Arsenii Panov

SUBSCRIPTION AND PAYMENT SYSTEMS FOR SAAS APPLICATIONS

Bachelor's thesis

Bachelor of Engineering

Degree Programme in Information Technology

2022

South-Eastern Finland
University of Applied Sciences

South-Eastern Finland University of Applied Sciences

Degree title	Bachelor of Engineering
Author(s)	Arsenii Panov
Thesis title	Subscription and payment systems for SaaS applications: A study case
Commissioned by	Mindhive Oy
Year	2022
Pages	39 pages
Supervisor(s)	Timo Mynttinen

ABSTRACT

Nowadays even small companies can create competitive software-based products. One of the best inventions which helps them is the ability to integrate multiple third-party services into digital products being created. These integrations between services take place through well-documented APIs, making it simpler for developers to implement high-quality services in a short period of time. In this thesis I present a study case of a SaaS digital product which benefits from the Stripe API to process online payments and user subscriptions.

In the theoretical part, the concept of Software as a Service was presented, as well as its most common business models and system architecture. Online payments and transactions were introduced, together with the elements that guarantee their security. The theoretical part was followed by an introduction to modern regulation and taxation for payment operations. Lastly, it presents important aspects of the design of user interfaces and user experience for subscription-based SaaS software, which are very important for acquiring paid users and increasing the revenue of these systems.

The practical part of the thesis presents the real-world study-case developed with Mindhive Oy, their client case, the software developed and its users, as well as the core of the business, the system architecture, and the front-end user interfaces. The practical implementation of the backend system, responsible for connecting the software and the third-party payment and subscription service provider, were described in detail.

The last part of the thesis brought the discussion of the main results and findings during the execution of this work, what could have been done differently, and what are the next necessary steps for the continuation of the bigger project.

Keywords: Software-as-a-Service, Online Payments, Subscription Plans, React

CONTENTS

1 INTRODUCTION

Software is everywhere: phones, TVs, cars, vending machines, even coffee makers now have software-driven features. The technology has been here for a while, but finally we are approaching the time where even small and mid-sized companies are transforming their business by digitalization and the creation of customized solutions. Modern software is not exclusive to big companies anymore. Nowadays companies, from large to small, can now provide their clients with modern and complex digital services.

Technology-wise, we see more and more applications that solve specific business needs by operating directly and completely from the cloud. Users have access to complex operations directly as online services available directly on their phones; no need to install software on computers that operate offline, or even download the apps. Services are available directly from the browsers. The technological leap of these cloud systems has reflected directly on new business models. For example, Software as a Service (SaaS) has exploded, especially after the pandemic. In this model, users can hire services on demand, pay by quantity or as you go, either monthly, annually or through hybrid models. Big players such as Netflix, Spotify, and Amazon prime have contributed to spread SaaS and subscriptions as the new standard to which users are used to hire software. This caused even traditional software to change their business towards this direction, such as the cases of Microsoft Office, Photoshop, among other traditional offline, installed directly on the computer software. Therefore, using such models is not a complete novelty for the users anymore, which allows any-sized companies to offer SaaS models for their customers as the standard format.

Before selling software and services, of course, companies need to create it. Agile development models, serverless and scalable solutions, and the ability to offer world-wide services allow even small teams to quickly create highly competitive products. The option of using integrations with other services, such

as offering online payments, secure user authentication and management, tools powered by artificial intelligence, and many other complex tasks are available for any developer through well-documented and simple APIs. The development technology has reached a place where great user experience can be achieved even for proof of concept and small software projects. Companies can thus release their software and services already at initial stages, being capable of acquiring a paying client base, raising money and funding for further development, as well as making their name in the market. This type of scalable software creates the opportunity for local players to become world-wide competitors.

My thesis is done in the context presented above. I present a study case that allows for a small company to provide a SaaS application for their users. The goal is to build generic solutions that can be used across the app created for a real business case from a client of the Mindhive Oy. While approaching the theoretical part as general as possible, the practical outcome of this work is aimed to be fully implemented for a working business solution.

In the remainder of this document, I present both the theory and practical part describing my work to implement a subscriptions system, which allows for a SaaS model that can scale to millions of users.

In the chapter 1 below I present the theoretical parts relevant to the topics discussed in this thesis. In chapter 2 I bring the practical part and the detailed technical report describing the hands-on implementation of the project. Lastly, in chapter 3, I present the discussion, main learnings and conclusions of my thesis.

2 THEORY OVERVIEW

In this chapter, I present the core information related to the operational process of creating a SaaS-model type of software, starting from what the service is and its technical details. The section emphasizes how modern digital payment operations work for businesses adopting such models, and how system's architecture and UI/UX design should be structured.

2.1 Software as a service

The key to understand how the present applications and software solutions are delivered directly from the cloud relies in understanding the Software-as-a-service models (Turner 2020). In short, with this licensing and delivery model, users gain access to the services via the Internet without needing to install and manage complex software and/or hardware applications. Nowadays, many companies use this model, including big players such as Netflix, Google, Zoom and others.

The benefits of this model for the customer is in the offered simplicity and flexibility: users just need to pay, and the services are directly and promptly accessible; users also pay as they go (e.g., monthly fees), often times only for the services they hired (e.g., using dynamic pricing models). The advantages for companies developing technology with such an approach is that the software is completely available on the cloud. The cloud servers are a controlled and scalable environment in which the code runs an environment dedicated to that, not dependent on customer hardware, local configurations, or specific versions and installations. Software updates are thus smoother, allowing for better continuous development and continuous integration practices. This allows companies to constantly improve their services, to constantly offer more value to their clients, and therefore constantly hook users to pay for their software.

Most SaaS services rely on a business strategy that gives users some level of free software access, aimed at converting them to paying customers as soon as possible. There are surely different models in which users do not have any level

of access before paying, but those are usually very niche specific products, and definitely not the most common strategy. There are two main ways to sell SaaS products and convert from free to paying users on the go: the low-touch sales model and the high-touch sales model (McKenzie, n.d.):

The core idea of the Low-touch model is to minimize human to human interactions (Ndukwu, 2021). The focus is on making conversions without a sales-dedicated operator. Such products usually offer their users a free trial version of complete services. The idea is to show users the full potential and value they get from using the services and advanced features. The assumption is that once users have tasted the best experience, they are reluctant to downgrade to free (and more limited) versions. The logic is to hook the customers from the great experience perspective. In this type of SaaS, a successful conversion happens at the end of the trial period, when users opt to pay for the services instead of down scaling to free versions. It is also possible to downgrade for free versions, and then the software will offer the option to upgrade again on many relevant places (e.g., when a user tries to utilize some key features, which are available only for paying users).

In reverse High-touch model, the idea is to communicate with the users early and actively (High Touch Business Model…, 2022). That kind of system usually has a big sales and customer support team which should help to adapt and customize the software to the client. The best of these models is that users will convert early and in quantity (for example, when big companies purchase licenses for all their employees).

Other common cases are hybrid models, with the low-touch used as a strategy to convert e.g., individual users, and the high-touch to convert enterprise and customized plans. Usually individual users can acquire their trials or standard paid plans and immediately start using the application, while enterprises should contact the service providers in order to define which would be the best pricing model for their situation. Such high-touch cases are interesting for the bigger number of users, as they usually involve huge discounts for the prices of the

services to be acquired at once for many users (e.g., companies and universities).

All these models are business decisions made not much by the development teams, but rather by the product owners, based on the company strategies to achieve their desired business goals.

2.2 Dynamic pricing model for services and features

As stated early, the pricing models involve strategic decisions from the business running the software. The way these pricing models are incorporated in the software is complex, and depends on many variables, such as how many users the app is targeted at, for how long they will be clients, how frequently they will use the software, and many other aspects. That is an entire field in itself, and the focus of this thesis is on the technical implementation of one of multiple options that would be suitable for the case at hand. I will thus focus solely on the subscriptions models available for the users (Pricing models for recurring billing…, n.d.). Nowadays, there are plenty of different pricing models available, with different logic of implementation and options for the payment of services (Law, 2021). Some of them are more commonly used because of its simplicity of usage, and others are chosen because of its complexity.

Many modern services offer a Flat-rated Good-Better-Best pricing model (Figure 1), where the customer has a choice to pick different subscriptions, and each one of them are more expensive and have more features available. The idea is to provide incremental value for the users, and charge it accordingly. This solution is technically simple to implement - the strategy the seller needs to decide is the best way to define the logic of the amount of features per choice of plan. Usually the logic goes similarly to the french fries strategy at McDonald's chain: The small fries option is so close to the medium option that most users end up deciding for the medium.

