INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




High Performing Transactions Processing - Best ways to scale up transaction processing from hundreds of thousands to hundreds of millions

FRANCISCA INÊS MARCOS DE BARROS
Junho de 2023

POLITÉCNICO
DO PORTO

ISEP INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO

High Performing Transactions Processing

Best ways to scale up transaction processing from hundreds of thousands to hundreds of millions

Francisca Inês Marcos de Barros

A dissertation submitted in partial fulfillment of the requirements for the degree of Master of Science, Specialisation Area of Computer Systems

Supervisor: Dr. Jorge Manuel Neves Coelho
Co-Supervisor: Dr. Luís Miguel Pinho Nogueira
Advisor: Gabriel Gois de Melo

Evaluation Committee:
President:
-

Members:
-

Porto, June 28, 2023

Statement of Integrity

I hereby declare having conducted this academic work with integrity.

I have not plagiarised or applied any form of undue use of information or falsification of results along the process leading to its elaboration.

Therefore the work presented in this document is original and authored by me, having not previously been used for any other end.

I further declare that I have fully acknowledged the Code of Ethical Conduct of P.PORTO.

ISEP, Porto, June 28, 2023

Dedicatory

To my family, colleagues and friends for all the support they gave me.

And

For Nola, whose unconditional affection and loyalty I couldn't repay with a lifetime of dog treats. But I'm going to give it a go.

Abstract

In an era in which technology is more pervasive in our everyday lives, ultimately substituting physical currency with dematerialised alternatives (for example, credit and debit cards, electronic wallets), merchants are being compelled to adapt. As a result, all of them should have some method of receiving payments aimed towards this technological advancement or risk losing clients and, ultimately, the business itself.

This demand is coupled with highly bureaucratic issues and the low investment capability of small and medium-sized merchants, which will almost likely make subscribing to these revolutionary new payment systems challenging. Hardware with a steep learning curve, long-term contracts, complex implementation charges, and undisclosed usage fees make it challenging for small local businesses to get paid.

Regarding the complete market offer for this purpose, systems with payment terminals, their resilience and capacity to operate in full condition whenever the demand of utilisation increases remains the most important consideration of all - considering the enormous amount of transactions processed by the rising customer base.

Therefore, organisations who want to enter this competitive industry must assure, in addition to security in the processing of sensitive data, the availability of their product throughout time, in order for these services to become accessible and trustworthy for their clients.

The purpose of this dissertation is to investigate the existing limitations of Saltpay's internal payment gateway, which constitutes one of the critical points of interaction for physical payment terminals, in order to contribute to the successful expansion of both the firm and its clients.

Some of the issues that have already been identified are related to the latency introduced by the various requests made within the scope of a transaction (risk analysis, parameter validation, message transformation, information enrichment, tokenisation, and so on), but also to the consumption of transaction events that will inform all downstream systems of the transaction's authorisation and that must be in near total synchrony.

Analysing the load testing findings made it possible to identify one of the crucial points in the processing of transactions, the Procesing-API, as well as various configurations that should be revised, such as Kubernetes' automated scaling. This allowed for the design of expansion strategies to meet the increasing number of daily transactions, resulting in a more complete and competitive market offering.

Keywords: Payments, Software Performance, Load Testing, Capacity Planing, Reliability

Resumo

Numa era onde a tecnologia se mostra cada vez mais proeminente no nosso quotidiano, substituindo até o dinheiro físico por soluções desmaterializadas (por exemplo, cartões de débito e crédito, carteiras eletrónicas), os comerciantes vêm-se obrigados a acompanhar esta mudança. Deste modo, todos estes têm que possuir alguma forma de aceitar pagamentos direcionados a esse avanço tecnológico, ou arriscar potencialmente a perda de clientes e, consequentemente, a perda do próprio negócio.

Esta necessidade é acompanhada por problemas burocráticos complexos e pela baixa capacidade de investimento dos pequenos e médios comerciantes, que certamente levará a dificuldades em adesão a estas novas soluções inovadoras de pagamentos. Hardware com uma elevada curva de aprendizagem, contratos de longo prazo, custos de implantação complexos e taxas de utilização ocultas dificultam o recebimento de pagamento por essas empresas locais.

No que concerne a toda a oferta disponibilizada no mercado para este fim, soluções com terminais para aceitar pagamentos, a sua robustez e capacidade de funcionamento em plenas condições sempre que se dá um aumento de carga de utilização continua a ser uma das prioridades mais importantes - tendo em conta a enorme quantidade de transações processadas pela crescente base de clientes.

Assim, as organizações que pretendem penetrar neste setor do mercado competitivo, devem assegurar, além da segurança no processamento de dados sensíveis, a disponibilidade do seu produto ao longo do tempo, para que estes serviços se tornem acessíveis e confiáveis para seus clientes.

Pretende estudar-se as limitações atuais da gateway de pagamentos interna da Saltpay, que é um dos ponto crítico de interação dos terminais de pagamento físicos, por forma a promover o crescimento bem sucedido tanto da empresa quanto dos seus clientes.

Alguns dos problemas já identificados estão ligados com a latência introduzida pelos diferentes pedidos feitos no âmbito de uma transação (análise de risco, validação de parâmetros, transformação de mensagens, enriquecimento de informação, tokenização, etc.), mas também com o consumo dos eventos de transações que vão informar todos os sistemas a jusante da autorização da transação e que precisam de estar em quase total sincronia

Analisando os resultados obtidos através da realização de testes de carga foi possível identificar um dos principal pontos críticos no processamento de transações, a Processing-API, e as diferentes configurações de componentes que devem ser melhoradas, como o escalonamento automático do Kubernetes. Deste modo, foi possível determinar estratégias de expansão para acomodar o número crescente de transações diárias, fornecendo no final um produto mais completo e competitivo no mercado.

Acknowledgement

I'd also like to thank my supervisor, Dr. Jorge Coelho, and co-supervisor, Dr. Luís Nogueira for providing the support necessary throughout the whole thesis development, and for listening and reassuring me when I needed it most. I cannot forget thanking my advisor Gabriel Melo, for always believing I was capable of completing this challenge.

This thesis would not have been possible without any one of you.

Introductory Note

In April 2023, SaltPay re-branded to Teya. All references to both firms' names should be considered as being one and the same.

Switch (also known as SwitchPayments) was acquired by SaltPay in January 2022 and was the business that developed the Payments Gateway. All references to this organisation should be interpreted as references to the Payments Gateway team or to SaltPay.

Contents

List of Figures **xvii**

List of Tables **xix**

List of Acronyms **xxi**

Glossary **xxiii**

1 Introduction **1**
1.1 Context . . . 1
1.2 Motivation . . . 3
1.3 Objectives . . . 4
1.4 Value Analysis . . . 5
1.5 Project Methodology . . . 5
1.5.1 Development Process . . . 5
1.5.2 Version Control System . . . 8
1.5.3 Workflow . . . 8
1.6 Document Structure . . . 9

2 Context and State of the Art **11**
2.1 Context . . . 11
2.1.1 The Payment Industry . . . 11
2.1.2 Key Context Definitions . . . 12
2.1.3 Business Concepts . . . 14
2.2 State of the Art . . . 15
2.2.1 Existing Solutions . . . 15
2.2.2 Comparison between solutions . . . 17
2.2.3 Relevant Technologies . . . 18
2.2.4 Comparison between technologies . . . 21

3 Value Analysis **23**
3.1 Innovation Process . . . 23
3.1.1 Opportunity Identification . . . 24
3.1.2 Opportunity Analysis . . . 25
3.1.3 Idea Genesis . . . 26
3.1.4 Idea Selection . . . 26
3.2 Value . . . 26
3.3 Value Proposition . . . 27
3.4 Business Model Canvas . . . 27
3.5 Quality Function Deployment . . . 29

4 Evaluation and Experimentation **33**
4.1 Research Hypotheses . . . 33
4.2 Evaluation Indicators and Information Sources . . . 34
4.3 Evaluation Methodology . . . 35
4.3.1 Load Tests . . . 35
4.3.2 Testing Scenarios . . . 36
4.3.3 Metric Dimensions . . . 38
4.3.4 Metric Criteria . . . 40
4.4 Results . . . 40
4.5 Assessment Completion . . . 44

5 Optimisation Strategies and Implementation Suggestion **47**
5.1 Optimisation Strategies . . . 47
5.1.1 Services Heterogeneity . . . 47
5.1.2 Transaction Parameters Enrichment . . . 48
5.1.3 Transaction Flow Simplification . . . 49
5.1.4 Source Code Optimisation . . . 51
5.1.5 K8s Optimisation . . . 53
5.1.6 AWS Optimisation . . . 54
5.2 Implementation Suggestion . . . 56
5.2.1 RabbitMQ Connection Reuse . . . 56
5.2.2 Processing-API Resources Adjustments . . . 58
5.2.3 Auto-Scaling . . . 59

6 Conclusions **61**
6.1 Objectives Completed . . . 61
6.2 Challenges Encountered . . . 61
6.3 Future Work . . . 62
6.4 Final Considerations . . . 63

Bibliography **65**

List of Figures

1.1 SaltPay services architecture 2
1.2 SaltPay Transaction Flow 3
1.3 Thesis Epic 6
1.4 Scrum pillars 6
1.5 Scrum sprint cycle 7

2.1 Payment Instruments 2021 11
2.2 Four-Party Scheme 13
2.3 Card Ecosystem 15
2.4 JMeter Screen 18
2.5 Locust Screen 19
2.6 Vegeta Command Output 20
2.7 K6 Results 20

3.1 Innovation Process 23
3.2 New Concept Development Model 24
3.3 Fintech Investments Worldwide, from 2010 to H1 2022 25
3.4 Business Model Canvas 27
3.5 Quality Function Deployment 31

4.1 Pay-By-Link Example 36
4.2 Card-Present Transaction Method 37
4.3 Create Instrument Method 38
4.4 Processing API Metrics Dashboard 39
4.5 K6 Output Report 39
4.6 Processing API Response Times and CPU Metrics 43
4.7 Transaction Lifecycle Steps Execution Time 44
4.8 Trace Example 44

5.1 Data Enrichers Sequential Call 48
5.2 Data Enrichers Thread-Pool Call 49
5.3 Risk-API Call 50
5.4 QuerySet.explain() Example 51
5.5 Connecting to RabbitMQ Span 53
5.6 Processing-API Auto-scaling Configuration 54
5.7 AWS Auto-scaling Group 55
5.8 Current Pika Client Implementation 56
5.9 Suggested Pika Client With Connection Pool 57
5.10 Connection Pool Pika Client Methods 57
5.11 Suggested Change to Reuse Connections 58
5.12 Suggested Upgrade to Processing-API Resources 58
5.13 Suggested Changes to Support Uvicorn 59

5.14 Suggested Kubernetes Auto-Scaling Configuration 60

List of Tables

2.1 Comparison between existing solutions 17

3.1 Benefits and sacrifices to the customer 26
3.2 QFD demanded qualities and weights 30
3.3 QFD functional requirements and direction of improvement 30

4.1 Results obtained in load tests with 20 VUs and 200 iterations 41
4.2 Results obtained in load tests with a constant-arrival-rate executor of 10 requests per second 42

List of Acronyms

3DS	3-D Secure.
AI	Artificial Intelligence.
AMQP	Advanced Message Queuing Protocol.
API	Application Programming Interface.
ASGI	Asynchronous Server Gateway Interface.
ATM	Automated Teller Machine.
AWS	Amazon Web Services.
BdP	Banco de Portugal.
BIN	Bank Identification Number.
CGD	Caixa Geral de Depósitos.
CLI	Command-Line Interface.
CPU	Central Procesing Unit.
CVV	Card Verification Value.
DSS	Data Security Standards.
EC2	Elastic Compute Cloud.
ECB	European Central Bank.
EEA	European Economic Area.
EU	European Union.
FaaS	Fintech as a Service.
FFE	Fuzzy Front End.
GUI	Graphical User Interface.
HMAC	Hash-based Message Authentication Code.
HPA	HorizontalPodAutoscaler.
HSM	Hardware Security Model.
ISV	Independent Software Vendor.
MPI	Merchant Plug-In.
NCD	New Concept Development.
NPD	New Product Development.
PCI	Payment Card Industry.

