detta innebär till exempel att webbsidan ska vara användbar enbart med tangentbordsnavigering eller med hjälp av en skärmläsare. Användargränssnittet och serversidan, samt serversidan och kundens resursplaneringssystem ska kommunicera med varandra med hjälp av REST API-gränssnitt. De sista funktionella kraven berör användarrättigheter. Användare i leverantör- eller köparroll ser olika vyer i webbportalen, men det ska också vara möjligt för en användare att ha tillgång till både SRM- och SCM -funktionalitet gentemot sina egna köpare eller leverantörer.

Till de icke-funktionella kraven hör till exempel att webbsidan ska vara snabb och lätt att använda. Den ska inte frysa eller krascha på vare sig användargränssnittet eller serverdelen. Serverdelen ska byggas så att olika användare inte kan komma åt andra användares data. Båda delarna ska programmeras på ett sätt som underlättar senare underhåll. Detta kan nås med att prioritera återanvändbara komponenter och med att minska på mängden duplicerad kod.

Webbportalens arkitektur avbildas i Figure 7 som visar hur alla olika interna och externa system kommunicerar med varandra. Webbportalens alla anrop från användargränssnittet kallas genom serverdelen som är kopplad till kundernas resursplaneringssystem med hjälp av API-gränssnitt. En huvudidé med konceptet är att det ska vara möjligt att bygga på serverdelen med ytterligare funktionella moduler som användarna kan köpa. Om en kund vill ta i bruk webbportalen måste de stöda API anrop från programmets serverdel. SAP och Microsoft Dynamics har planerats vara kompatibla, men stöd för fler resursplaneringssystem är också möjligt i framtiden.

Schemat för den delade databasen för POC-lösningen kan ses i Figure 8. Den innehåller alla tabeller och relationer som krävs för att uppfylla POC-lösningens krav. När fler funktioner för webbportalen planeras och implementeras kommer naturligtvis databasdesignen också att växa. Planen är att använda Azure SQL Database som databasmotor eftersom den stöder MTA-design [25]. I POC-lösningen används däremot en lokal instans av SQLite 3 för att underlätta snabb utveckling.

Användargränssnittet

Eftersom det krävs att webbportalens användargränssnitt ska vara tillgängligt för personer med funktionsnedsättning och att koden ska vara lätt att underhålla, har hela användargränssnittet byggts med det JavaScript-baserade ramverket React. Fokuset i React ligger på komponentbaserad design, vars ändamål är att minimera duplicerad kod. Ett layoutbibliotek – CoreUI Pro – användes för att programmeringsprocessen skulle vara smidigare med funktionaliteten som högsta prioritet.

Tillgängligheten på små skärmar uppnåddes med Bootstrap CSS-klasser och anpassad, egen CSS på strategiska platser. Övriga tillgänglighetsfrågor har skötts med att noggrant välja färger med tillräckligt hög kontrast och med lämplig HTML 5-struktur.

Blanketter hanteras på ett så användarvänligt sätt som möjligt. Obligatoriska fält markeras med en asterisk (*) och ogiltiga inmatningsvärden visas med valideringsmeddelanden. Meddelanden från alla REST-anrop visas på skärmen i en färgkodad ruta.

Serverdelen

På grund av att användargränssnittet inte är beroende av teknologin som används i serverdelen kan nästan vilken som helst teknologi användas. För POC-lösningens ändamål valdes Node.js som serverteknologi eftersom programmeringsspråket är det samma som för användargränssnittet, nämligen JavaScript. Express-ramverket används för själva REST API-funktionaliteten. För att hantera data från databasen används ett ORM-verktyg (Object Relational Mapping) som möjliggör utförandet av databasoperationer med JavaScript-kod och JSON-objekt. Eftersom POC-lösningen och projektet som en helhet använder SQL-baserade databaser används Sequelize-biblioteket som ORM-verktyg. Sequelize stöder många olika SQL-dialekter och kan hantera dem med samma kod.

Autentisering och säkerhet

Säkerheten behandlas på projektets båda delar. De flesta säkerhetsåtgärderna behandlas i serverdelen, men användargränssnittet måste också förhindra missbruk av systemet.

All data som användaren matar in i användargränssnittet valideras, vilket också syns på blanketten och hindrar slutanvändaren från att skicka i väg det. Innan validerad data skickas till serversidan kontrolleras det ännu en gång. När användaren loggar in i webbportalen med rätt e-postadress och lösenord skickar serverdelen en ”JSON Web Token” (JWT), alltså en kodad textsträng med autentiseringsinformation, det vill säga information om vilka vyer och funktioner användaren har tillstånd att komma åt. Varje HTTP-anrop mot serverdelen görs genom en specialbyggd klass som använder JavaScripts inbyggda funktioner. Alla anrop till serverdelen förutom inloggningsanropet måste innehålla JWT-strängen.