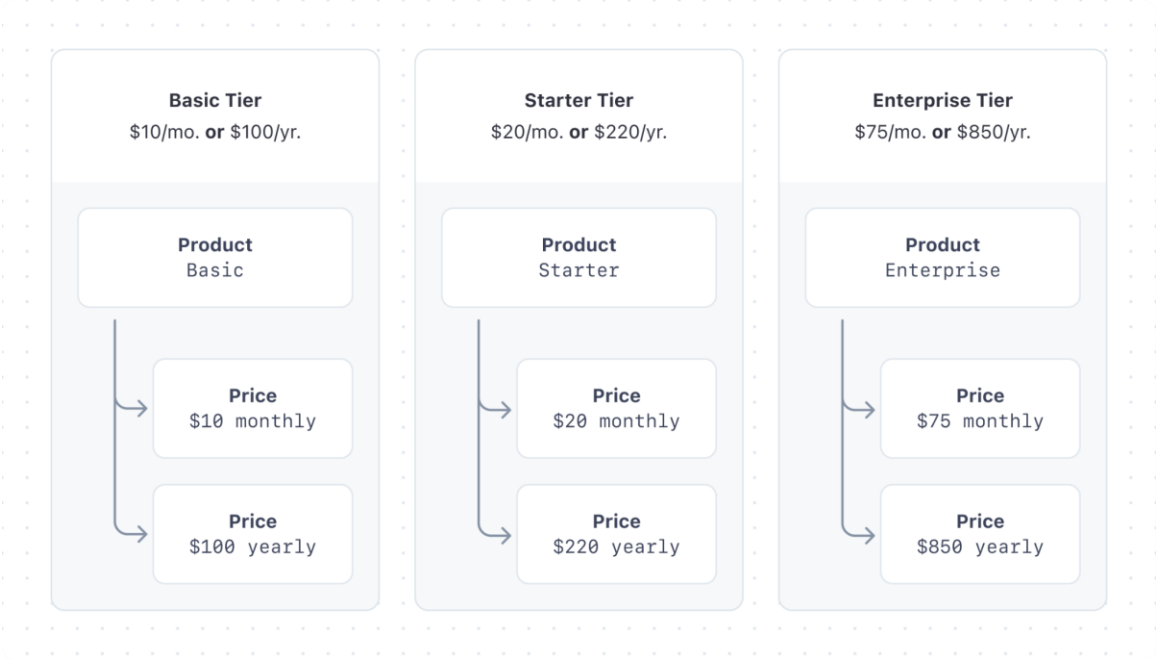

Figure 1. Example of flat-rated (Good-Better-Best) pricing model. (Users pay a fixed amount over a given period of time to have access to different numbers of features.)

The real-world might be more complex than that. In these cases, if we want our services, for example, to be dependent on the amount of users using it, it would be better to choose a Per-seat pricing model (Figure 2). In that strategy the service price is calculated based on the number of user places the customer decides to hire. Users can define it initially and adjust for their needs on the go, by either adding more seats, or reducing its number.

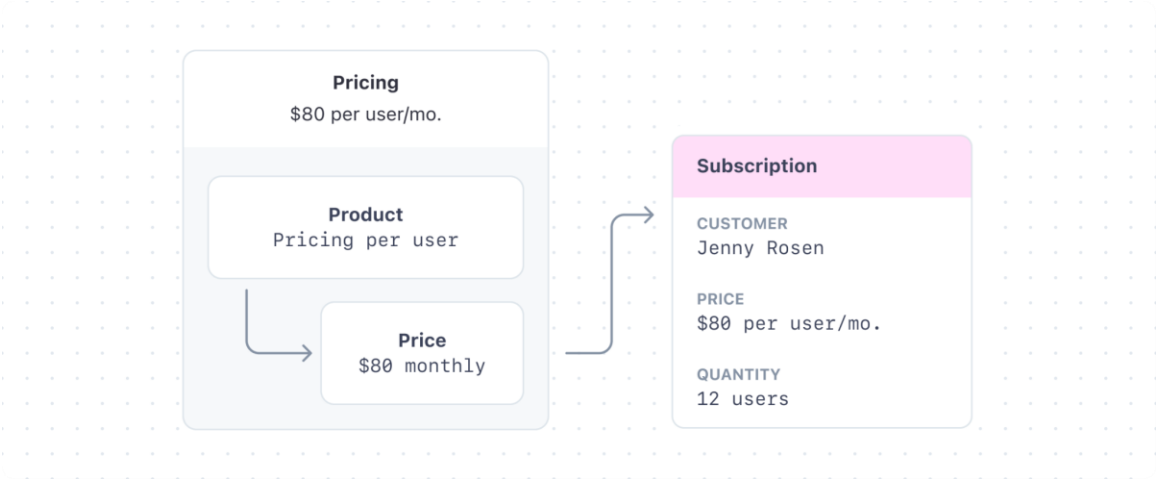

Figure 2. Example of a per-seat pricing model. (Users pay an amount proportional to the number of seats (users) using the services over a fixed period of time.)

Another very important pricing model is the Usage-based pricing model (Figure 3), in which the cost of the subscription is based on the amounts the users are using of the service product. It could be based, for example, on quotas, such as minutes of usage, quantity of memory or computing resources etc. In this model the prices can vary quite substantially, and it is up to the users to decide, monitor and manage the amounts spent.

Figure 3. Example of usage-based pricing model. (Users pay an amount proportional to the quantity of services used over a fixed period of time. These models allow users to pay as they go, and to control how much quotas they want to use for the quantities they want to hire.)

It is worth mentioning that sometimes the needs of the services logic require the subscription pricing models to be more complex and advanced. For example, the software presented here as study-case has a hybrid pricing model, with both a fixed-rate and a dynamic pricing components: The users need to pay for a certain plan the same way he would do using the Flat-rate pricing model to get access to a predefined amount of features. But they also need to pay for the “seats” that additional users will use. For the later component, the prices will differ depending not only on the quantity of seats, but also about their types, i.e., if it is a “facilitator” type of seat, the price will be higher, as opposed to the other available type called “panelist” (see practical section below, box 1 to understand better the

relationships between users in the study case).

To conclude, SaaS-type of software can involve multiple strategies for converting users, as well as charging for its services. The complexity of these strategies will translate into different options for acquiring the services, especially though time-based subscriptions to the services. These subscription models, either fixed or dynamically calculated, are used to charge users for the services. Regardless of the amount and periodicity of such payments, users are charged for the hired services. The remaining aspects of such systems involve how users are charged (i.e., the options available for secure payments and billing), and how these subscriptions are managed in the software (e.g., user management, unlocking new features, upgrading and downgrading plans, etc.). These are the topics of the next sections below.

2.3 Common payment and billing methods

Dealing with electronic payment transactions is impossible without knowing the available payment and billing methods, common practices and tools (A guide to payment methods…, n.d.)(Learn about payment methods, n.d.). The more payment options the service will provide during the payment procedure, better the flexibility is for the users. Also, different users have different personal preferences, and different requirements (individual users' needs and preferences are very different from companies). The need they all have in common, however, is security. That is by far the most important feature online transactions tools need to offer. Security granted, then diversity comes in hand, and the more options a system can offer, the higher are the chances that the users will have a good experience.

Probably the most commonly used tool for online payments are debit/credit cards. Cards support many features, but the important thing is that they accept transactions almost everywhere in the world, and with different currencies. In

addition, cards support using Manual capture API feature, which allows the charging part to place a hold on the funds when the customer authorizes the payment, but don’t capture the funds until later, which is not supported by all payment methods. That feature is very handy for automating and scheduling payments.

Bank transfers are also an important payment method, because they provide a way to send money directly to given bank accounts, being a common solution for larger needs, such as business to business services. An alternative very needed for businesses are invoices. In many cases it is actually a requirement to pay using such systems in the public sector. A key problem with bank transfers is that one can not control the amount of money users will send, and there could be accidents such as users sending too much or too little money. Luckily, present-day technologies can avoid such problems by using a special customer balance holding his bank transfers, which then is used to collect payments.

A last very modern and well-known technique to implement online transactions is digital wallets, such as Google and Apple pay. They store saved credit card information or have their own digital balance. Such global wallets as the mentioned Google pay or Apple pay supports invoicing and subscriptions, but there are some which are not that flexible (Wallets, n.d.). The wallets authenticate users by themselves, asking them to do the fingerprints on mobile or using the face recognition, the choices are defined by the wallet and are picked by the users.