PIN	Personal Identification Number.
POS	Point of Sale.
PR	Pull Request.
PSP	Payment Service Provider.
QFD	Quality Function Deployment.
QoS	Quality of Service.
RAM	Random Access Memory.
RC	Release Candidate.
SICOI	Portuguese Interbank Clearing System.
SLA	Service Level Agreement.
SMB	Small and Medium-Sized Business.
SMS	Short Message Service.
SUT	System Under Test.
TSG	Technology Stage Gate.
URL	Uniform Resource Locator.
VPA	VerticalPodAutoscaler.
VU	Virtual User.
WSGI	Web Server Gateway Interface.

Glossary

3-D Secure (3DS)	E-Commerce authentication protocol. Enables secure validation of card transactions.
Caixa Geral de Depósitos (CGD)	Portuguese state-owned banking corporation.
acquirer	A Bank or Financial Institution that processes credit and debit card payments.
backlog	Prioritized task list for the development team.
fintech	Refers to *financial technology*. One of the digital world's fastest expanding industries.
issuer	Financial institution that is able to offer the card to consumers for the major card schemes.
Jira	Atlassian's proprietary issue tracking software, which enables agile project management.
K8s	Kubernetes, open-source system for automating deployment, scaling, and management of containerised applications.
payment gateway	Tool that securely validates the customer's credit card information and transmits it to the payment processor.
scheme	Central payment network that uses credit and debit cards to process payments.
Scrum	Project management framework.
settlement	The last stage of the payment lifecycle, in which the merchant gets funds for a transaction (or set of transactions) with a client.
turnover	The entire amount of sales made in a business within a specific time period. It is also known as income or gross revenue.

Chapter 1

Introduction

Payment processing, as regulated by the Payment Card Industry (PCI), can only be performed by a certified acquirer, as handling sensitive cardholder information is done securely if Data Security Standards (DSS) are followed. Payment solutions that are efficient, secure, and reliable are what fuel business expansion and success.

This thesis will be focused on the service provided by this payment acquirer[1], specifically its components ability to handle hundreds of millions of transactions per day without incident, as rapid and secure payment processing can be what assists businesses to better manage and develop their operations.

1.1 Context

In an era in which technology is pervasive in our lives and physical money is increasingly obsolete, merchants are compelled to provide methods of accepting payments via cards or mobile devices, known as Point of Sale (POS) terminals.

In this circumstance, a major issue arises for merchants, particularly small and medium-sized ones: with so many rivals in the financial sector selecting the one that best suits their needs has become complicated. The defining characteristic they are looking for is, in general, prompt acceptance of funds from the customer and settlement of their earnings at the conclusion of a specific time period (can be done daily, weekly, monthly, upon request or agreed). However, the presence of complex pricing structures, hidden fees, and long-term contracts makes it much more difficult for these relatively small companies to be successful and survive in the market.

As a result, companies operating in the payment acceptance and processing industry must reshape their proposal in order to make it as transparent and intuitive for the merchant as possible, without sacrificing the quality and efficiency that they expect, while also making it more appealing than the offers of their close competitors.

SaltPay is a Fintech firm with a worldwide presence, that was established in 2019 whose goal is to create payment and software solutions to assist small and medium-sized companies expand their operations. Within these options is the acceptance of payments in a straightforward, secure, and trustworthy manner using the most recent POS terminals.

The goal of developing load test plans for the various components that comprise the SaltPay system is to examine the data acquired in order to provide potential solutions for the system's

[1] An Acquirer is a financial institution that connects the merchant with card payment networks. A more in-depth definition can be found in chapter 2.1.2.

overall improvement and scalability. As the number of merchants that use this product grows, the various components of the system must be able to manage a growing number of transactions on a daily basis.

Given the complexity of the overall system, the load test plans will concentrate primarily on the Payments Gateway as well as the Acquiring Host in order to maximise the response capacity of these services. As shown in figure 1.1, all transactions performed by a merchant via its terminal are routed to the Payments Gateway, which is responsible for initiating calls to the various systems involved.

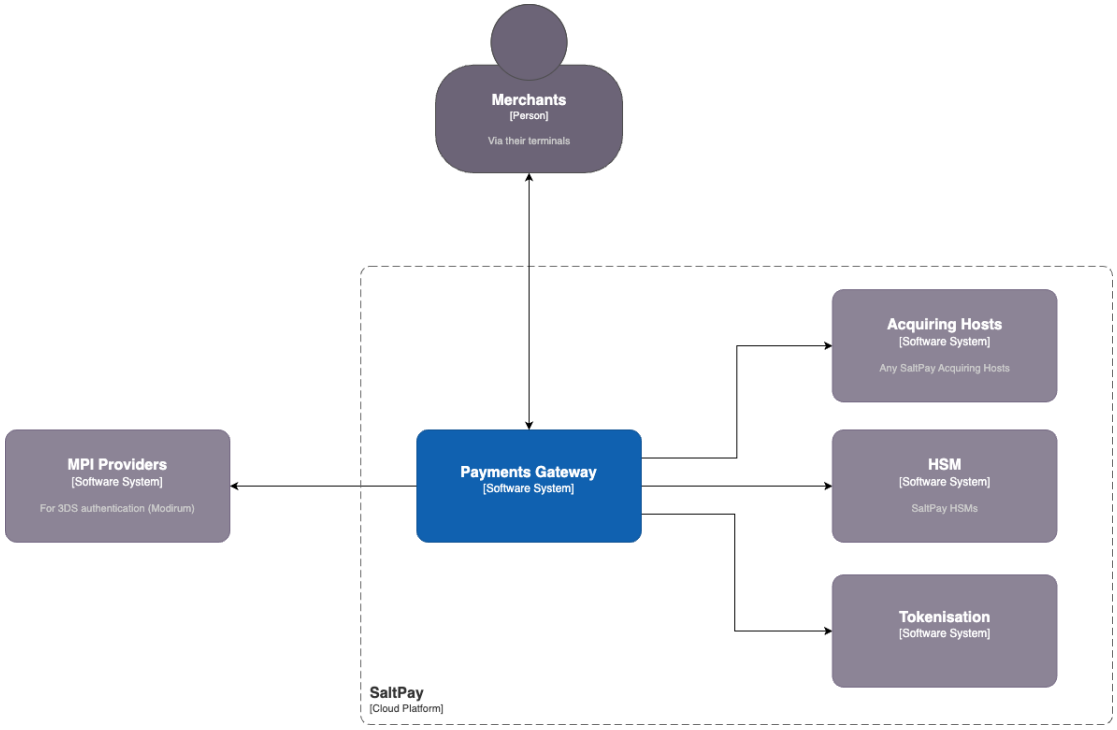

Figure 1.1: SaltPay main services architecture

By accepting a payment through its terminal, the merchant transaction request is sent through the Payments Gateway. The Payments Gateway is the internal gateway that is used to process transactions and perform all of the requests required in a transaction lifecycle, such as tokenisation of sensitive information, Hardware Security Model (HSM) calls to decrypt/encrypt cardholder data, Merchant Plug-In (MPI) calls to request 3DS authentication, and the acquiring host itself, which will facilitate transaction processing between the merchant, the customer's bank, and the card network. When a transaction is authorised or rejected, the result from the customer's bank and card network is sent back to the Gateway, which displays the appropriate response on the terminal.

Whenever a merchant wishes to perform a transaction using its terminal, it will typically input the amount to be charged on the touchscreen, and the customer will have to use its credit or debit card to complete the payment through the chip, contactless capabilities, or magnetic stripe.

Although the entire transaction seems to be transparent to both the client and the merchant, there are various requests to be performed behind the scenes. Risk analysis, input parameters validation, message transformation, data enrichment, and tokenisation of sensitive parameters are some of the several requests performed within the context of a transaction that

add latency to the overall process. A more comprehensive overview of these requests can be seen in figure 1.2.

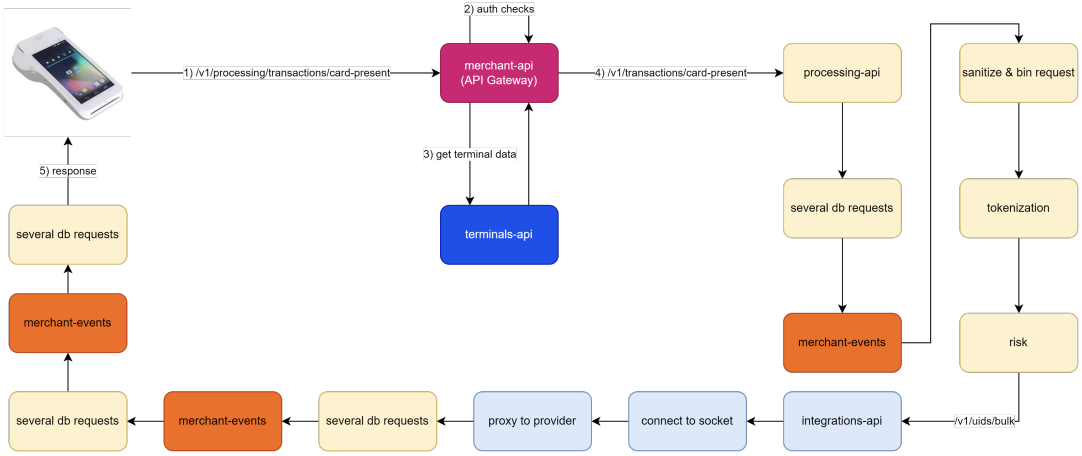

Figure 1.2: SaltPay transaction flow

The purpose of this dissertation is to improve the transaction processing time by the system that makes up SaltPay's product, namely the Payments Gateway and Acquiring Host, by determining where latency is introduced. It is also intended to analyse ways of allowing scalability as the number of transactions processed increases.

1.2 Motivation

Local commerce is one of the most popular shopping alternatives for a country and its citizens, owing to its convenience, fewer customers than traditional supermarkets, and closeness. These companies are important not only to the economy, but also to the community. At the moment, it is estimated that 99% of European businesses are in this category and may thus be deemed crucial for the economy[2].

Considering the organic shift from physical cash to the usage of credit and debit cards, as well as contactless alternatives, these small businesses are almost forced to provide a form that allows them to receive payments through these methods - or risk becoming unable to continue in operation. To this purpose, there are numerous options available on the market, which seek to allow these merchants expand their business, where the SaltPay solution is included.

Therefore, and with relation to the current dissertation, it is intended to assess the limitations of the solution offered to small and medium-sized local business owners. In other words, the assessments will emphasise on SaltPay's internal payment gateway, which is a critical point of interaction for terminals, in order to enhance and promote the solution's successful growth.

As previously stated, small and medium-size companies account for nearly all European businesses and about 70% of European jobs. This initiative arose in this organisation as a result of the drive for global spread and the ambition to become one of the most recognisable rivals in the market. It is beneficial for the organisation since some of its structural components

[2]https://single-market-economy.ec.europa.eu/smes_en

will be studied and improved, as well as for its clients because they will now have a more competitive and whole solution in their hands, reducing the need for regular support service.

In reality, competition exists in this expanding business segment. In the case of Portugal, Marc Bale de Jesse, the previous European Central Bank's Director General of Payments and Market Infrastructure, openly criticised and accused the organisation SIBS of having a monopoly in the payments industry (Brytfmonline 2021; O Jornal Económico 2021), given that it is desired to develop pan-European solutions.