I servedelen ska alla inkommande API-anrop först autentiseras och auktoriseras. JavaScript-biblioteket Passport sköter om dessa uppgifter och kan behandla flera olika autentiseringsstrategier, bland annat JWT som används i projektet. För att ett anrop ska accepteras och att serverdelen kan skicka tillbaka data till användargränssnittet måste JWT-strängen vara giltig. Användaren som gjorde anropet måste dessutom ha rättigheter att komma åt funktionen. JWT valdes som den främsta autentiseringsmetoden på grund av att den inte är beroende av externa tjänster. JWT-strängarna lagras i webbläsarens minne endast så länge sidan är öppen, och är endast giltiga i 15 minuter. Att lagra JWT-strängen på användarens dator ökar risken för olika attacker [46]. För att användaren inte ska behöva logga in varje gång sidan öppnas om eller om giltighetsperioden är över, implementerades en ytterligare långlivad JWT som används för att fråga efter en ny autentiseringssträng. Hela JWT-flödet avbildas i Figure 9, Figure 10 och Figure 11.

Resultat

Den slutliga POC-lösningen består av skilda användargränssnitt- och serverprojekt. Koden för användargränssnittet har storleken 18,6 MB, medan serverdelen är 38,4 KB stor. SQLite 3-databasen som användes i POC-lösningen var liten och fylldes med 136 KB genererad data.

Bilder på användargränssnittet finns i Figure 12, Figure 13, Figure 14, Figure 15, Figure 16 samt Figure 17. Varje krav har analyserats och testats, och POC-lösningen uppfyller kraven som hörde till den. Den har visats till ett antal kunder som har visat intresse för projektet. Den tekniska designen ska användas som en utgångspunkt för vidare utveckling av projektet.

Avslutande diskussion

Avhandlingens huvudsyfte var att lägga en grund för att kunna vidareutveckla ”SCM and SRM as a Service”-konceptet. Detta är exakt vad som bidragits till Cerion. Det finns dock vissa saker som kunde ha gjorts annorlunda. Dessa idéer kan också användas som möjliga framtida utvecklingsmål inom projektet.

Eftersom JavaScript har dynamiska typer var det svårt att hitta misstag i användargränssnittets kod eftersom React bygger på komponenter som överför data till varandra i långa kedjor. Med statiska typer skulle dessa misstag ha hittats tidigare, vilket är varför TypeScript kunde ha varit ett bättre val. Att arbeta med CoreUI Pro-biblioteket var bekvämt eftersom det var möjligt att fokusera på funktionaliteten i stället för det visuella, men den genererade en massa överloppskod på grund av att den var beroende av många andra kodpaket. För serverdelen var Node.js och JavaScript bra val, men det kunde vara en bra idé att välja en mer effektiv serverteknik när projektet fortsätter växa.

Bibliography

[1] D. M. Lambert, Supply Chain Management: Processes, Partnerships, Performance, Supply Chain Management Institute, 2014.

[2] J. Heikkilä, "From supply to demand chain management: efficiency and customer satisfaction," *Journal of Operations Management,* vol. 20, no. 6, pp. 747-767, 2002.

[3] X. Cao, H. Xia and H. Wen, "The Customer Relationship Management Model Based on Optimal Control," in *2010 International Conference on Challenges in Environmental Science and Computer Engineering*, Wuhan, 2010.

[4] Y. A. Bolumole, A. M. Knemeyer and D. M. Lambert, "The Customer Service Management Process," *The International Journal of Logistics Management,* vol. 14, no. 2, pp. 15-31, 2003.

[5] D. Thomason, "Strategic, tactical, operational," *Manufacturing Engineer,* vol. 83, no. 3, pp. 34-37, June-July 2004.

[6] Q. C. N. Sie, "Order Management of Supply Chain: A Case Study in X-Fab Sarawak Sdn Bhd," in *2011 International Conference on Information Management, Innovation Management and Industrial Engineering*, Shenzhen, 2011.

[7] T. J. Goldsby and S. J. García-Dastugue, "The Manufacturing Flow Management Process," *The International Journal of Logistics Management,* vol. 14, no. 2, pp. 33-52, 2003.

[8] D. S. Rogers, D. M. Lambert and A. M. Knemeyer, "The Product Development and Commercialization Process," *The International Journal of Logistics Management,* vol. 15, no. 1, pp. 43-56, 2004.

[9] D. S. Rogers, D. M. Lambert, K. L. Croxton and S. J. García-Dastugue, "The Returns Management Process," *The International Journal of Logistics Management,* vol. 13, no. 2, pp. 1-18, 2002.