All these systems are extremely complex, and because security is a must, they are not implemented by the software parts itself; it is a very common case that they will rely on third party services that are extremely reliable, and dedicated to ensure security throughout all the digital payment pipeline. That is the current market practice even for giant products that still rely on third party renowned payment companies and services.

2.4 General regulations and taxation for online payments

Each company which sells a product, no matter if it is digital or physical, needs to collect taxes through the payment transaction (Introduction to sales tax…, n.d.). It is important to note that those taxes are indirect taxes, except for the direct taxes which are paid based on the earning and profit. Indirect taxes are taken based on the sales of goods and services. They have different values and names in different countries and even states, but in Europe that tax is called VAT (Introduction to EU VAT & VAT OSS, n.d.).

That is another complex and highly difficult domain, in which companies can not afford mistakes. When trying to understand where, when and how to collect taxes, companies need to consider many aspects. Luckily, modern payment services supports automatic tax collection and even does not require installing additional plugins for it. Again, these services support business in implementing complex systems without the need of experts and dedicated teams to develop the functionality.

Altogether, the theoretical sections above make up the entire pipeline necessary to create and manage subscriptions, and their billing and payment, as well as all the complex elements related to it such as security and correct taxation.

2.5 Technical aspects of online payments

Depending on needs, the amount of responsibilities that one puts on third party services can be defined differently. Small companies can even adopt no-code versions of the service, and with just a few clicks put up a working payment system. For middle-sized and bigger companies, however, it is necessary to integrate the software and payment services with a code.

For that purpose the most common (and secure) approach is to use APIs. An API (Application Programming Interface) is a programmed set of instructions that

enables devices, applications and databases to share information (Heegaard, 2021). In accordance - payment API are used to help to optimize and simplify the payment process for both sellers and users. For the users, for example, by using Payment APIs they do not need to fill out checkout forms, because the payment information is already stored (given they authorized the service to do so). This way APIs can easily manage both ongoing and one-time payments, and the users do not need to worry about it. Automation of payments is made easy. For the service provider, the customer data and payment information is usually stored by the service, which highly increases data security and fraud prevention.

Those payment service technologies which connect clients with customers, granting access to the online and mobile payments are called payment gateways (Jones, 2020). Payment gateways are used to read and transfer the data of the payment from the customer’s bank account to the merchant's bank account, simplifying the payment process. Such services as Stripe or Paypal are very well known payment gateways.

To understand how the digital payment works, one needs to take into account all of its 4 actors: The cardholder (user customers paying for the services), the Merchant (the company providing the service), the acquirer and the issuing bank (Introduction to online payments, n.d.). When a cardholder-client wants to process money to a merchant (the service provider), the connection to the acquirer bank or payment processor is needed to be established. The institution will then process the payment to a payment network, for example, Mastercard or Visa, which then finally connects to the issuing bank. To secretly capture the details, a payment gateway is also needed, see Figure 4 for a schematic overview.

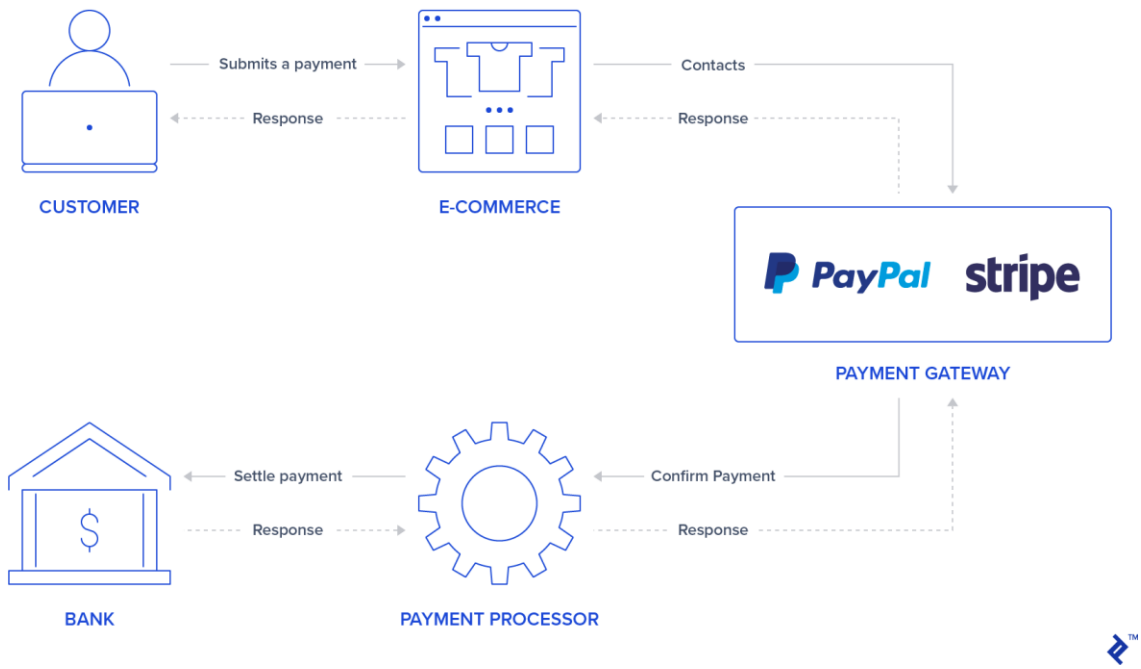

Figure 4. Whole process of the payment operation using API (Henrique, n.d).

One important thing to understand is that for their duties, the services take a fee, consisting of interchange and scheme fees. Interchange fee is taken by the issuing bank because it takes the risk by extending credit or banking services to the cardholder. The Schema fee is taken by the card networks themselves. The amounts taken as processing fees vary depending on the location, card type, the channel (in person or online) and Merchant Category Code.

All the conditions for such payment of fees for all service providers involved in transactions are clearly stated in the contracts, terms and conditions agreed for using these services. Usually, companies utilizing third-part payment systems incorporate the costs of these processing fees in the amounts charged for the subscriptions, and the end user hiring the services is the part covering all the costs, so companies can provide services while still profiting.

2.6 Security, data integrity and fraud avoidance

When talking about the network payment process, it is worth mentioning the regulations of this mechanism. Users do not want personal data and funds to be stolen, and companies charging for the services do not want to be victims of fraud. These can cause huge damage to companies, not only to their image for their customers, but big losses that can cause bankruptcy. That is the main reason why the Payment Card Industry Data Security Standards (PCI DSS) was created (Dahn, n.d.)(What is PCI?, n.d). These are a set of regulations for all organizations which handle cardholders and their authentication data, aiming to ensure security throughout the process for all parts involved. All reliable payment providers follow those recommendations.

PCI DSS includes three main core recommendations:

1) The data is collected and transmitted securely

   If the organization is not needed to handle card data, it should not. That is the case when you are using such providers as Stripe which takes that handling on their side which reduces the amount of needed security controls.

2) Data is stored securely, including monitoring the system and its access to the card data.

   The company storing such data should control the scope of its cardholder data environment (CDE) which consists on information of which are the people, the processes and the technologies that store, process or transmit the credit card data – or any system that is connected to it. The PCI security controls should apply to every device on its corporate network.

3) Annual control of the security system including different types of operations like questionnaires or external vulnerability scanning services.

As stated in many points of this thesis before, these are no simple issues, and that should be taken very seriously by the companies offering and charging for their services. It is clearly a difficult task, and better done when supported by dedicated and reliable systems and services developed specifically for that.

2.7 Compliance with the General Data Protection Regulation

The last aspect to be considered in the payment pipeline is data security. Not only the payment information, but any user data that is acquired at any level of the process. One aspect is how to collect and securely store such data, and the other aspect is how to be transparent to users how, why and when this data is being collected. Users should have access to this information, and most importantly, be capable of managing their own data, and delete it when necessary or when services are no longer under user control.

It is mandatory to obey the General Data Protection Law on the use of personal data (Hoffman, 2018) (Wolford, n.d.). This applies to all organizations within the EU, and those that supply goods or services to the EU or control EU citizens. The key points of this law are the following:

- The organizations must have a data control officer who ensures compliance with the rules
- The law applies without exception to all companies collecting personal data
- The client must receive the data provided to the organization upon request
- The client can delete all his data irrevocably
- There are penalties for non-compliance with the law

This law is important because it obliges the company to use protective tools, such as data encryption or two-factor authentication, which in turn increases the

security of personal data.

2.8 User experience and customer engagement

So far we have considered the technical aspects of building SaaS models, with a specific focus on the subscription and payment systems. The end users, however, are more interested in two factors: that their data will be handled securely, and that the payment process will go as smoothly as possible. The best paid services are the ones that users even forget they are paying for them.