In this circumstance, alternatives like SaltPay, which has been presented throughout this introduction, are attractive. Its global influence and desire to break the monopolistic tendency that prevails in certain countries, as well as the convenience and pleasant onboarding experience provided to new merchants, position it as a powerful rival and strengthen its market presence.

As a result of the foregoing, it is expected that all of the services that allow merchants to accept payments through their terminals to also be able to accommodate the increasing number of merchants transacting daily without any limitations.

For those reasons, it is critical to examine the platform's overall limits in order to determine which expansion strategies to employ if these are surpassed, and how to maintain the systems consistent while keeping the same processing capabilities as previously.

1.3 Objectives

The tests that will be developed will have the primary goal of analysing the present constraints of the Payments Gateway and the Acquiring Host in order to provide suggestions for system enhancement and scalability.

The findings will allow for enhancements to the existing system, making it more resilient and able to bear the additional load anticipated as the number of users increases.

The following are the primary goals to be attained:

- Creation of load test plans for the various Payments Gateway sub-systems, more specifically its core - the Processing API, which is responsible for creating transactions - and all the components that interface with the latter.
- Identification of high-latency spots throughout a transaction's lifecycle.
- Assessment of improvement points in the synchronisation across Payments Gateway sub-systems, performed through the consumption of transaction events.
- Identification of the system's various components' limitations.
- Suggestion for a solution to minimise the latency brought into the SaltPay system during the lifecycle of a transaction by the many components that communicate with Payments Gateway or by Payments Gateway sub-systems.
- Proposal for an approach to improve existent synchronisation between Payments Gateway sub-systems and transaction event indexing.
- Proposal for a way to automate the system's scalability in consideration of the predicted increase in traffic.

1.4 Value Analysis

SaltPay is a one-of-a-kind solution for small and medium-sized local merchants. After analysing the existing market and the payment service model, it was reasonable to assume that it is now dysfunctional. Complex pricing systems, hidden fees, and lengthy contracts make it difficult for small businesses to operate on a daily basis when it comes to accepting payments.

Given that efficiency drives local businesses and helps them to adapt promptly to changing customer needs, there was a need to optimise the existing system offered and guarantee that it is capable of fulfilling a larger demand on its services.

The solution described in this thesis intends to evaluate the existing boundaries of the various system components in charge of payment processing inside SaltPay's service, notably the Payment Gateway and the Acquiring Host. Furthermore, automated scaling solutions will be investigated, so that as transaction processing increases, the likelihood of an incident or downtime is mitigated.

The alternative approaches proposed will make it easier for the people who participate in the entire support process - from Customer Support to the Developers responsible for investigating and resolving issues with the services - as well as increase the quality and reliability of the services currently provided, offering merchants with a better service.

The value analysis of the solutions will be discussed in depth in the third chapter.

1.5 Project Methodology

The project described in this document was carried out in accordance with the Scrum framework, which is a component of the Agile movement and describes a method of delivering value in a collaborative and incremental manner (Schwaber and Sutherland 2020).

After a task is created, its description is clear, and a time estimate for that work is defined, it is possible to add a new feature, solve an issue that was discovered, or change the features that have already been implemented. Because SaltPay utilises Jira (Atlassian) to better manage all of the teams' tasks, the time spent on them, and the teams themselves, it was also the software chosen to manage all of the tasks linked to this project.

To enable full traceability, all modifications to the teams' stack necessitate the creation of a particular ticket in the project's board. Typically, integrating new features and performing direct modifications to the core behaviour necessitates a meeting with the Head of Product to ensure that all ideas are aligned, the new changes would benefit the customers, and that new tasks can be planned and integrated into the team's current workplace. Fixing bugs that are already in the various services and impacting customers at any level receives top priority and is completed as soon as possible to minimise any outage or disruption in regular processes.

1.5.1 Development Process

As stated in the previous section, the project was constructed in an iterative and collaborative manner, adopting a Scrum methodology. Several procedures, strategies, and approaches can be used within this framework, according to Schwaber and Sutherland (2020). As a result,

it is feasible to conclude that the technique used was not a rigorous Scrum, but rather a flexible Scrum, in order to fulfil the objectives of this project as best as possible.

At first, in order to establish the initial tasks and epics to be worked on, various brainstorming sessions were held with the team responsible for the Payments Gateway to better grasp the primary principles behind this project. The epics would be formed after analysing the outcomes of those sessions, and the primary goals would be broken down into smaller, achievable tasks.

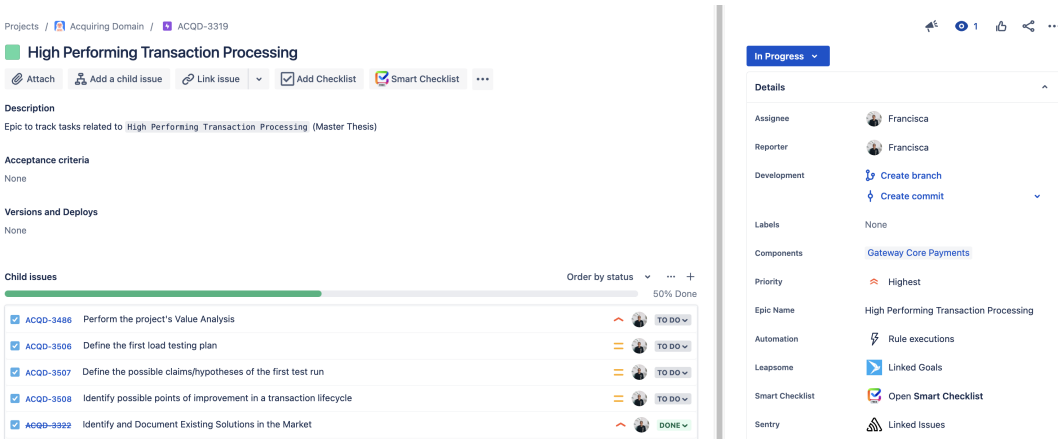

Figure 1.3: Epic created to track tasks related to this dissertation

The Epic illustrated in figure 1.3 was designed to keep track of all tasks linked to the development process of this dissertation. Some of the tasks belonging to it are identifiable, many of which are related to researching and reviewing current academic topics on the area, while others are related to the development and implementation of test plans. Every task has a description, as well as a task reporter (the user who created the issue), assignee (the person who will be handling this issue), priority, and story points, which are examples of the details that a task might include.

Following that, at the first Sprint Planning meeting, a dedicated board for the project was prepared, where it was specified that each sprint would last two weeks as well as the initial set of tasks that had to be achieved by then. The remaining tasks would be placed to the backlog and added to a future sprint.

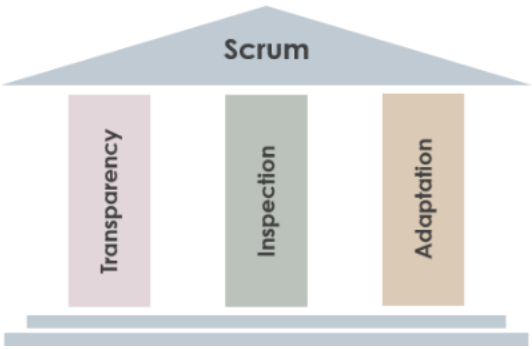

Figure 1.4: Scrum pillars (from: VisualParadigm (2022))

As the project expanded over time, as its development and design proceeded, additional business aspects would ultimately become tasks. Problems that arose as time went by would likewise be added to the current sprint as tasks with the greatest priority.

Scrum, according to Schwaber and Sutherland (2020), reveals the respective usefulness of present management, environment, and work approaches, allowing for adjustments. As a result, the efficacy of its four formal events is based on the empirical pillars of transparency, inspection, and adaptation.

Transparency, as the name suggests, strives to provide complete sight of the process to those doing or receiving the job, allowing for inspection. In turn, continuous and rigorous inspection of the artifacts generated helps to mitigate unwanted consequences and allows for adaptation - the result of a successful inspection in which the results produced depart from permissible bounds.

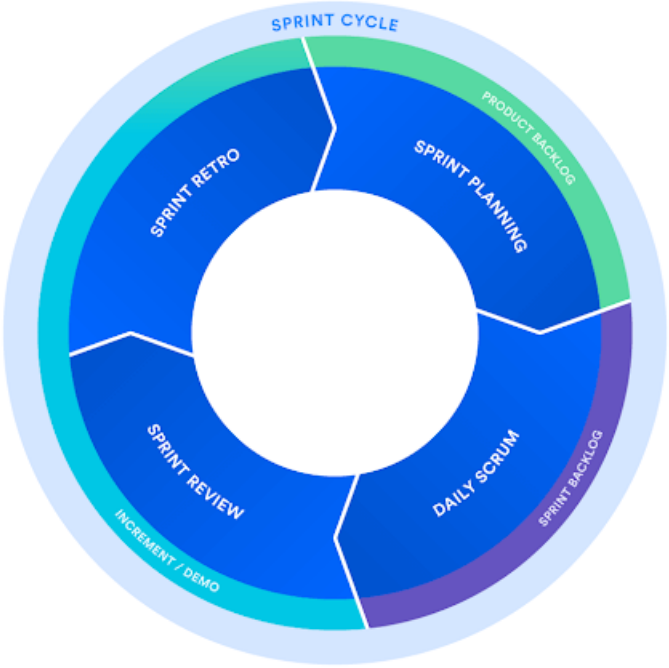

Figure 1.5: Scrum sprint cycle (from: Atlassian (2022))

As previously stated, there are four major events in Scrum, each of which represents a formal chance to apply the three pillars of this technique, all of which are incorporated inside the Sprint, as shown in the figure 1.5 above.

Every sprint begins with the Sprint Planning event, which addresses the purpose of the sprint, what can be accomplished in it, and how the planned tasks will be completed. To create this plan, the complete Scrum Team works together.

The Daily Scrum is a brief daily meeting for developers in which they often refer to what they worked on the preceding day, what their strategy for the present day is, and what issues

they are encountering. This meeting provides an overview of the progress made toward the Sprint Goals, as well as allows for adjustments to the backlog as needed.

The Sprint Review is the Sprint's second-to-last event, during which the Sprint's achievements are examined and the future actions are collaboratively discussed. It typically involves the Scrum Team as well as the key stakeholders.

Finally, strategies to improve the quality and effectiveness of the Scrum Team are planned during the Sprint Retrospective. The most significant improvements are handled as quickly as feasible, and they may be added to the backlog to be addressed in the following Sprint. This event brings the Sprint to a close.

One of the primary advantages of using an Agile approach is the adoption of a continual delivery mindset, which contributes to the attainment of the objective and the refinement of the customer experience on a regular basis.

This allows for continuous value generation through adaptive solutions for difficult challenges, as well as modelling the product based on client requirements and demands, which may lead to cost reduction over time.

1.5.2 Version Control System

Version control refers to a system that can record and retain a history of changes to a file or collection of files (Chacon and Straub 2022), allowing specific versions to be recalled later.

Since all the repositories with the source code of the main focus of this work - the Payment Gateway and Acquiring Host services - are located in GitHub, it will be the tool used as the version control system for the thesis.

1.5.3 Workflow

Every team that wants to effectively manage their development process needs a well-defined and disciplined development workflow. Working within a team may be rigorous and exhausting, especially if no mechanism is in place for evaluating, verifying, and, most importantly, preventing faulty code from reaching the production environment is in place.

The Acquiring domain development method considers the presence of three key environments: the local environment, the staging environment, and the production environment.

The first, as the name suggests, relates to the environment that each developer must have configured on their own computer. The same programmer develops and tests the changes to the source code locally before committing it to the project repository and opening a Pull Request (PR) so that it may be added to the code base.

After the PR has been reviewed, tested, and accepted by other developers in their local environments, it may be merged and the deployment process can begin. To begin, the developer must create a tag for the new Release Candidate (RC) version of the service in question. The developer may then initiate the deployment of the RC to the staging environment, where it can be tested.