[10] Institute for Supply Management, "Supplier Relationship Management Insights," [Online]. Available: https://www.instituteforsupplymanagement.org/content.cfm?ItemNumber=20233&SSO=1. [Accessed 3 April 2020].

[11] D. M. Lambert and M. A. Schwieterman, "Supplier relationship management as a macro business process," *Supply Chain Management: An International Journal,* vol. 17, no. 3, pp. 337-352, 2012.

[12] A. Das, R. Narasimhan and S. Talluri, "Supplier integration—Finding an optimal configuration," *Journal of Operations Management,* vol. 24, no. 5, pp. 563-582, 2006.

[13] M. Swink, R. Narasimhan and C. Wang, "Managing beyond the factory walls: Effects of four types of strategic integration on manufacturing plant performance," *Journal of Operations Management,* vol. 25, no. 1, pp. 148-164, 2007.

[14] P. J. Singh and D. Power, "The nature and effectiveness of collaboration between firms, their customers and suppliers: a supply chain perspective," *Supply Chain Management: An International Journal,* vol. 14, no. 3, pp. 189-200, 2009.

[15] B. B. Flynn, B. Huo and X. Zhao, "The impact of supply chain integration on performance: A contingency and configuration approach," *Journal of Operations Management,* vol. 28, no. 1, pp. 58-71, 2010.

[16] M. G. Enz and D. M. Lambert, "Using cross-functional, cross-firm teams to co-create value: The role of financial measures," *Industrial Marketing Management,* vol. 41, no. 3, pp. 495-507, 2012.

[17] D. M. Lambert and R. Burduroglu, "Measuring and Selling the Value of Logistics," *The International Journal of Logistics Management,* vol. 11, no. 1, pp. 1-18, 2000.

[18] A. R. Zablah, D. N. Bellenger and W. J. Johnston, "An evaluation of divergent perspectives on customer relationship management: Towards a common understanding of an emerging phenomenon," *Industrial Marketing Management,* vol. 33, no. 6, pp. 475-489, 2004.

[19] D. M. Lambert and T. L. Pohlen, "Supply Chain Metrics," *The International Journal of Logistics Management,* vol. 12, no. 1, pp. 1-19, 2001.

[20] IBM Cloud Education, "Benefits of Cloud Computing," IBM, 10 October 2018. [Online]. Available: https://www.ibm.com/cloud/learn/benefits-of-cloud-computing. [Accessed 22 August 2021].

[21] R. Krebs, C. Momm and S. Kounev, "Architectural Concerns in Multi-Tenant SaaS Applications," in *Proceedings of the 2nd International Conference on Cloud Computing and Services Science - Volume 1: CLOSER*, Porto, 2012.

[22] IBM Cloud Education, "Multi-Tenant," 20 January 2020. [Online]. Available: https://www.ibm.com/cloud/learn/multi-tenant. [Accessed 27 May 2020].

[23] C. Brook, "SaaS: Single Tenant vs Multi-Tenant - What's the Difference?," Digital Guardian, 1 December 2020. [Online]. Available: https://digitalguardian.com/blog/saas-single-tenant-vs-multi-tenant-whats-difference. [Accessed 6 June 2021].

[24] Z. H. Wang, C. J. Guo, B. Gao, W. Sun, Z. Zhang and W. H. An, "A Study and Performance Evaluation of the Multi-Tenant Data Tier Design Patterns for Service Oriented Computing," *2008 IEEE International Conference on e-Business Engineering,* pp. 94-101, 2008.

[25] Microsoft, "Multi-tenant SaaS database tenancy patterns," Microsoft, 25 January 2019. [Online]. Available: https://docs.microsoft.com/en-

us/azure/azure-sql/database/saas-tenancy-app-design-patterns. [Accessed 6 June 2021].

[26] Microsoft, "Elastic pools help you manage and scale multiple databases in Azure SQL Database," Microsoft, 9 December 2020. [Online]. Available: https://docs.microsoft.com/en-us/azure/azure-sql/database/elastic-pool-overview. [Accessed 6 June 2021].

[27] A. Z. and S. A., "Three Database Architectures for a Multi-Tenant Rails-Based SaaS App," RubyGarage, 4 January 2020. [Online]. Available: https://rubygarage.org/blog/three-database-architectures-for-a-multi-tenant-rails-based-saas-app. [Accessed 6 June 2021].

[28] T. Brazelton, "The Bad Multi-Tenant SAAS Pattern," DEV Community, 15 July 2020. [Online]. Available: https://dev.to/mrbrazel/the-bad-multi-tenant-saas-pattern-20b4. [Accessed 6 June 2021].