In order for that to happen, and also to increase sales and conversions, UI/UX design comes into play. Companies, and designers and programmers, invest a huge amount of effort to create great experiences, so that users feel cared for and secure.

There are common recommendations aimed to improve user experience for the specific elements of choosing a subscription plan (Brandall, 2018):

- The number of subscriptions should not be more than five or fever than three, since a larger number can confuse the users, and the lack of alternatives can prevent the users from understanding which is the best available option.

- It is also important to highlight the strategically preferred option in the center, people tend to choose something that is in the center.

- A signature about who the plan is aimed at allows the client to understand his benefit from choosing one or another plan. It is important to support the decision on where to invest their money.

- People are more pleased when the price goes up, from left to right. That goes with their natural expectation.

- The price for the largest plan should be replaced with a label, something like "call us" so as not to scare customers with large numbers in the price.

That strategy also allows for more customizable options, and a sense of being exclusive.

- It's good if the client can use the trial version of the plan, because this allows him to understand how good the product is, and it allows a smoother onboarding to the software. It is much harder to convert users that do not know the software, and the benefits it can get.

2.9 Industry standards and customer's expectation for using SaaS

The adoption of SaaS models for very diverse companies has raised the bar for developing such solutions. Because so many software we utilize in our lives utilize these systems in a very secure, smooth, and pleasant way, the customers' expectation is that any SaaS service will be as simple to acquire as the known services (such as Netflix, Spotify, etc). In order to achieve such high standards and be competitive, small companies need to hire and benefit from third part - service providers. However, utilizing these service providers is only part of the equation. The other aspects to be considered are that the business decisions, the system and database architecture, the user interfaces and the users experiences are all thought under the perspective of SaaS. It is an entire industry that emerged, and came to stay. After the Covid19 crisis this has gained even more emphasis and relevance for business to customers and business to business solutions.

3 PRACTICAL IMPLEMENTATION

In this chapter, I present the case and context in which the thesis work was done, as well as the results of my hands-on research regarding the topic. I present the system's architecture in general, followed by a detailed assessment of the billing and subscription system, the core of my thesis. I finish by presenting the actual

code implementation and discussion the next steps and conclusions of this thesis.

3.1 Mindhive Oy

This thesis is done as part of the author's traineeship at Mindhive Oy, a startup based in Mikkeli, Finland.

Mindhive's mission is to help businesses to thrive by adapting to the opportunities provided by digitalization. Mindhive's philosophy starts from listening to the client and helping them to deliver their domain expertise in the form of a working digital product. Mindhive's focus is on the methodology that can help experts from different domains to express themselves in a manner that can be translated into a user-friendly software product.

On top of delivering the digital product, general customer experience is a core value of Mindhive. Projects are thus developed in an active partnership with the clients, and products are jointly defined with the client and Mindhive's experts. The goal is to use Mindhive's expertise to support the client in choosing the best set of technologies suitable for their product to excel in the market.

I've been working for the company since January 2022, joining the team as a part-time software developer. For the hands-on work related to this thesis, I had support from a team of developers, and worked on an already existing project that had a solid base and well-defined quality standards. Rather than building everything from scratch, I expanded the application that is currently operating as a freemium model to be capable of operating under a full subscription model (see the chapter 3.2 below for the study-case context).

3.2 The client case and its background

Metodix Oy owns the eDelphi software, a digital platform where users can utilize the Delphi method (see Box 1, below). The eDelphi has users and licenses on all continents except Antarctica. In 2021, there were users from 135 different

countries. The eDelphi has been developed during 20 years together with the Finnish future research institutions including the University of Turku Futures Research Centre and the Society for Futures Research. It has been used in many big projects and doctoral dissertations both in Finland and abroad, and it already has a solid base of users from academia and the public sector.

Box 1. The Delphi method in a nutshell.

The Delphi method is a structured communication technique originally developed as a systematic and interactive forecasting method which relies on opinions emitted by a panel of experts (Twin, 2021). The technique can be used in face-to-face meetings, as well as in remote meetings and virtual environments, being extremely relevant in the post-corona context.

Delphi has been widely used for business forecasting and has certain advantages over other structured forecasting approaches. It is based on the principle that the forecasts/opinions from a structured group of individuals are more accurate than those from unstructured groups. The experts usually answer questionnaires in two or more rounds. After each round, a facilitator provides an anonymised summary of the experts' forecasts from the previous round, as well as the reasons they provided for supporting their judgments. Experts are thus encouraged to revise their earlier answers in the light of the replies of other members of their panel. It is believed that during this process the variation in the answers will decrease and the group will naturally converge towards the "correct" answer. Finally, the process is stopped after a predefined stopping criteria (e.g., number of rounds, achievement of consensus, stability of the results), and the mean or median scores/opinions of the final rounds determine the results.

In a partnership with Mindhive Oy, two companies are developing a new version of the software called xDelphi. This new version of the software is not simply an

update, but a complete change in the platform's business model, visuals and user experience, aimed at expanding the use of the method and software to broader target audiences. Because of an NDA agreement, my thesis will not bring key and specific details of the new platform, but the core concept I worked on is presented without loss for the quality of this manuscript.

Briefly, the new software is a cloud-based platform operating with a Software-as-Service business model (Figure 5). This implies a modern software architecture and development technology, as well as the need of utilizing secure and reliable billing and subscription payment services, where users can choose the services they hire on the go, and are charged accordingly.

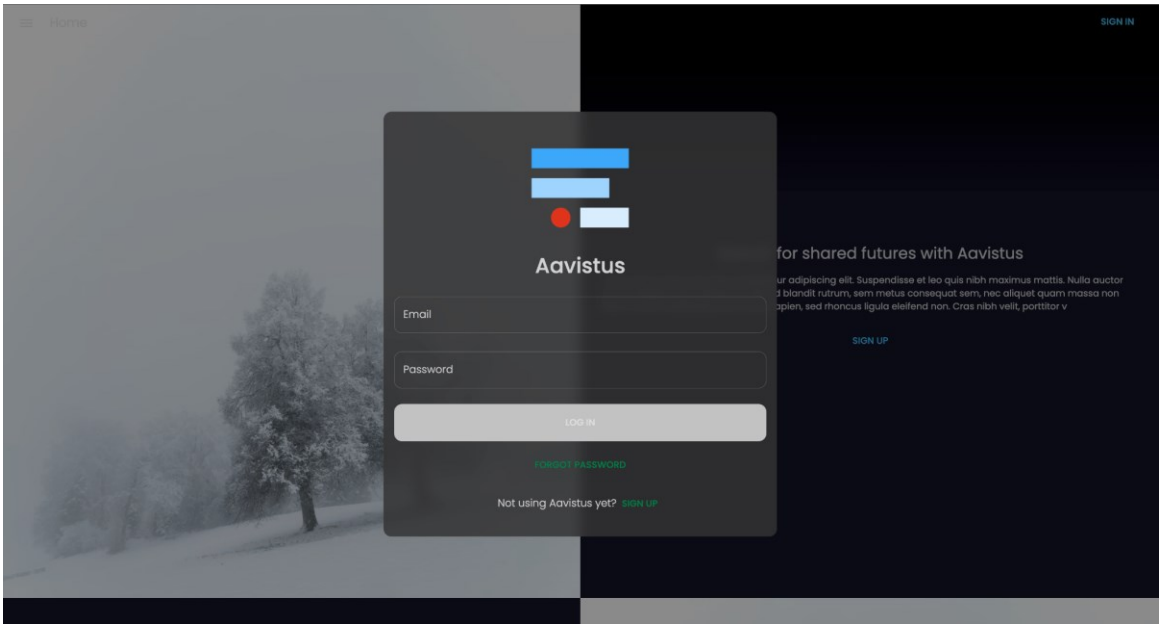

Figure 5. Landing page of the application focus of the study case in this thesis. (Aavistus is the broad-audience version of the platform that also exists targeted at the academic sector under the name of xDelphi.)

The details of this billing and subscription system, and its connection with the payment services are the object of my thesis, and its implementation in the xDelphi/Aavistus platform form the study case I present in the practical part of the thesis.

3.3 The application and the services offered by xDelphi

The application is structured around Spaces (Figure 6). On each Space the Admin users can create multiple Projects that contain queries with questions to be answered by the experts taking part into the discussion (i.e., the so-called panelists). The users operating as questionnaire facilitators moderate and stimulate the discussions by applying multiple tools and features offered in the application (through the Facilitator dashboard). The facilitators can analyze the opinions and shared knowledge arising from the discussions, and generate reports to be saved and exported. New insights and knowledge about topics are thus compiled by facilitators, and generated from the crowd-sourced information provided by the experts (panelists).