Once the RC has been successfully tested and no unfavourable outcomes have occurred, the new version of the service may be successfully deployed to the production environment. This environment is accessible to all clients that have a valid account under the domain and perform transactions on a daily basis.

1.6 Document Structure

The report is broken down into 6 chapters. Aside from this introductory chapter, Chapter 2 is divided into 2 main sections - where the first one discusses the context of this work and some relevant details to the system's final solution; the State of the Art, which is the second section, presents the research work on the technologies currently available in the market and other approaches to the issues that will be discussed and solved in this paper.

Chapter 3 will discuss the Value Analysis of the solution proposed in this report.

In Chapter 4, detailed test plans that will be used to identify the existing constraints of the services will be outlined and carried out. With the outcomes of these studies, it will be feasible to hypothesise and test alternative solutions to those known limitations. The design of the produced solution will be described, as well as a comprehensive comparison with alternative techniques by referring to the advantages and drawbacks of each one, in accordance with Computer Engineering best practices.

A more extensive discussion of how the solution was developed, as well as the considerations made, is provided in Chapter 5. It also includes the results of the tests and analysis performed for the solutions offered.

Lastly, chapter 6 finishes the report by providing a summary of the work completed to date as well as planned work for this project.

Chapter 2

Context and State of the Art

2.1 Context

2.1.1 The Payment Industry

Card-based payments, whether performed online or in-person at POS terminals, are among the options that are available to the great majority of businesses. Matter of fact, recent data found that this segment has been steadily growing over the past few years, with predictions that the amount of "cashless" transactions will triple by 2030 (PWC 2021).

The Covid-19 virus epidemic has also contributed to the transformation of the traditional paradigm of commercial transactions, that used to employ cash as the preferred mean of payments and individual retailing to online businesses, boosting the adoption of cards for transactions. Aside from its simplicity of use, notably when it comes to contactless payments, the additional levels of security provided as well as the high acceptance rates are appealing qualities for its customers.

According to Banco de Portugal (BdP), there has been a significant growth in payments made, both in volume and in value, as a result of the economy's recovery. In addition, electronic payment instruments accounted for 99.5% of all payments handled by Portuguese Interbank Clearing System (SICOI). Payment cards are still heavily favored among Portuguese customers, accounting for 86.5 percent of total SICOI payments. The 9.4% rise in the number of POS, when compared to the same period the year before, drove the growth in card transactions (Banco de Portugal 2022).

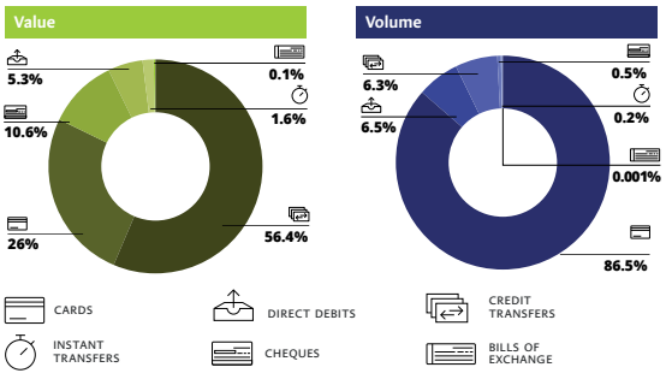

Figure 2.1: Use of payment instruments in 2021 (from: Banco de Portugal (2022))

Considering the European perspective, the cumulative number of payments processed inside the European Economic Area (EEA) using electronic instruments increased by 12.5 percentage points in 2021, when compared to the same period the previous year. Analogously to what occurred in Portugal, payment cards accounted for the majority of these transactions, contributing roughly to 49% of their total (European Central Bank 2022).

The historical data points documented in recent years, along with the continuous technological progress perceived, have contributed to the growing interest in this economic sector. As a result, particularly with regard to the European initiatives to establish a pan-European payment solution, competition among the various Payment Service Providers (PSPs) has intensified, leading to increased commitment on their side to provide their consumers with the finest and most comprehensive service available.

This project focuses specifically on this topic, and its primary objective is to investigate the platform's present limitations as well as provide alternatives for enhancing the reliability and availability of the SaltPay service, considering the increasing number of merchants that have adopted their solution.

2.1.2 Key Context Definitions

A number of players are involved in the various actions associated with payment processing (European Central Bank 2022). To completely understand the context, important conceptions must be specified.

- **Acquirer:** An Acquirer is a financial institution that connects the merchant with card payment networks.

  As it is a broad concept, the term can be applied in different scenarios:

  - Entity that provides terminals.
  - Entity that owns merchant deposit accounts and to whom data for a card-present transaction is delivered.
  - *Card-Present/POS Terminal Transactions:* The entity to whom the information required to conduct a card-present transaction is transmitted.
  - *ATM Transactions:* Entity that provides the cardholder with tangible money.

  Can also be referred as *Acquiring Bank*.

  SaltPay is an example of an Acquirer.

- **Acquiring Host:** An Acquiring Host is a software system that facilitates transaction processing, which involve the merchant, the customer's bank, and the card network.

  It is responsible for communicating with the card network, validating the client's payment card, and requesting transaction permission. Upon receipt of this authorisation, it sends a confirmation to the merchant's POS terminal and executes the transaction, transferring funds to the merchant's bank account.

  Solar and *Way4* are to be regarded the acquiring hosts under the SaltPay domain.

- **Card Scheme:** A Card Scheme is a central payment network that processes payments using credit and debit cards and administers transactions in accordance with a set of

procedures, regulations, and agreements that permit cardholders to use their cards with external parties.

There are two types of card-schemes, as per European Central Bank (2022):

- *Three-Party Scheme:* Involves the card scheme itself (acting as an Issuer and Acquirer), the cardholder and the card acceptor. American Express in an example of this type of schemes.
- *Four-Party Scheme:* Involves the Issuer, the Acquirer, the cardholder and the card acceptor. Examples of four-party schemes are Visa and Mastercard.

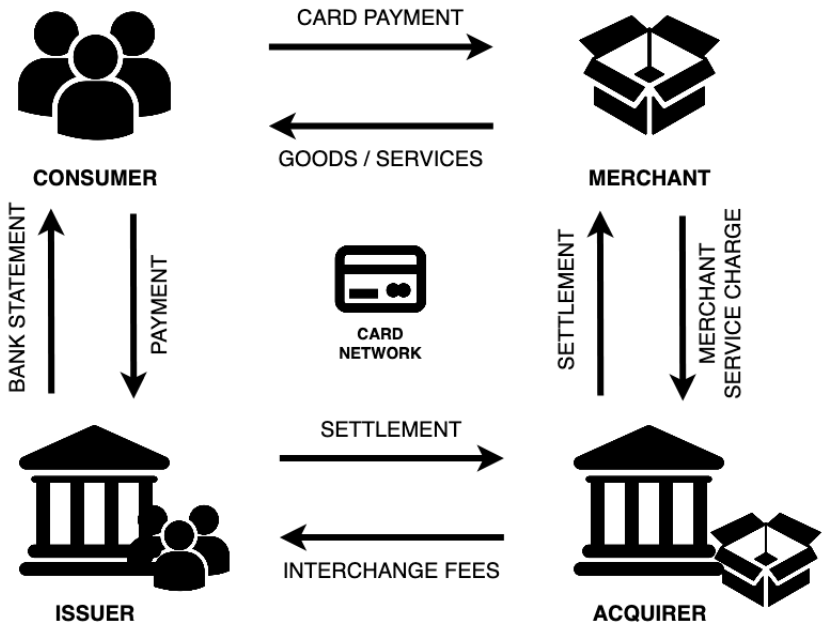

Figure 2.2: Four-Party Scheme (adapted from: Switch (2021))

- **Card Issuer:** A Card Issuer is a financial institution that offers payment cards to individuals, which enable them to pay for products and services.

  Issuers also authorise transactions (POS terminals or Automated Teller Machines (ATMs)) and guarantee payment to the acquirer. In *three-party* schemes, the Issuer is the scheme, and in *four-party* schemes, the Issuer is a PSP that is a member of a card scheme.

  Can also be referred as *Issuing Bank*.

- **Payment Service Provider (PSP):** A Payment Service Provider is a third-party entity that links a payment service user (often a merchant) to the financial infrastructure, enabling them to accept payments (such as credit and debit cards, direct debit, bank transfers).

  In the European Union (EU), there are six categories of PSPs: credit institutions; electronic money institutions; post office giro institutions; payment institutions; the European Central Bank (ECB) and national central banks; EU Member States or their regional or local authorities.

  Can also be referred as *Merchant Service Provider*.

- **MPI Provider:** A MPI Provider is an auxiliary software module that aids with 3DS verifications. To combat credit card fraud, the vast majority of online transactions employ this protocol, which might be provided by third-party certified suppliers.

Although it is true that there are numerous parties and concepts in the payments sector and processing scene, the ones discussed above will be emphasised the most throughout this dissertation. As a result, it was opted not to describe any further supplementary concepts.

2.1.3 Business Concepts

It is necessary to provide certain terminology, related to business specific concepts, in order to fully comprehend the workflow within the architecture of the services provided by SaltPay, as illustrated in figure 1.1 and highlighted throughout this dissertation.

- **Merchant:** A Merchant, as defined by SaltPay, is a Small and Medium-Sized Business (SMB), which constitutes the target demographic for SaltPay's services and products.

  SMBs are often defined as having fewer than 250 employees and a turnover of less than €50 million.

  Only merchants that rely strongly on card-present transactions (i.e., those who require a POS terminal) will be addressed in the context of this thesis.

- **Payments Gateway:** The Payments Gateway is the internal gateway, whose primary focus is transaction processing. The team team inside Saltpay's Acquiring Domain that is in charge of creating and maintaining the internal gateway is the Gateway team.

  The gateway has the following high-level characteristics:

  - A platform that can readily integrate with any acquiring host or local scheme;
  - Provides appropriate Application Programming Interfaces (APIs) for transaction processing performed by any interested party (terminals, e-commerce, platforms, and so on);
  - Publishes transactional events in a standardised format, allowing any internal client to obtain all of the information about every transaction that passed through it.

  There are various services that constitute the entire gateway. However, the fundamental emphasis of this thesis will be on Processing-API (the "core" of the gateway), Merchant-API, Transactions-API, and Integrations-API.

- **Transaction:** A transaction, inside the Payments Gateway context, may be viewed as a collection of elements that together constitute the whole payment lifecycle.

  This lifecycle contains various phases that should be noted.

  - *Charge:* The Charge is the transaction's initial lifecycle element, and it communicates the merchant's desire to conduct a financial transfer (either pulling or pushing into a customer's account). The payment method and provider are chosen during this stage (that is, the integration to use).
  - *Instrument:* The second stage in the lifecycle of a transaction, and where the exchange of funds officially begins. It contains all of the required information

to authenticate the client in one or more payments using the desired form of payment.

- *Payment:* The funds capture happens in the Payment, and this element represents the transaction authorisation from a provider. The Gateway can initiate payments for specific transaction flows.
- *Reversal:* Reversals are a method of cancelling a transaction prior to it having cleared. It is possible to invalidate an authorisation request, or a payment or refund that has not yet been completed.
- *Refund:* Refunds, as opposed to Reversals, occur only after the payment has already been settled. They are always initiated by the merchant, and it is possible to entirely or partially refund a payment.

Every element in the lifecycle is linked to the preceding one by a unique identity (ID). A basic sale transaction does not require all of the elements due to the possibility of no Reversals or Refunds.