[29] M. Hall, "Amazon.com," Encyclopedia Britannica, 9 April 2020. [Online]. Available: https://www.britannica.com/topic/Amazoncom. [Accessed 6 June 2021].

[30] C. Harvey, "AWS vs Azure vs Google Cloud: 2021 Cloud Platform Comparison," Datamation, 22 October 2020. [Online]. Available: https://www.datamation.com/cloud/aws-vs-azure-vs-google-2020-cloud-comparison/. [Accessed 6 June 2021].

[31] Synergy Research Group, "Incremental Growth in Cloud Spending Hits a New High while Amazon and Microsoft Maintain a Clear Lead," Synergy Research Group, 4 February 2020. [Online]. Available: https://www.srgresearch.com/articles/incremental-growth-cloud-spending-hits-new-high-while-amazon-and-microsoft-maintain-clear-lead-reno-nv-february-4-2020. [Accessed 6 June 2021].

[32] Amazon Web Services, Inc., "Security and Compliance," Amazon Web Services, Inc., 12 April 2021. [Online]. Available:

https://docs.aws.amazon.com/whitepapers/latest/aws-overview/security-and-compliance.html. [Accessed 6 June 2021].

[33] K. Dent, "AWS vs Azure vs Google: The battle for cloud supremacy," Jefferson Frank, [Online]. Available: https://www.jeffersonfrank.com/insights/aws-vs-azure-vs-google-cloud-provider-comparison. [Accessed 6 June 2021].

[34] Amazon Web Services, Inc., "Amazon Web Services Cloud Platform," 12 April 2021. [Online]. Available: https://docs.aws.amazon.com/whitepapers/latest/aws-overview/amazon-web-services-cloud-platform.html. [Accessed 6 June 2021].

[35] G. P. Zachary and M. Hall, "Microsoft Corporation," Encyclopedia Britannica, 12 November 2020. [Online]. Available: https://www.britannica.com/topic/Microsoft-Corporation. [Accessed 6 June 2021].

[36] W. L. Hosch and M. Hall, "Google," Encyclopedia Britannica, 11 May 2020. [Online]. Available: https://www.britannica.com/topic/Google-Inc. [Accessed 6 June 2021].

[37] B. L. Muhammad-Bello and M. Aritsugi, "TCloud: A Transparent Framework for Public Cloud Service Comparison," *2016 IEEE/ACM 9th International Conference on Utility and Cloud Computing (UCC),* pp. 228-233, 2016.

[38] C. Binnig, D. Kossmann, T. Kraska and S. Loesing, "How is the Weather tomorrow? Towards a Benchmark for the Cloud," in *Proceedings of the 2nd International Workshop on Testing Database Systems*, Providence, 2009.

[39] B. L. Muhammad-Bello and M. Aritsugi, "A transparent approach to performance analysis and comparison of infrastructure as a service providers," *Computers & Electrical Engineering,* vol. 69, pp. 317-333, 2018.

[40] Y. El-Khamra, H. Kim, S. Jha and M. Parashar, "Exploring the Performance Fluctuations of HPC Workloads on Clouds," *2010 IEEE Second International Conference on Cloud Computing Technology and Science,* pp. 383-387, 2010.

[41] R. Bauer, "What's the Diff: VMs vs Containers," Backblaze, 28 June 2018. [Online]. Available: https://www.backblaze.com/blog/vm-vs-containers/. [Accessed 6 June 2021].

[42] R. R. Yadav, E. T. G. Sousa and G. R. A. Callou, "Performance Comparison Between Virtual Machines And Docker Containers," *IEEE Latin America Transactions,* vol. 16, no. 8, pp. 2282-2288, 2018.

[43] L. Hasija, "Azure Kubernetes, Service Fabric and App Service compared," Coder's Bistro, 5 January 2019. [Online]. Available: https://www.codersbistro.com/blog/aks-service-fabric-and-app-service-compared/. [Accessed 6 June 2021].

[44] K. Lane, "Kubernetes vs. Docker: What Does it Really Mean?," Sumo Logic, 3 September 2020. [Online]. Available: https://www.sumologic.com/blog/kubernetes-vs-docker/. [Accessed 6 June 2021].

[45] D. Henry, "Azure Container Orchestration 101: Azure Web Apps vs ACI vs AKS," DragonSpears, 11 December 2019. [Online]. Available: https://www.dragonspears.com/blog/azure-container-orchestration-101-azure-web-apps-vs-aci-vs-aks. [Accessed 6 June 2021].

[46] T. Gopal and V. , "The Ultimate Guide to handling JWTs on frontend clients (GraphQL)," Hasura Inc., 9 September 2019. [Online]. Available: https://hasura.io/blog/best-practices-of-using-jwt-with-graphql/. [Accessed 4 July 2021].