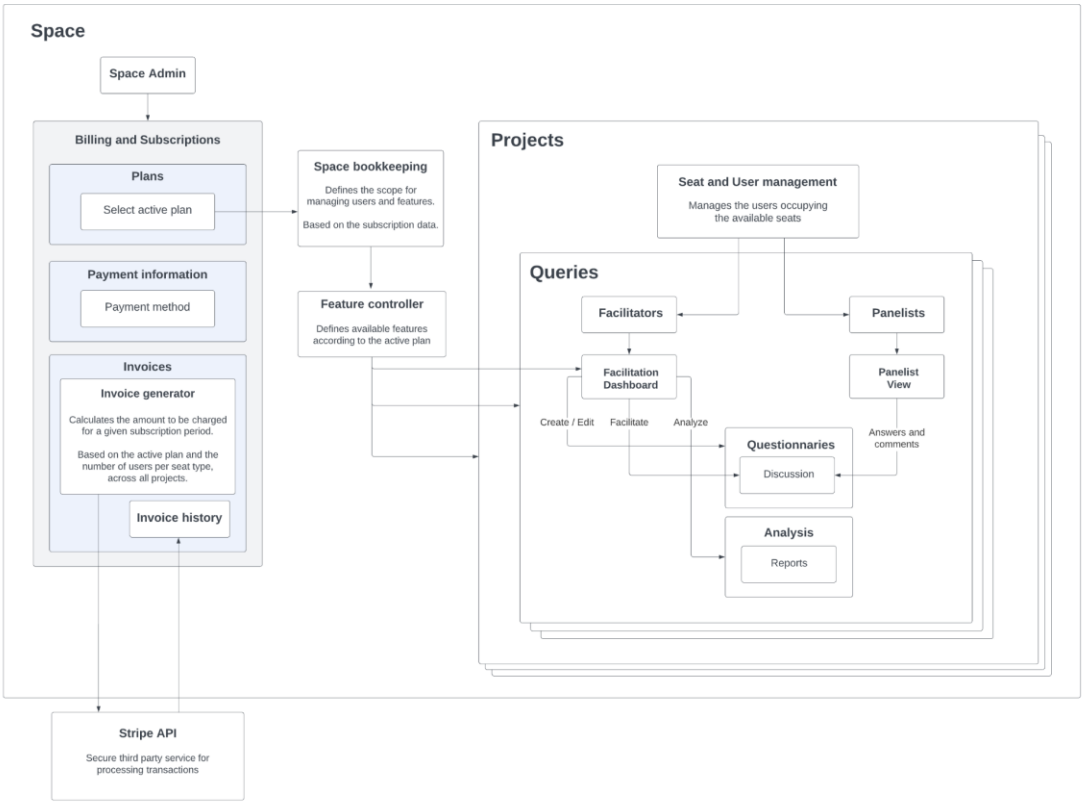

Figure 6. Simplified schematic representation of the xDelphi application, its structure, main functionality and user types. (The billing and subscriptions logic (colored boxes) are in the scope of this thesis, particularly the Invoices logic.)

The strength of the method, as well as of the software, is to offer facilitators with a great toolbox of questions and facilitation tools to extract the relevant information

from the panelist experts. The software offers its users multiple question types, and different analysis and reporting tools. The facilitators can then structure the questionnaires in very diverse ways and strategies, making the platform very versatile.

3.4 The service's pricing models, user interfaces, and its technical implementation

The application has a fixed and a dynamic pricing model for its subscription. In short, an invoice is an upfront monthly or yearly payment (2-month free discount for yearly) for the overall services hired within a given space. The space is charged by summing up the individual cost for all projects inside the space, proportionally to the number of Facilitators and Panelists used on each project, and added to the cost of the number of days used under any given plan used in the period. Equation 1 below describes the variables used to calculate the price.

$$
\begin{aligned}
&Monthly\ \text{cost}\ = \\
&\sum_{1:nproj}^{NProj}\left(\sum_{1:nplans}^{NPlans}\left(\sum_{1:nfac}^{NFac}\left(Nfac * \left(Fac\,€/day\right) * Ndays\right) + \sum_{1:npan}^{NPan}\left(Nfac * \left(Fac\,€/day\right) * Ndays\right)\right)\right) + \\
&\sum_{1:nplans}^{NPlans}\left(n\ days\ in\ the\ plan * \left(Plan\,€/day\right)\right)
\end{aligned}
$$

$$
\begin{aligned}
&Yearly\ \text{cost}\ = \\
&\left(\sum_{1:nproj}^{NProj}\left(\sum_{1:nplans}^{NPlans}\left(\sum_{1:nfac}^{NFac}\left(Nfac * \left(Fac\,€/day\right) * Ndays\right) + \sum_{1:npan}^{NPan}\left(Nfac * \left(Fac\,€/day\right) * Ndays\right)\right)\right) + \sum_{1:nplans}^{NPlans}\left(n\ days\ in\ the\ plan * \left(Plan\,€/day\right)\right)\right) * 0.833
\end{aligned}
$$

Equation 1. Variables used to calculate the price of a subscription invoice for a given space.

The monthly cost consists both of a dynamic (first row of the equation) and a fixed element (second row of the equation). The dynamic part depends on the numbers of facilitator- and panelist-seats hired for each plan active inside each project; the fixed part relates to the plans hired for the period. To get the total cost, we sum up the dynamic and fixed levels for all projects in the space, always proportional to the number of days spent on each different combination of seats and plans. The yearly cost applies the very same idea, but charged over the entire year period, with a 2-month discount.

The spaces Admin users get charged proportionally according to the model described above, for the chosen time period. Costs outside the original invoiced quantities are charged in separate, according to the plan or seats hired. Figure 7 is an example of an invoice created because the Admin user added more facilitator seats to a given project.

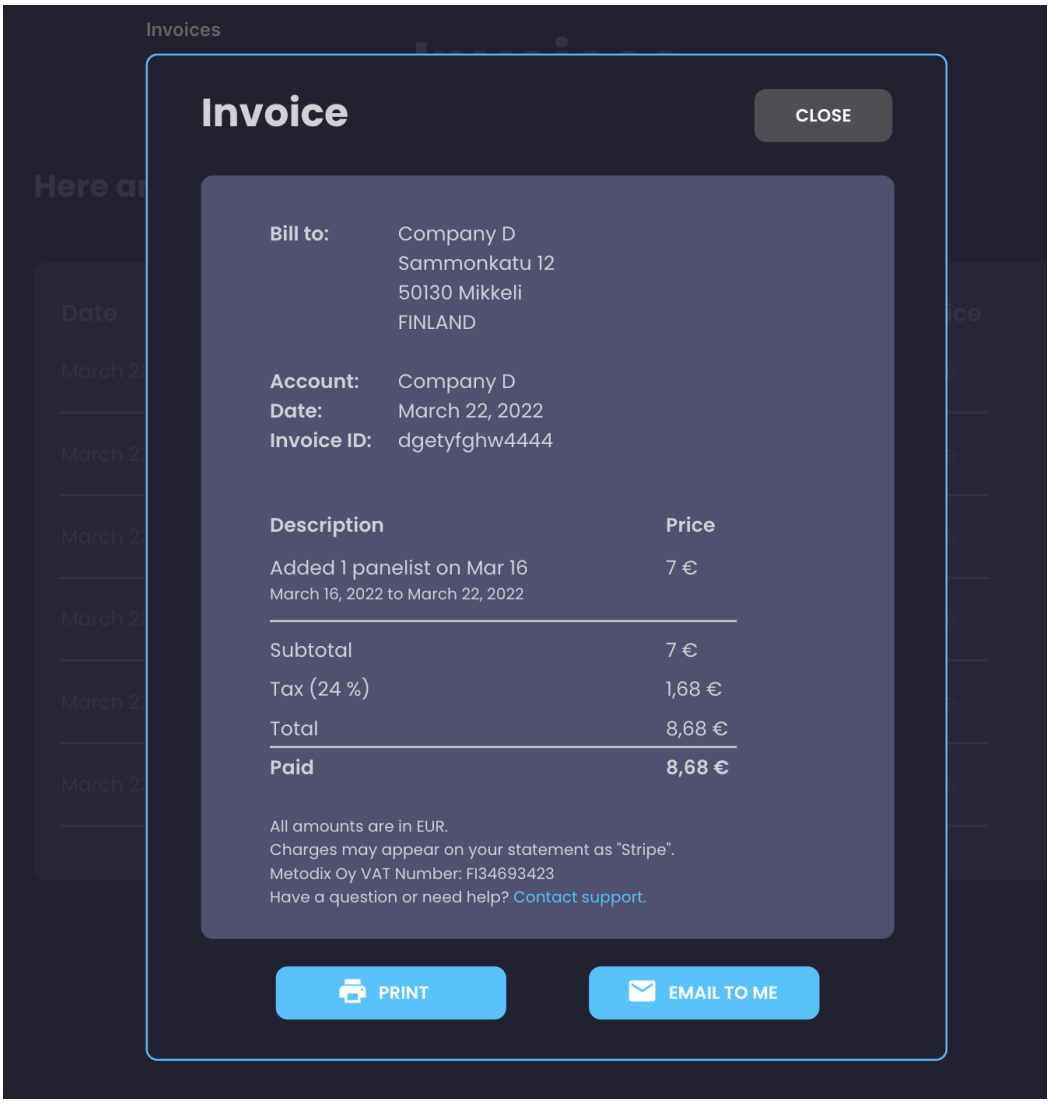

Figure 7. Invoice example for subscription acquired under a given period. (Items are described and charged according to the users' plans and specific space configuration.)