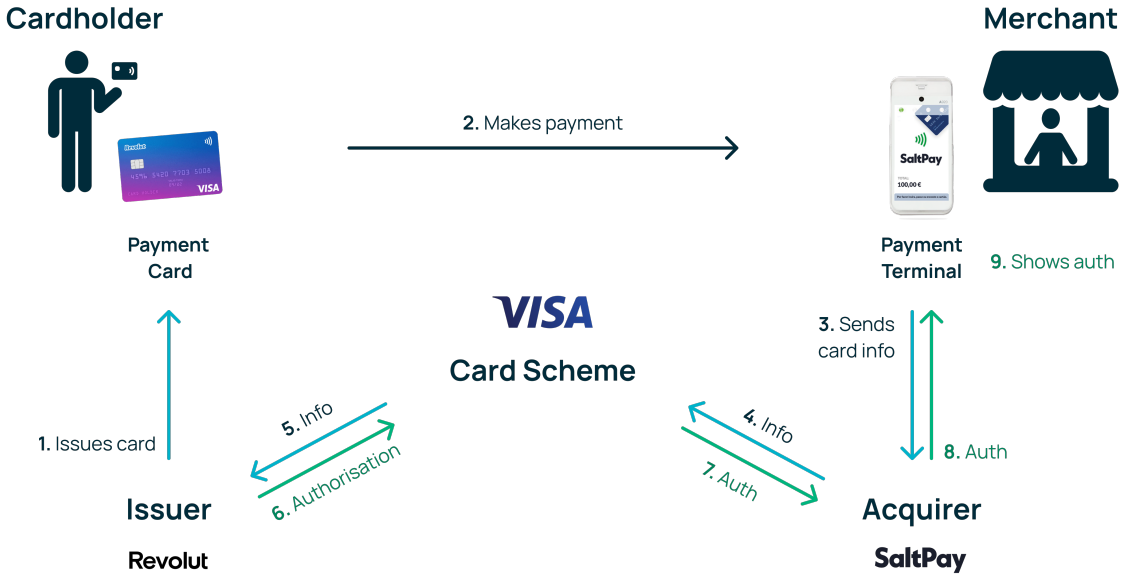

Figure 2.3: Card Ecosystem (from: SaltPay Learning Team (2022))

2.2 State of the Art

2.2.1 Existing Solutions

The system under testing and refinement in this dissertation is a system that is internal to SaltPay. As a result, assessing the presence of direct competitors to it is very challenging, as all companies in the industry have a unique method of processing transactions, wrapped up with their own internal nuances.

However, given the existence of considerable market competition with similar offerings to the one that SaltPay aims to provide, the subsequent paragraphs will provide a concise overview and comparison of some of them. It should be emphasised that all information

given originates solely from the company's websites, due to the lack of means of determining their internal implementations.

FlutterWave

Flutterwave (2023) is a Nigerian fintech company whose major goal is to assist small local merchants in growing their businesses. Their target market, according to research, is Africa, particularly Nigeria, but they also operated in other locations such as the United States and the United Kingdom. This solution is available as a Web App and a Physical App (in POS terminals), and it offers the following primary services:

- Process Transactions (online and in-person, as well as payouts, payment links)
- Integrated checkout
- Online store creation
- Create invoices
- Card issuing
- Loans
- Simplified process of registering and incorporating a business
- Fintech as a Service (FaaS)

Flutterwave's technology are unclear since this type of information is not disclosed on the company's webpage.

Yoco

Yoco (2023) is a South-African startup who develops mobile POS card machines and their systems, specifically aimed to help small South-African businesses grow. Aside from various card machines, the key services it provides are as follows:

- Process Transactions (online via their gateway and in-person with the terminals, as well as gift vouchers, payment links)
- Business Management App
- Cash advance
- Business-aimed Portal

Yoco's technology stack is also undisclosed.

VivaWallet

VivaWallet (2023) has a unique presence on the European continent. Its market segment comprises businesses of all sizes, and it strives to offer sophisticated card acceptance services through its unique solutions. Its primary services include the following:

- Process Transactions (in-person and ecommerce solutions)
- Business Account and Dashboard

- Mobile App (with a digital debit card)
- Independent Software Vendor (ISV) Program

VivaWallet, as with all of the main competitors mentioned above, does not expose the technology it implements.

CGD

CGD (2023), a state-owned financial organisation, is Portugal's biggest bank. Despite its presence in other countries, CGD provides in-person solutions that are solely available in Portuguese territory. The following are the key services it provides, as outlined on the POS terminals web page:

- Process Transactions (using a POS terminal, the mobile app, or via their payment gateway)
- POS Terminal statistics
- Tax Free support

The technological stack for CGD is unknown. However, being a product tailored for the Portuguese market, all transactions are processed by SIBS, along with the dashboards provided for seeing terminal statistics.

2.2.2 Comparison between solutions

Table 2.1 presents a concise comparative assessment between the SaltPay solution and some of the existing businesses in the industry, as outlined in the previous subsection.

Table 2.1: Comparison between existing solutions

	SaltPay	**FlutterWave**	**Yoco**	**VivaWallet**	**CGD (Financial Institution)**
Card-not-Present Transactions (On-line)	✓	✓	✓	✓	✓
Card-Present Transactions	✓	✓	✓	✓	✓
Global card acceptance	✓	✓	✓	✓	✓
Global availability	✓	×	×	×	×
Fixed contract	×	✓	×	×	✓

As evidenced from the table above, the existing competition examined is, overall, nearly equivalent to one another. Aside from the transaction costs and the fixed extended contract terms, there aren't many expressive distinctions that make a client prefer one over the other based on the information retrieved from the competition's websites.

Among the most common reasons merchants express concerns to SaltPay, or any other competitor, is the connection to internal services - whether due to network failure or service failure -, the capacity to handle large amounts of transactions efficiently, and the prompt accessibility of funds in their accounts.

As a natural result, and given the primary purpose of serving the merchant by delivering the best services available, developers have been prioritizing these issues.

2.2.3 Relevant Technologies

One of the primary objectives of this dissertation is to define load and performance test scenarios and interpret their outcomes, in order to propose suggestions to improve the system's scalability and, consequently, its reliability. Therefore, it is essential to look into relevant technologies in the field of load and performance testing.

Load testing a web service is essential since it indicates how well the system will operate whenever a large number of users attempt to interact with it at the same time. Menasce (2002) suggests that load testing could be interesting when a "significant traffic increase is anticipated".

Performance testing plays an important role in complement to load testing, since it "evaluates performance related aspects of a software system" (Z. M. Jiang and Hassan 2015), such as the system's responsiveness, stability, reliability, speed, and resource utilisation.

There are multiple tools available for performance and load testing, some with support for only one programming language, others with multi-language support, Command-Line Interface (CLI) only tools, and tools with Graphical User Interfaces (GUIs). Menasce (2002) states that "load-testing tools are quite useful. They can generate scripts for a few virtual users to measure service demands". For these reasons, the following paragraphs will provide a short description of some of the most widely used performance and load testing tools currently available.

Apache JMeter

Apache JMeter is the most popular tool in regards to performance testing. It is a "100% pure Java, open-source software", that was "designed to load test functional behavior and measure performance" (Apache Software Foundation 2022).

Aside from web application testing, Apache JMeter can perform different testing tasks. Since it is a Java program, it can provide a Java-oriented system, which is useful for evaluating Java systems.

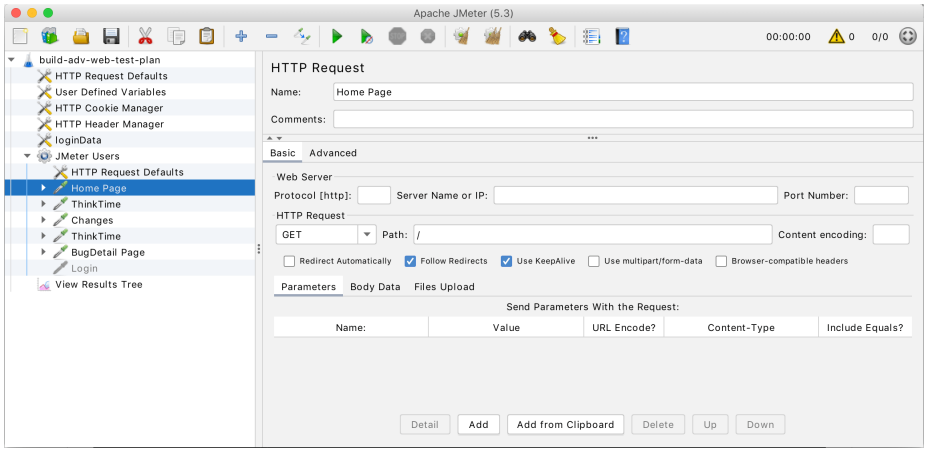

Figure 2.4: JMeter GUI (from: Apache Software Foundation (2022))

One of the most significant downsides of this testing tool is the challenge in using it in a large distributed test, apart from the Groovy syntax barrier and the complex GUI, that is

illustrated in figure 2.4. It has also been mentioned that when a certain limit is reached (often when executing computationally-heavy tests), the excessive memory usage produces issues for the Virtual Users (VUs) running the test (Apache Software Foundation 2022).

Locust

Locust is an open-source performance testing tool written in Python, described as "easy to use, scriptable and scalable" (Locust 2023). Its success is built on allowing developers to specify the behavior of their test users using Python code rather than a counter-intuitive GUI or a domain specific language that leverages callbacks or other mechanisms. Despite the fact that the test scripts are executed using its CLI, Locust offers a "web interface that shows the progress of your test in real-time".

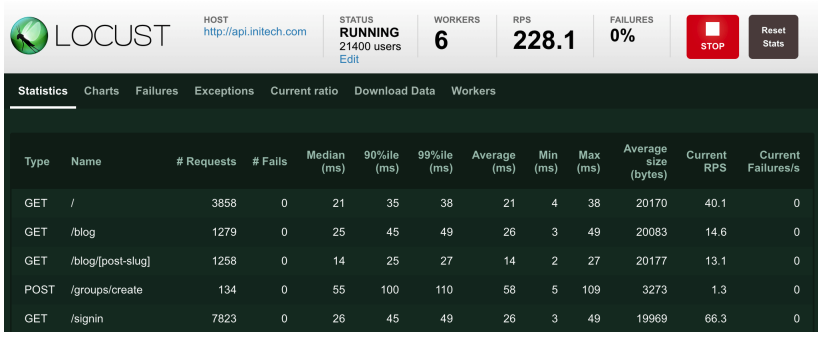

Figure 2.5: Locust GUI (from: Locust (2023))

Unlike JMeter and other tools, which rely on a thread-based architecture, Locust takes an event-based approach. This results in the consumption of less resources and, as a result, makes it significantly faster than its competitors.

Because the Gateway team develops the majority of its applications in Python, this tool has previously been utilised by its engineers in the past to determine how the system would react under a possible heavy load. The response received on its use was mostly positive, with the main negative aspect being the difficulty in increasing the number of co-routines, which limited the client from issuing many requests. This could have been due to hardware restrictions, as all tests were performed on personal computers, rather than in the cloud.

Vegeta

"Vegeta is a versatile HTTP load testing tool", that is written in Go. "It can be used both as a command line utility and a library" (Senart 2022). Unlike Locust or Apache JMeter, Vegeta does not require the user to install Python, Java, or any programming language in order to use the tool. It is solely required to install the tool itself before proceeding to test the HTTP service.

Despite the lack of a GUI, Vegeta includes a command that "outputs an HTML time series plot of request latencies over time", *plot*, as well as integration with tools created for the macOS terminal emulator iTerm2 that plot a report in real-time (Senart 2022).