The end-users do not need to know the billing details to the level specified here (such as equation 1 above). It is enough for them to know the plans and number of seats they are hiring, for which timescale and the total costs, as well as the

costs of eventual changes in the amounts. These data are available to the end users through the Billing and Subscriptions user interfaces (Figure 8).

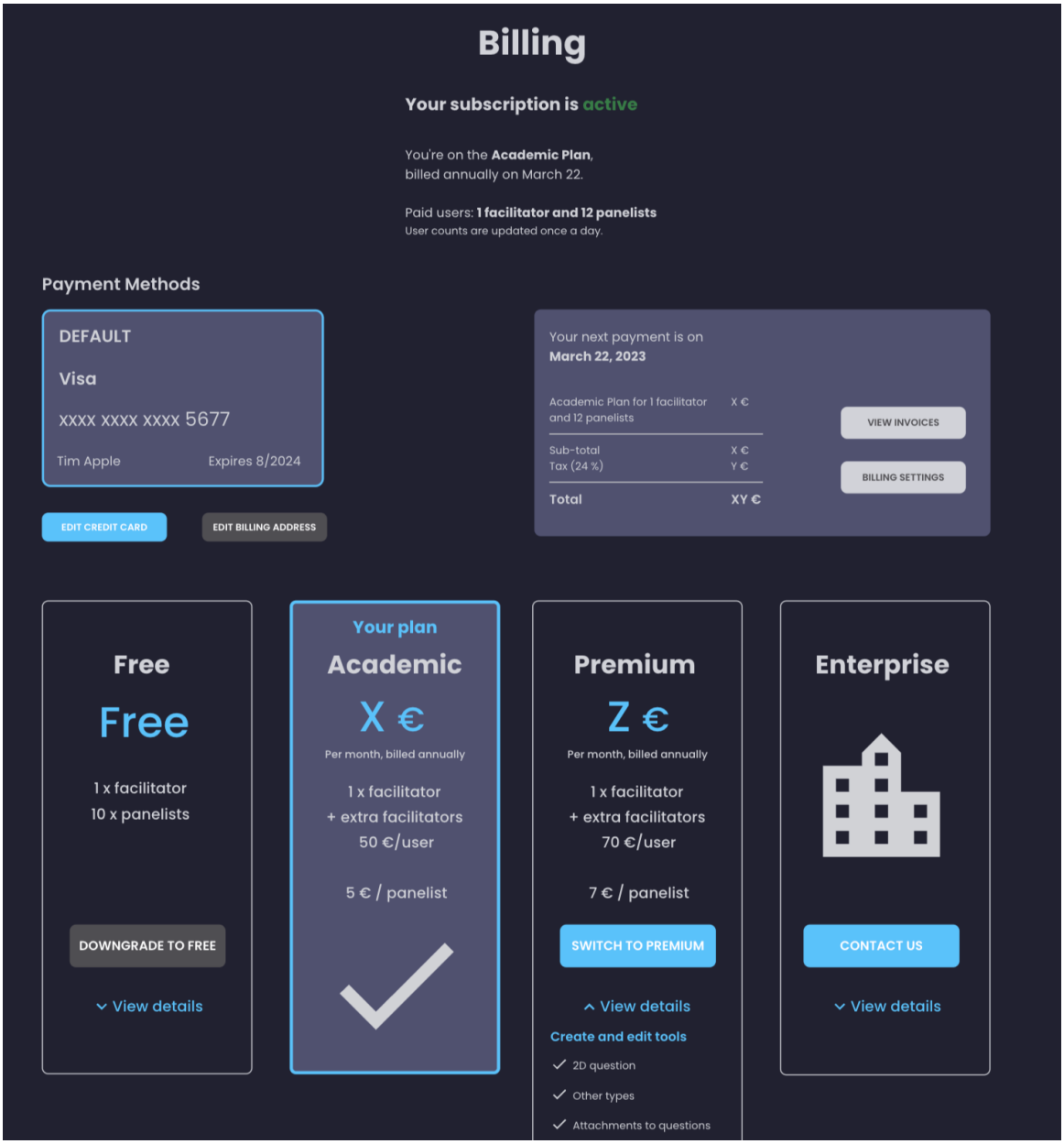

Figure 8. The Billing screen, from where users can upgrade/downgrade plans, add/edit payment methods and billing information, as well as see and download the previous invoices.

The system Admin users need to be capable to manage and monitor their payments for the services, as well as update payment information and other types of needed data. Figure 8 shows some of the screens from these domains.

3.5 Hands-on implementation and main findings from the study case

Though the xDelphi software and services have a non-trivial subscription pricing model, the billing system in itself is not as complicated as it sounds. In short, the component responsible for the invoices takes the responsibility of managing the invoice generation according to the rules described above and the specific context, on a case per case basis. This invoice for a given set of services is prepared and its data is passed to the stripe API, responsible for the actual payment and the secure transactions. The API passes back the information necessary for the application about the status of the transaction (i.e., whether successful or not) and to manage the bookkeeping of payments, subscription activation periods, and unlocking of the features and seats for the respective spaces and projects (Figure 6). I will now detail the technical implementation of this specific sub-system of the application (coloured part in Figure 6).

The implementation code is separated in three layers: 1) the Front-end, which brings components responsible for the user interaction with the service, 2) the Back-end, which has components responsible for the “background” operations, i.e. sending requests from the Front-end to the API, and managing and returning the responses or storing customer data, and 3) the API remote part, which is responsible for the actual payment operations in the secure environment, as well as returning different kind of responses (for example, if the payment operation is successful and the subscription is thus activated). Figure 9 below shows one example of a successful payment operation.

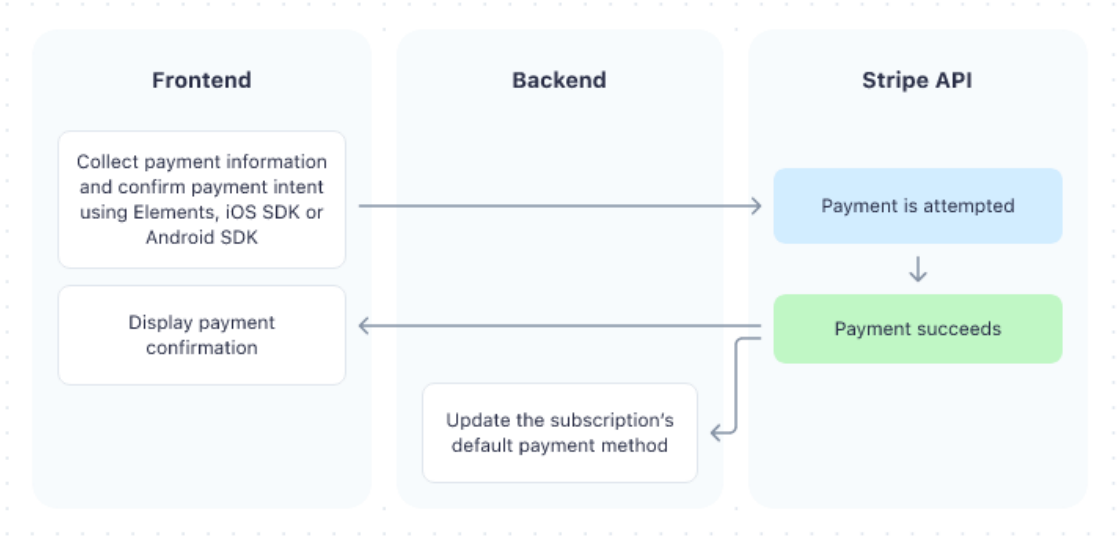

Figure 9. Example of a successful payment operation with the implemented system (How subscriptions work, n.d).

Going in more detail, let's start with the Front-end. Whenever we need to modify some data for the Stripe API (for example, when creating customers, managing subscriptions, managing invoices), the action starts with a button click from the user, on the web application page. In most cases the web part will send different kinds of requests to the server part, and thus receive different sorts of responses. For example, if one would like to register a new client as a customer in the service, then the button on the web page which is responsible for triggering that action needs to be clicked. When the user clicks it, it sends a request to the back-end which will fetch the necessary data about the user and send it to the API in order to register the user as a subscribing customer. The API will respond to the backend whether the registration succeeded or not, with the associated customer id in Stripe.

In this sequence, the backend takes responsibility to communicate with the API, and to provide the Front-end with the needed information after the request is processed. The backend has functions to send different requests to the Stripe API, for example “create subscription” or ”retrieve subscription information”. The function gets the response from the API, stores information in the database when needed, and returns the response data to the front end. In this work, the backend utilizes Node.js, and the front end is built with the React framework.

Lastly, the API executes the needed actions and manages the responses, and returns different data or errors depending on the result of the operation. All in all, it is very important to establish a reliable system with good communication between those 3 parts of the service, because all of them have their own responsibility and act differently to accomplish the necessary operations. The most risky processes all take place in the Stripe side, which adds another layer of security for users' data and payment transactions.

Stripe also provides developers with a react library to be installed to simulate the payment environment. With the aid of such a library, it is easier to test the code, and emulate payments to verify if the implementation is working as expected. The backend code is implemented in the form of various cloud functions, each of them operating when a specific kind of request is sent from the front to the backend (Figure 10 and 11).

```
const createSubscription = async (priceId) => {
  const {subscriptionId, clientSecret} = await fetch('/create-subscription', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
    },
    body: JSON.stringify({
      priceId
    }),
  }).then(r => r.json());

  setSubscriptionData({ subscriptionId, clientSecret });
}
```

Figure 10. Create a subscription request example.

```
app.post('/cancel-subscription', async (req, res) => {
  // Cancel the subscription
  try {
    const deletedSubscription = await stripe.subscriptions.del(
      req.body.subscriptionId
    );

    res.send({ subscription: deletedSubscription });
  } catch (error) {
    return res.status(400).send({ error: { message: error.message } });
  }
});
```

Figure 11. Cancel subscription function, running in the backend (Cancel a subscription, n.d.)(Cancel subscriptions, n.d.).

Since most of the procedures are handled by the API, the main hands-on work for my thesis was to research and code the components that make the integration of the prebuilt solutions offered by Stripe with the software developed at Mindhive. When constructing the integration, Stripe provides its users with very good and rich documentation (Build a subscriptions integration, n.d.).

3.6 How the creation and management of subscriptions happen at the API level

Stripe handles most of the processes relative to create and maintain the active subscription plans, their billing and payments. The process goes as follows: Firstly, the backend needs to create the subscription plans data as products, with all the necessary data, time periods and prices. For the price model used in the project, this means the fixed and dynamic components, with different plans and prices for seats, respectively. For example, the fixed-term products can be the plans called Free and Premium (Figure 12). The key data which define these products are their name (free and premium, respectively), the currency (e.g., EUR or USD), the unit amount, or cost (€0 and €200,00, respectively), and the time interval it covers (Monthly or Yearly). The dynamic part of the pricing, on the other hand, could be defined using plan-seat-type combinations for the products: For example, the panelist type of seat for two given types of plans, say Academic and Premium, when paid monthly would cost respectively €5,00 and €7,00 per seat, per month (Figure 13).

```
{
  "name": "free_plan",
  "path": "/v1/products",
  "method": "post",
  "params": {
    "name": "Free"
  }
},
{
  "name": "free_plan_price",
  "path": "/v1/prices",
  "method": "post",
  "params": {
    "product": "${free_plan:id}",
    "lookup_key": "sample_basic",
    "currency": "usd",
    "unit_amount": 0,
    "recurring": {
      "interval": "month"
    },
    "metadata": {
      "sample": "fixed-price"
    }
  }
},
```

```
{
  "name": "premium_plan",
  "path": "/v1/products",
  "method": "post",
  "params": {
    "name": "Premium"
  }
},
{
  "name": "premium_plan_price",
  "path": "/v1/prices",
  "method": "post",
  "params": {
    "product": "${premium_plan:id}",
    "lookup_key": "sample_premium",
    "currency": "usd",
    "unit_amount": 20000,
    "recurring": {
      "interval": "month"
    },
    "metadata": {
      "sample": "fixed-price"
    }
  }
}
```

Figure 12. Example of data defining the price for the fixed part of the model. It defines two types of plans, Free and Premium, when paid monthly. The values are respectively zero and €200,00 per month.

```
{
  "name": "panelists",
  "path": "/v1/products",
  "method": "post",
  "params": {
    "name": "Panelists"
  }
},
{
  "name": "academic_panelists_price",
  "path": "/v1/prices",
  "method": "post",
  "params": {
    "product": "${panelists:id}",
    "lookup_key": "sample_academic_panelist",
    "currency": "usd",
    "unit_amount": 500,
    "recurring": {
      "interval": "month"
    },
    "metadata": {
      "sample": "fixed-price"
    }
  }
},
{
  "name": "premium_panelists_price",
  "path": "/v1/prices",
  "method": "post",
  "params": {
    "product": "${panelists:id}",
    "lookup_key": "sample_premium_panelist",
    "currency": "usd",
    "unit_amount": 700,
    "recurring": {
      "interval": "month"
    },
    "metadata": {
      "sample": "fixed-price"
    }
  }
}
```

Figure 13. Example of data defining the price for the dynamic part of the model, i.e., defined by the number of seats acquired. It defines the panelist type of seat for two types of plans, Academic and Premium, when paid monthly. The values are respectively €5,00 and €7,00 per seat, per month.

After the products are defined, Stripe will create ids for those prices and products and the backend can use this information to put together the complex invoicing of

subscriptions. This data can be requested from the API as shown in Figure 14.

```
app.get('/config', async (req, res) => {
  const prices = await stripe.prices.list({
    lookup_keys: ['sample_basic', 'sample_premium'],
    expand: ['data.product']
  });

  res.send({
    publishableKey: process.env.STRIPE_PUBLISHABLE_KEY,
    prices: prices.data,
  });
});
```

Figure 14. Example on how to request product data from Stripe API.

The next step is to gather the necessary data to create a customer. The backend thus creates a customer request to the API (Figure 15).

```
app.post('/create-customer', async (req, res) => {
  // Create a new customer object
  const customer = await stripe.customers.create({
    email: req.body.email,
  });

  // Save the customer.id in your database alongside your user.

  res.send({ customer: customer });
});
```

Figure 15. Example on how to create a customer from Stripe API.

After having created the plan-seats-prices and the user data, it is time to put together the information necessary to generate an invoice to be paid by the user.

The way the subscription invoices are put together is a simple purchase of one unit of a given type of plan-product (e.g., Academic), added with the desired units of plan-seat-type products for facilitators and panelists (for example, 5 units of facilitator-academic-product and 30 units of panelist-academic-product). This combination would result in a monthly payment for the Academic plan containing a single project with 5 facilitator and 30 panelist seats in it.

That is the simplest way to describe the basic unit. Of course, the total cost of the subscription applies the same rules and logic, but summing up the number of seats across all the projects that the user created during the billing period.

With all the ingredients at hand, the API can create the subscription using the user id, the plan price id, the recurring payment period which is part of the price and the amount of panelists and facilitators (Figure 16).

```
const subscription = await stripe.subscriptions.create({
  customer: customerId,
  items: [
  { price: req.body.planPriceId },
  { price: req.body.panelistsPriceId, quantity: req.body.panelistsQuantity },
  { price: req.body.facilitatorsPriceId, quantity: req.body.facilitatorsQuantity }],
  payment_behavior: 'default_incomplete',
  expand: ['latest_invoice.payment_intent'],
});
```

Figure 16. Example on how to create a subscription using the Stripe API.

After the creation, the API returns with the Subscription data object, which is then used by the backend to activate the features of the app, number of seats and all necessary information to grant all users under a given subscription access to the application.