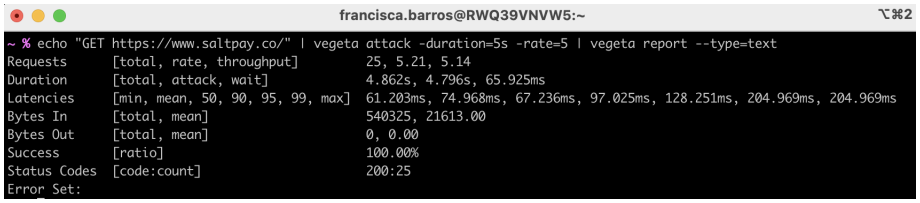

Figure 2.6: Vegeta command output

Figure 2.6 depicts and example of Vegeta's usage. The *echo* command is used first to give the Uniform Resource Locator (URL) to the following command, *vegeta attack*, which conducts the load test at 5 requests per second for 5 seconds. Finally, *vegeta report* presents the attack report as text.

The known limitations of this tool are mainly related to the conditions present in the machine that is being used for the testing, notably Central Procesing Unit (CPU), memory, or incorrect tuning of the system resource restrictions in file descriptors and processes (Senart 2022).

K6

K6 is developed in Go and has an integrated JavaScript engine that enables users to write testing scripts in JavaScript. Among its common use cases are load testing, browser testing, chaos and resilience testing, and performance and synthetic monitoring (Grafana Labs 2023b, K6 Documentation).

Despite the fact that K6 is a CLI tool, it is incredibly flexible, featuring interaction with third-party tools and extensions to broaden its possible use cases (Grafana Labs 2023b, K6 Documentation).

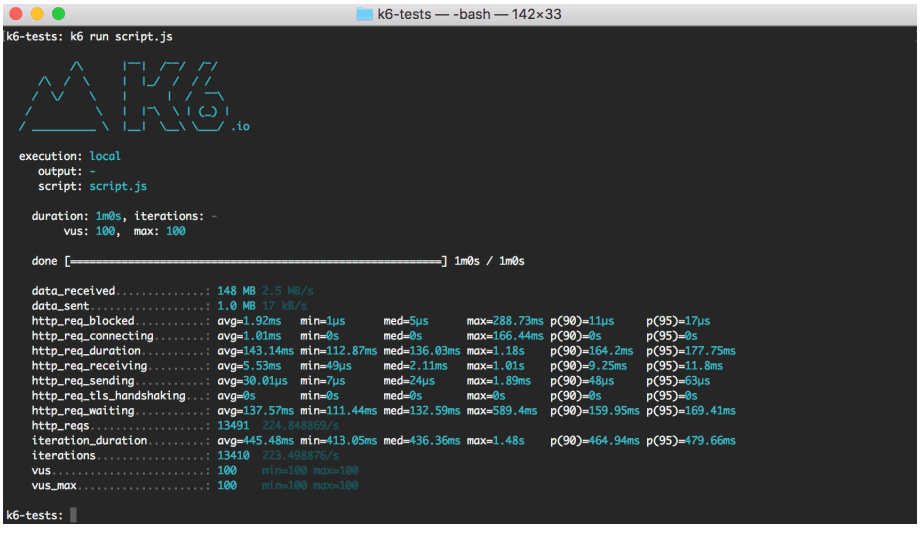

Figure 2.7: K6 Test results (from: Grafana Labs (2023b, K6 Documentation))

After a test script has done executing, a text report comprising numerous metrics, such as the median and average values, minimum and maximum values, 90^{th}, 95^{th} and 99^{th} percentile values, is presented in the user's terminal, as illustrated in figure 2.7. There are

extensions that allow the report to be generated in HTML format and made available on a local web page.

While looking for Locust alternatives, the Gateway team ended up turning to K6 to attempt to mitigate the drawbacks highlighted. Fortunately, the latter performed better than the former, allowing developers to inundate the services with more requests per second, making it the preferred tool for load and performance testing.

2.2.4 Comparison between technologies

As indicated by the preceding paragraphs, it is feasible to infer that the various available tools for the creation and execution of load and performance tests that were mentioned constitute candidates which could adequately suit the goal of this dissertation.

With the exception of when building and maintaining Java-based projects, the team's developers stated that they had no experience with Apache JMeter. Locust was the tool that had been used more frequently in the past to test the services since it is a Python framework and its interface gives real-time data which may be helpful.

However, according to Menasce (2002), "a load test is valid only if virtual users' behaviour has characteristics similar to those of actual users" because "failure to mimic real user behaviour can generate totally inconsistent results". Locust did have a lot of restrictions, and it did not allow engineers to adequately scale up the amount of requests done, leading to fairly consistent, and not particularly accurate, results regardless of when the test was run.

Considering that, Vegeta and K6 have lately been considered. These tools appear to overcome the constraints in the number of VUs spawned and presumably perform the test scripts without requiring as much resources as Locust.

When comparing the sample usage and base report generated by both tools, the former presents an approach strongly linked with the terminal and its commands, using several pipes for the execution of an *attack*, and a simpler final report, whereas the latter relies on a small JavaScript script in which the test plan is defined, and displays a more user-friendly report with more metrics.

Despite the fact that both tools are appropriate for the planning and execution of test plans, K6 was chosen because it is extremely flexible and the report generating plugins are better suited to the demands of this project. This tool will be used to carry out test plans that will be useful in assessing the platform's existing constraints as well as the suggested strategies to increase the system's scalability.

Chapter 3

Value Analysis

Creating value is not a simple process. According to Haksever, Chaganti, and Cook (2004) there does not appear to be an "emphasis on the universal aspects of the value creation", based on their analysis of a lot of research on the subject.

Value analysis can be defined as a systematic, formal and organised process of analysis and evaluation, that is applied to compare the function of products, solutions or services required to meet the demands or application needed by a customer or consumer (Rich et al. 2000). Identifying how a service or product adds value to its target market is crucial to improve it over time.

In this chapter, the value analysis process will be presented, based on the use of several tools and techniques, including the Innovation Process using the New Concept Development (NCD) proposed by P. A. Koen, Ajamian, et al. (2002), the value of the solution, the value proposition, the Business Model Canvas and Quality Function Deployment (QFD), suggested by Rich et al. (2000).

3.1 Innovation Process

New hardware and software continues to reach the market at an increasing rate as technology continues to reshape itself throughout time. Furthermore, the need for high-quality products and services to meet the general market requirements and interests is increasing. This desire and demand are frequently associated with innovation, which is a term strongly linked to change.

Quality is an important part of a solution's value given that it is one of the key attributes that enable both it and the business to grow and prospect. P. A. Koen, Ajamian, et al. (2002) presented a technique with multiple phases to describe this innovation process in order to secure its assurance and the creation of a quality product.

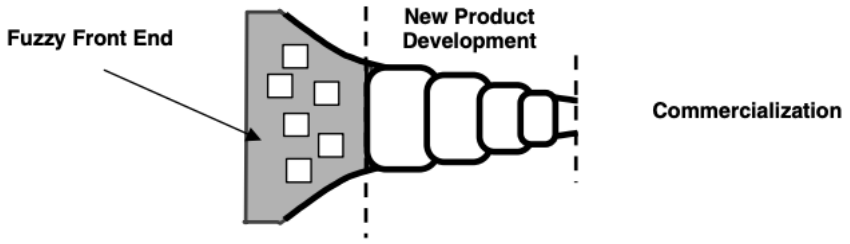

Figure 3.1: Innovation Process (by: P. A. Koen, Ajamian, et al. (2002))

As demonstrated in figure 3.1, the innovation process is divided into three stages: Fuzzy Front End (FFE), New Product Development (NPD) and commercialisation.

The first phase, the FFE, is perceived as "one of the greatest opportunities for improvement of the overall innovation process" (P. A. Koen, Ajamian, et al. 2002). Even though decisions made in this phase may influence which innovation possibilities are considered in subsequent stages, the FFE model can be difficult to follow and unpredictable, due to the lack of consistency in its description (P. A. Koen, Ajamian, et al. 2002; P. A. Koen, Bertels, and Kleinschmidt 2014).

To address these challenges and provide better comprehension of the previous model, P. A. Koen, Ajamian, et al. (2002) created another theoretical model, the New Concept Development (NCD) model, with the aim of providing an uniform explanation and characterisation of the fundamental components that comprise the FFE.

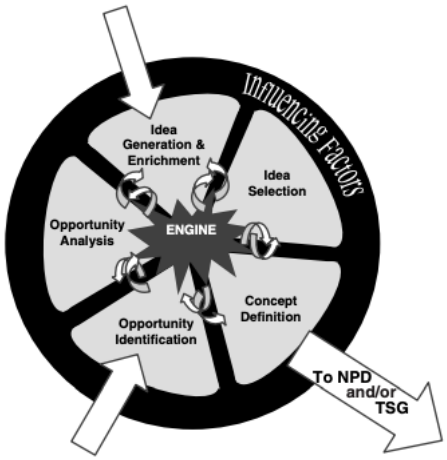

Figure 3.2: New Concept Development Model (by: P. A. Koen, Ajamian, et al. (2002))

The NCD model is composed of three fundamental pieces, as shown in figure 3.2 (P. A. Koen, Ajamian, et al. 2002):

- The **engine**, which represents the organisation's leadership, culture, and business strategy and drives the five major aspects of the FFE.
- The **five key activity elements** that comprise the business's unspoken area, which are *Idea Generation and Enrichment*, *Idea Selection*, *Concept Definition*, *Opportunity Identification* and *Opportunity Analysis*.
- The **influencing factors**, which include uncontrollable external elements such as the outside environment, that may have an impact on the whole innovation process.

According to P. A. Koen, Ajamian, et al. (2002), the projects begin at either opportunity identification or idea generation and enrichment, and then the concepts will leave the model and enter either the NPD or the Technology Stage Gate (TSG) process.

3.1.1 Opportunity Identification

The adoption of card-based payments, and consequently the usage of POS terminals, has been presenting a rising trend in the last years, as described in the previous chapter.

Interest in the fintech sector has also grown in recent years, therefore investment in this sector increased considerably (Mention 2019), as seen in figure 3.3. Technology advancements, shifting customer behaviour, and a greater emphasis on financial inclusion are some of the reasons of this rise.

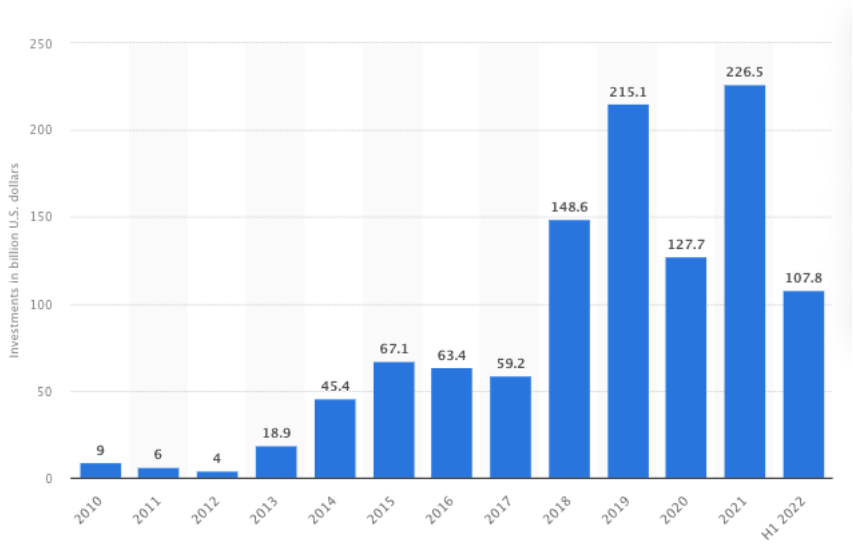

Figure 3.3: Investments into fintech companies globally, from 2010 to H1 2022 (from: Statista Research Department (2022))

The enhancement of an existing solution can assist the organisation in growing and expanding its operations as envisioned. Grafana, the observability platform utilised, will give insights into potential breaking points and where delay is caused in transaction processing. This way, it will be feasible to advise adjustments in these important areas, which may result in a reduced overall processing time, making the product more competitive and more adapted to market needs.

It is not assured that processing times will decrease, as the volume is expected to increase over time and it is impossible to eliminate all latency introduced to the current flow. Nonetheless, these factors will be included in the study to determine areas for development.

3.1.2 Opportunity Analysis

The opportunity identified in section 3.1.2 strongly indicates that it may be advantageous for SaltPay to differentiate itself from rivals and offer a distinct value proposition to its customers.

Previously covered in chapter 2, the payment card terminal solution industry is highly competitive, with other comparable rivals striving to compete for market share. As stated in a recent study by Tohang, Ramadhan, and Djajadiningrat (2021), "the emergence of FinTech ecosystems is one of the most important developments in the financial industries of the present age", and the statistics presented in the section above support this statement. The fintech sector's severe fragmentation exacerbates the demand for well-consolidated solutions with a large market share.

Therefore, innovation, availability and reliability are essential factors in expanding the number of merchants who subscribe to these types of services, and increasing their market value.

3.1.3 Idea Genesis

The "birth, development and maturation of the opportunity into a concrete idea", according to P. Koen et al. (2001), is represented through genesis.

This idea is centred on the needs of merchants that want to grow their operations and wish to implement a payment solution based on POS terminals.

To further grasp the concept, a set of questions it should be able to answer was established:

- How can the solution establish an even stronger market presence?
- How can transaction processing times be lower?
- When load increases, how can the system keep the same processing capability as before?

3.1.4 Idea Selection

Several software solutions for running the test plans designed to detect potential high-latency areas in a transaction's lifecycle, as well as improvement points in the synchronisation across all Payments Gateway sub-systems, are studied and analysed in chapter 2.

The results of the test plans and Grafana's request tracing function, which allows for real-time visualisation and analysis of the application's performance, will be used to answer the questions in section 3.1.3.

Given the nature of this dissertation, a variety of methodologies and technologies might be employed. After reviewing the aforementioned ideas, the best option was chosen.

3.2 Value

Value is an increasingly important notion in marketing, yet it is a wide concept whose definition varies depending on the circumstances (Bowman and Ambrosini 2000; Lindgreen and Wynstra 2005). Neap and Celik (1999) suggested that value can be defined as "the owner(s)'/buyer(s)' desire to retain or obtain a product".

The perceived value of a product or service is a subjective judgement for the customer, as different customers perceive different value for the same product or service (Ulaga and Eggert 2006). As such, perceived value may be viewed as a relationship between perceived advantages and perceived sacrifices. If the advantages outweigh the drawbacks, the customer will recognise the value in the product or service.

The desired solution should deliver value to its clients, the merchants, by reducing transaction processing times and maximising the reliability of the present services in a smooth and transparent manner.

Table 3.1: Benefits and sacrifices to the customer

Benefits	Sacrifices
Improved transaction processing times	Trouble identifying latency points
Readiness to deal with heavy use loads	System's scaling costs

Table 3.1 summarises the benefits and sacrifices of this project in order to determine its value to the client. Improved transaction processing speeds will result in higher merchant acceptance rates and, as a result, more sales. Regarding the ability to handle high use loads, this may lead to an increase in the number of merchants that already utilise this service. Nonetheless, the sacrifices should be kept in mind, even though they are mostly connected to the expense of using the service and its existing implementation constraints.

3.3 Value Proposition

Several authors define value proposition in different ways. According to Bagchi and Tulskie (2000), a value proposition is a declaration of advantages provided by an organisation to its customers, whereas Osterwalder and Pigneur (2003) define it as an overall perspective of the company's goods and services that represent value for a certain customer segment.

Customers who nowadays utilise POS terminal solutions to handle payment cards in their businesses, as well as those who want to adopt these kind of solutions, will benefit from the intended solution.

Aside from strengthening trustworthiness in this service, enhancing the reliability of the system while providing the same features as before is predicted to enhance the company's productivity because manual intervention will be less necessary when dealing with heavy load issues.

Reduced transaction processing times combined with a well-tested, consolidated, and proven service can be the difference between a merchant selecting SaltPay's solution over any other competitor.

3.4 Business Model Canvas

According to Osterwalder and Pigneur (2010) a business model explains the logic for how a company develops, distributes, and collects value.

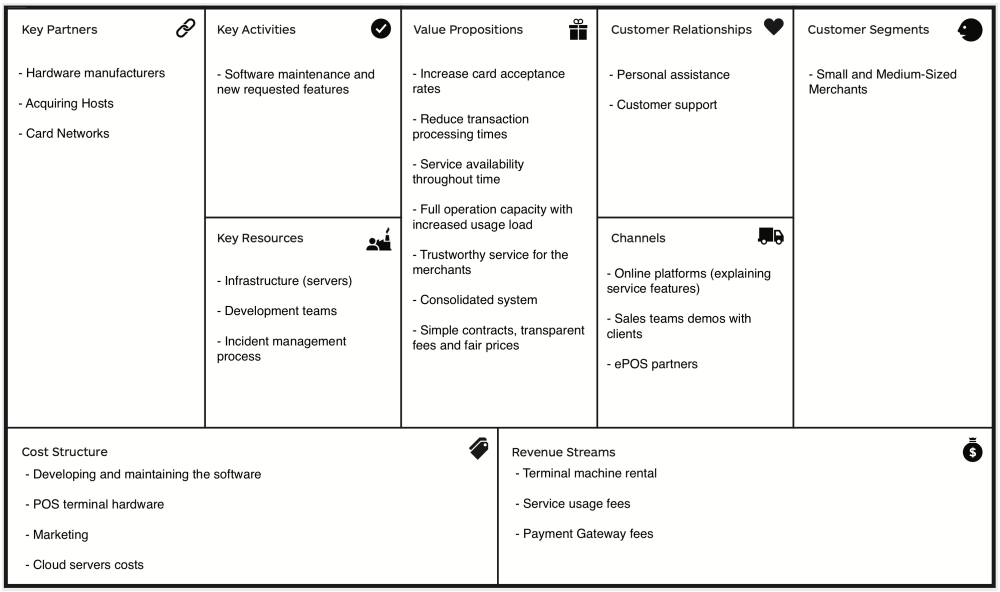

Figure 3.4: Business Model Canvas

Figure 3.4 depicts the Business Model Canvas designed to describe this project. The Business Model Canvas is a basic idea that provides a standardised vocabulary for describing and developing a business model, and it is a fundamental model for strategy planing. This model is made up of nine building blocks that may be separated into front and backstage.

The front stage refers to the component of the business that is immediately confronting the customer, and is constituted by the value proposition, customer relationships, customer segments, channels, and revenue streams.

At its core, the value proposition includes a number of critical components that highlight the benefits and value that SaltPay can deliver to its customers in particular. By providing these, SaltPay hopes to separate itself from competitors and give merchants a compelling incentive to use their services.

- **Increase card acceptance rates**: by increasing acceptance rates, merchants may extend their client base, enhance revenues, and lessen the risk of losing potential consumers who prefer card payments over alternative ways;
- **Reduce transaction processing times**: when it comes to card payments, swift and efficient transaction processing is critical. By reducing processing times, merchants may deliver a more seamless and easy shopping experience to their consumers, resulting in speedier checkout procedures and more customer satisfaction;
- **Service availability throughout time**: SaltPay should guarantee that the services it provides are always available to merchants, demonstrating its reliability;
- **Full operation capacity with increased usage load**: SaltPay should ensure that its infrastructure can manage rising use and transaction volumes, demonstrating its capacity to meet expanding needs and ensuring that the service stays resilient and capable of servicing an increasing number of merchants;
- **Trustworthy service for the merchants**: trust is a strong conviction, and building trust with merchants is critical for Saltpay instilling confidence in their payment processes, data security, and overall service reliability, encouraging long-term partnerships and loyalty;
- **Consolidated system**: having a unified solution and system benefits not just SaltPay but also the merchants that utilise their services. It is critical to maintain the trust relationship by providing merchants with a single solution that streamlines their payment operations, simplifies their procedures, and lowers the complexity associated with maintaining numerous systems or suppliers;
- **Simple contracts, transparent fees, and fair prices**: one of SaltPay's aims is to provide merchants with simple and transparent terms and conditions, charge structures, and pricing methods. By providing simplicity and transparency, confidence is created, unexpected charges are avoided, and merchants have a clear grasp of the prices connected with the service.

Small and medium-sized merchants are the demographic of customers that SaltPay seeks to target. They share equivalent wants, preferences, and behaviours, especially desiring a POS terminal and payment solution to help them build their business. To reach this customer segment, the channels that represent how SaltPay communicates, distributes, and delivers its value proposition to its customer segments include the website, which explains the service's features, ePOS partners, and demos performed by the Sales team with customers,

demonstrating how the service works. Lastly, SaltPay will create and maintain personal assistance and customer support connections with the customer segments it targets, both with the purpose of assisting and resolving any difficulties that may emerge in the customer's service.

The backstage, on the other hand, relates to the commercial perspective and is made up of key partners, key activities, key resources, and cost structure blocks.

Key partners are essential in supporting and improving SaltPay's operations and value proposition. Collaboration with them assures access to trustworthy and compliant payment hardware, improves the payment process, and, finally, broadens the company's reach and acceptance possibilities.

The key activities are concerned with preserving and updating the software solution provided. Investing in software upkeep and adding new features SaltPay is able to create a dependable, efficient, and adaptable product that matches the changing demands of its consumers. This, in turn, helps to establish client loyalty, promote customer happiness, and keep a competitive advantage in the market.

In terms of key resources, the infrastructure, specifically servers, provides the computing power and storage capacity required to support the software system's performance and reliability, the development teams contribute their expertise and skills to develop and maintain the software, and the incident management process aids in the mitigation of any disruptions or incidents that may occur. Investing in these guarantees that SaltPay can keep its software solution reliable and robust, satisfy customer expectations, and deliver a seamless user experience.

Last but not least, cost structure, representing the main cost that SaltPay is expected to incur when providing its services, is related to the investment in software development and maintenance, which guarantees the usability and continuous improvement of its software solutions, POS terminal hardware, which is required to allow merchants to use the services, marketing, which is critical for creating brand awareness and attracting customers, and cloud server costs, which may be one of the most expensive. In terms of revenue streams, which are the sources of funding, the leasing of terminal machines, service and gateway fees contribute by charging merchants for the use of hardware and transaction processing. Diversifying revenue streams helps to reduce risk and provide a secure and sustainable income strategy.

3.5 Quality Function Deployment

Quality Function Deployment is a concept that originated in Japan in the late 1960s and refers to a technique for creating a product or service based on the needs of the consumer.

Its objective is separated into three core aspects: releasing a higher-quality and lower-cost product to the market faster, achieving customer-driven product design and providing a tracking system for future improvements (Chan and Wu 2002).

The House of Quality, a design tool included with the QFD, and is intended to transform customer needs into product specifications, is represented in figure 3.5. This method was implemented to assure the quality of present and prospective clients.

The demands that the merchant requests, which is the client and interested party in the scope of this dissertation, are to be segmented and evaluated, utilising methods that should also be evaluated.

Table 3.2 displays the requirements ("whats"), that are demanded qualities, and their weights. The value of which was assigned based on the merchant's significance, as well as the complexity and development time connected with that demand. In accordance to the table, the two most significant consumer demands are reduced transaction processing times and support for the growth of the merchant's operations, both of which are directly related to enhancing the overall performance of the product.

Table 3.2: QFD demanded qualities and weights

Demanded Quality	**Weight**
Increased card acceptance rates	8
Reduced transaction processing times	10
Support operations development	10
Cost reduction	8
Ease of use	9

Table 3.3 shows the quality characteristics ("hows") that indicate the technical manner the aforementioned desired qualities are to be attained, as well as their direction of improvement. The goals are to maximise reliability, availability, and abstraction while improving maintainability and usability, which are the target.