Note, however, that after the subscription is created its status is “unpaid”. Then from the Web side we request Stripe to proceed our payment with card details we filled and when completed it will return the paymentIntent which has the Status of the operation (Figure 17).

```
let { error, paymentIntent } = await stripe.confirmCardPayment(clientSecret, {
  payment_method: {
    card: cardElement,
    billing_details: {
      name: name,
    }
  }
});

if(error) {
  // show error and collect new card details.
  setMessage(error.message);
  return;
}
setPaymentIntent(paymentIntent);
```

```
    if(paymentIntent && paymentIntent.status === 'succeeded') {
      return <Redirect to={{pathname: '/account'}} />
    }
```

Figure 17. Status of the operations, returned by the Stripe API.

If the payment succeeds, the user subscription will be activated and the payment situation completed; If it fails, then the software needs to notify the user to repeat the operation, or edit payment information, or any other action we might have interest to perform.

Every time a user utilizing a subscription from a given plan enter the site, we verify that the subscription is active, which will grant access to the relevant features and seats; If the status of the subscription changes, then the user will be locked from using the features, and a notification for checking the situation is sent to the user.

Lastly, Stripe manages the subscriptions and recurrent payments automatically, which means that if the user gave permission to be charged, the payments will be done automatically directly from the users card. To stop the payments, the user needs to cancel the subscription for the plan. When that is done, Stripe will stop

the automatic payments, and the user will have access to the application until the end of the subscription period. After that the user is downgraded to the free plan, and will have access to a limited set of features and seats. At any point the user can activate and upgrade plans again, and the payments will start again.

4 DISCUSSION AND CONCLUSION

This thesis deals with the topic of utilizing modern torrid-party services to build secure payment and subscription systems for Software-as-a-service type of digital products. I developed and implemented the code capable to create and manage users' subscriptions in a safe environment, providing users with a smooth experience to acquire extra features for the services hired. The approach benefits from the integrations with Stripe services, allowing digital online payments in all modern formats. The thesis also presents a detailed theoretical introduction to the relevant topics of SaaS and online payments, such as security, data regulations and taxation.

The strength of this thesis is to use a real-world study case from its start, allowing me to develop real-world working skills other than solely computer programming. The approach for the project was modern and holistic, and one of the clear goals was to combine my coding skills with the team of developers, in order to be capable to develop this project as part of a real software being developed by Mindhive Oy. That was surely not an easy task, and challenged me in many ways.

It is also worth mentioning that this work is a solid first step that will still be developed in many different directions. Currently, the implementation handles the basic cases. The calculation of the cost of the bills for the subscriptions tool is complex, bringing a fixed and a dynamic term, which makes it somewhat complex even for simple cases. The real-world exception cases, however, can be even more complex. The examples implemented will not change in time, and the execution of the payments will not return money proportionally, i.e., when a user downscale its subscription. The paid quantity will remain the same, and the new value will be valid only for the future payments.

The reality of complex systems can be quite different. Users usually acquire different plans, and it is quite common action to downscale and upscale plans very frequently. That type of changes alter the fixed term of the price calculation, relative to the plans. However, changing plans imply changes in the price of seats, and therefore will alter the price of the dynamic part. When dealing with such cases, it is common that the software will charge more from the users when necessary, but will also return money to credit cards when that is the case.

This becomes even more complex when the subscriptions are paid for the yearly mode. Throughout a full year, it is very common that different projects will have seats added and reduced, implying charging and returning extra money to the users cards. If the users are changing plans, it is another thing to be considered. Lastly, the software might offer discounts, promotions and bonuses, that will all have an effect on the dynamics of the charges and returns to the cards. Luckily, Stripe provides very good documentation, tutorials and code examples that cover all complex variations discussed here.

The next steps of this work, i.e., the continuation of what was started in this thesis, is exactly to incorporate the complex cases discussed above. To conclude, this thesis resulted in a great deal of professional development for me, and provided the broader project I took part at Mindhive Oy with a solid implementation for the subscription and payment systems that will now be continued by me and my team mates. Lastly, the concepts in this thesis are general enough that others can benefit from the learnings and case presented here.

REFERENCES

Turner, B. 2020. What is SaaS? Everything you need to know about Software as a Service. Blog. September 17, 2020. Available at: https://www.techradar.com/news/what-is-saas [Accessed 1 March 2022].

McKenzie, P. n.d. The business of SaaS. Web page. Available at: https://stripe.com/en-gb-fi/atlas/guides/business-of-saas [Accessed 1 March 2022].

Ndukwu, D. 2021. How To Build A Low Touch Sales Funnel That Sells Like A Human. Blog. December 21, 2021. Available at: https://bloggingwizard.com/low-touch-sales-funnel [Accessed 1 March 2022].

High Touch Business Model - How Does It Work? 2022. Herein Wikisme. Web page. Available at: https://www.wikisme.com/high-touch-business-model/ [Accessed 1 March 2022].

Stripe. n.d. Pricing models for recurring billing. Web Page. Available at: https://stripe.com/docs/products-prices/pricing-models [Accessed 1 March 2022].

Law, R. 2021. The ultimate guide to saas pricing models, strategies & psychological hacks. Blog. July 12, 2021. Available at: https://www.cobloom.com/blog/saas-pricing-models [Accessed 1 March 2022].

Stripe. n.d. A guide to payment methods. Web page. Available at: https://stripe.com/en-gb-fi/payments/payment-methods-guide [Accessed 1 March 2022].

Stripe. n.d. Learn about payment methods. Web Page. Available at: https://stripe.com/docs/payments/payment-methods/overview [Accessed 1 March 2022].

Stripe. n.d. Wallets. Web Page. Available at: https://stripe.com/docs/payments/wallets [Accessed 1 March 2022].

Stripe. n.d. Introduction to sales tax, VAT and GST compliance. Web page. Available at: https://stripe.com/en-gb-fi/guides/introduction-to-sales-tax-vat-and-gst-compliance [Accessed 1 March 2022].

Stripe. n.d. Introduction to EU VAT & VAT OSS. Web page. Available at: https://stripe.com/en-gb-fi/guides/introduction-to-eu-vat-and-vat-oss [Accessed 1 March 2022].

Heegaard, S. 2021. Payment API. Blog. August 31, 2021. Available at: https://rechargepayments.com/glossary/payment-api/ [Accessed 1 March 2022].

Jones, A. 2020. A Guide to Payment Gateway APIs. Blog. May 25, 2021. Available at: https://www.globalpaymentsintegrated.com/en-us/blog/2020/08/25/a-guide-to-payment-gateway-apis [Accessed 1 March 2022].

Stripe. n.d. Introduction to online payments. Web page. Available at: https://stripe.com/en-gb-fi/guides/introduction-to-online-payments [Accessed 1 March 2022].

Henrique, S. n.d. Integrating Stripe and PayPal Payment Methods in Ruby on Rails. Blog. Available at: https://www.toptal.com/ruby-on-rails/ruby-on-rails-ecommerce-tutorial [Accessed 1 March 2022].

Dahn, M. n.d. A guide to PCI compliance. Web page. Available at: https://stripe.com/en-gb-fi/guides/pci-compliance [Accessed 1 March 2022].

What is PCI? n.d. PCI Compliance Guide. Web page. Available at: https://www.pcicomplianceguide.org/faq/#1 [Accessed 1 March 2022].

Hoffman, S. 2018. General Data Protection Regulation (GDPR). Web page. September 25, 2020. Available at: https://stripe.com/en-gb-fi/guides/general-data-protection-regulation [Accessed 1 March 2022].

Wolford, B. n.d. What is GDPR, the EU's new data protection law? Web page. Available at: https://gdpr.eu/what-is-gdpr/ [Accessed: 16 April 2022].

Brandall, B. 2018. I Analyzed 250 SaaS Pricing Pages — Here's What I Found. Blog. June 28, 2018. Available at: https://www.process.st/saas-pricing-pages/ [Accessed 1 March 2022].

Twin, A. 2021. Delphi Method. Blog. September 04, 2021 Available at: https://www.investopedia.com/terms/d/delphi-method.asp [Accessed 1 March 2022].

Stripe. n.d. How subscriptions work. Web Page. Available at: https://stripe.com/docs/billing/subscriptions/overview [Accessed 1 March 2022].

Stripe. n.d. Build a subscriptions integration. Web Page. Available at: https://stripe.com/docs/billing/subscriptions/build-subscriptions [Accessed 1 March 2022].

Stripe. n.d. Cancel a subscription. Web Page. Available at: https://stripe.com/docs/api/subscriptions/cancel [Accessed 1 March 2022].

Stripe. n.d. Cancel subscriptions. Web Page. Available at: https://stripe.com/docs/billing/subscriptions/cancel [Accessed 1 March 2022].