Table 3.3: QFD functional requirements and direction of improvement

Quality Characteristics	**Direction of Improvement**
Reliability	Maximise
Availability	Maximise
Maintainability	Target
Usability	Target
Abstraction	Maximise
Cost	Minimise

Based on the findings collected, which are apparent in the House of Quality in figure 3.5, it is feasible to deduce that the majority of the merchant's requirements are significantly related to improved availability and reliability.

Row #	Max Relationship Value in Row	Relative Weight	Weight / Importance	Demanded Quality (a.k.a. "Customer Requirements" or "Whats") / Quality Characteristics (a.k.a. "Functional Requirements" or "Hows")	1 Reliability	2 Availability	3 Maintainability	4 Usability	5 Abstraction	6 Cost
				Column #	1	2	3	4	5	6
				Direction of Improvement: Minimize (▼), Maximize (▲), or Target (x)	▲	▲	x	x	▲	
1	9	17,8	8,0	Increased card acceptance rates	Θ	Θ	O		Θ	O
2	9	22,2	10,0	Reduced transaction processing times	Θ	Θ			O	
3	9	22,2	10,0	Support operations development		Θ		Θ	O	O
4	3	17,8	8,0	Cost reduction	O	O	O			▲
5	9	20,0	9,0	Ease of use		▲		Θ		
6										
7										
8										
9										
10										
				Target or Limit Value	100%	Long term	Lifetime	Achieved	100%	
				Difficulty (0=Easy to Accomplish, 10=Extremely Difficult)	8	10	10	5	8	
				Max Relationship Value in Column	9	9	3	9	9	3
				Weight / Importance	413,3	633,3	106,7	380,0	293,3	137,8
				Relative Weight	21,0	32,2	5,4	19,3	14,9	7,0

Figure 3.5: Quality Function Deployment - House of Quality

Chapter 4

Evaluation and Experimentation

Considering the objective of increasing the number of transactions handled daily on SaltPay's platform while lowering the average time it takes to execute a transaction, it is critical to create and execute several load tests plans.

Load testing is an approach for determining how a large-scale software system behaves under stress, in order to identify load-related issues (Z. Jiang and Hassan 2015). According to Beizer (1984), *load* may be defined as the rate at which various requests are delivered to the System Under Test (SUT).

Depending on the study's intention, the load to which the system will be subjected may differ from predicted values for the system to receive while it is fully operational, or loads greater than anticipated, in order to identify potential functional or non-functional issues. Since the goal of this dissertation is to examine present constraints and provide suggestions to enhance them when larger usage loads are predicted, the system will be subjected to higher-than-expected loads.

This fourth chapter will provide a comprehensive review of the existing solution, based on the findings of the load tests developed to perform this evaluation and make conclusions based on their results.

4.1 Research Hypotheses

Throughout this dissertation, it has been stated that its key objective is to create load test plans for the various system components, as well as analyse the outcomes of those tests and provide potential alternatives for system improvement and scalability.

The system, which is composed of multiple components, will need to be thoroughly examined, with a particular emphasis on the Payments Gateway (which includes the "core" of the whole process), the Acquiring Host, and the consumption of transaction events that enable the near synchronisation of all system components.

Lavrakas (2008) states that a research hypothesis is a specific, clear, and testable proposition or predictive statement about the possible outcome of a scientific research study .

In this manner, the following research hypotheses have been established in order to evaluate whether or not that objective was met:

- **Null Hypothesis (H_0):** The conclusions of this document have no effect on the organisation.

  - The theoretical information presented throughout this dissertation is not relevant to the organisation;
  - The development team does not consider the issues mentioned on the services' status to be a problem;
  - The tests used to analyse the present condition of the Payments Gateway are ineffective, demonstrating that the services cannot manage a high volume of requests;
  - The suggestions for improving service performance are ineffective and cannot guarantee a reduction in processing times;
  - It is not possible to improve the user experience with payment terminals.

- **Alternative Hypothesis (H_1):** The conclusions of this thesis have a positive effect on the organisation.
  - The theoretical information related to best practices and new guidelines for implementing and improving services is relevant to the organisation;
  - The development team does considers the issues mentioned on the services' status to be a problem, and looks into way to improve them;
  - The tests used to analyse the present condition of the Payments Gateway are effective in demonstrating the service limitations and capacity to handle large volumes of requests;
  - The suggestions for improving service performance are effective, and guarantee the reduction in processing times;
  - It is feasible to improve the user experience with payment terminals by lowering processing duration P50 and P90 (percentiles) to 2500ms and 2900ms, respectively.

4.2 Evaluation Indicators and Information Sources

The assessment of the Payments Gateway services that are fundamental to this master's thesis - that is, the Processing-API, Merchant-API, Transactions-API, and Integrations-API - was based on a set of fundamental indicators, that facilitate the evaluation of the hypotheses stated in the preceding section.

- **Reliability:** If the system can handle an increase in the number of requests for transactions while maintaining the same processing capabilities as before;
- **Relevance:** If the development team considers that the findings of this dissertation are relevant to both the Payments Gateway team and the organisation as a whole;
- **Usability:** If the proposals presented for improving system performance and allow for scalability can be carried out by the development team in the present infrastructure;
- **Real-Life Application:** If the suggested changes mitigate the system's constraints and allow for a better user experience with payment terminals by reducing average processing time.

The following information sources will be required to accurately measure the aforementioned indicators:

- **Load Tests:** Load tests will be done on the services to thoroughly assess the present constraints of the Payments Gateway and the Acquiring Host.

  The outputs produced by these load tests will be used to assess the current state of the APIs. The number of requests per second, the number of failed and successful requests, and the average request duration are some of the outputs that the load testing tool provides.

- **Observability Metrics:** Grafana is an open-source framework for managing logs, metrics and traces, providing observability over services (Grafana Labs 2023a).

  The services implement metrics and traces that allow assessing the execution times of certain methods while processing a transaction. These measurements, as well as the logs, will be utilised in conjunction with the results of the tests.

4.3 Evaluation Methodology

Evaluation methodologies enable the combination of descriptive data and empirical evidence with appropriate values in order to make explicit evaluative judgements (Davidson 2004).

The hypotheses stated above require an evaluation methodology approach that is dependent on the indicators and information sources indicated above in order to be effectively assessed.

4.3.1 Load Tests

Load testing allows for assessing how a web site, or other software service, supports the workload it expects, by running a set of script that emulate customer behaviour at different load levels (Menasce 2002). Forecasting of substantial increases in user traffic, such as the projected growth in the number of transactions handled daily, is a scenario in which load testing may be highly beneficial in verifying the services' response times, availability, and reliability.

Different test scenarios have been created so that the load testing process could cover all of the different ways to pay supported by a POS payment terminal.

- **Card-Present:** Card-present payments, as the name indicates, are those in which the cardholder is present and uses a physical debit or credit card as payment.

  The magnetic stripe, chip or contactless capabilities of the card are used in this type of payments, and the pin can be required.

- **Pay-By-Link:** Pay-By-Link is a method of initiating a transaction without the presence of the cardholder and the card.

  It enables the merchant to request payment by Short Message Service (SMS) or email. The client is sent to the payment page after obtaining the payment link, where they may enter their credit card information and confirm the payment.

  Given the nature of this transaction flow, a pay-by-link transaction can ultimately be viewed as an e-commerce transaction in the scope of this dissertation, due to their similarities.

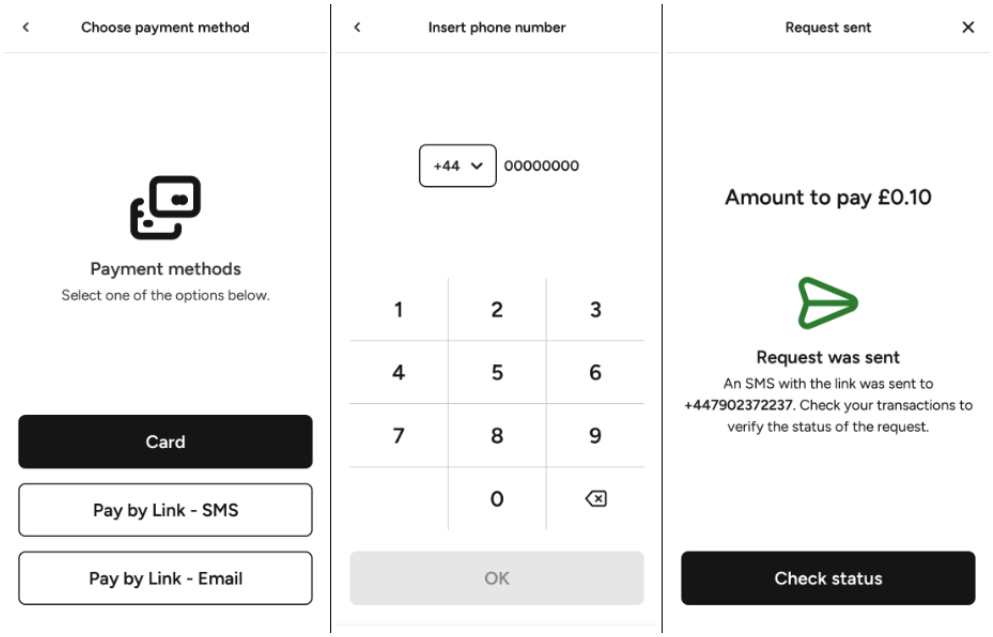

Figure 4.1: Terminal Pay-By-Link View With Phone Request (from: Teya Help Center (2023))

As described in the chapter 2, section *Comparison between technologies*, K6 was the tool that was chosen to conduct load testing.

4.3.2 Testing Scenarios

Considering the fact that there are two separate situations to examine, that were mentioned in the preceding subsection, two independent test scripts have been created: one to carry out load tests on card-present transactions and another for performing load tests on e-commerce transactions.

All queries routed through a terminal enter the Payments Gateway's services stack via Merchant-API, which then proxies the request to Processing-API, which acts as the core where the transactions are really performed. Elements such as the Charge and Instrument, referred to in the chapter 2, are created here in response to API queries.

Both test scripts use a set of common environment variables that are relevant for the requests to be performed to the API, such as the base URL for Merchant-API, merchant ID and merchant key, which are used to authenticate the requests and the terminal ID, which is mapped to an ID on the Acquiring Host side.

Different sets of test options, which specify the behaviour of the test, have been used to replicate different circumstances and metrics, such as the number of requests per second. The sections that evaluate the test results will offer a more detailed explanation of the alternatives employed.

It is important to mention that both tests scripts were designed to simulate transactions with Way4, one of the Acquiring Hosts. Considering the fact that Solar's development environment can be somewhat unreliable, it was decided to execute the tests in Way4, which is more stable and delivers accurate outcomes.

Card-Present Script

Requests are made to a particular endpoint created for card-present transactions, which acts as an abstraction layer between calls to the charges and instruments endpoints (elements that were previously covered in chapter 2). In this script, the card entry mode *magstripe* was used to simulate the magnetic stripe swipe of an actual card.

In addition to the endpoint authentication, these transactions require a signature, which is computed using the terminal's Hash-based Message Authentication Code (HMAC) secret and the current timestamp. The timestamp and signature are both appended to the request's headers.

```
/**
 * Creates a valid card-present transaction with Way4, using Mastercard.
 * @param {String} card_present_url - The url to create card-present transactions
 * @param {Object} headers - The headers used in the request
 * @param {String} hmac_secret - The HMAC Secret of the terminal used in the transaction
 */
function createCardPresentTransaction(card_present_url, headers, hmac_secret) {

    // Define the card-present payload
    let payload = '{"amount":{"value":"10.0","currency":"EUR","gratuity":"2.0"},"card":{"entry_mode":"magstripe",'

    // Get current timestamp and generate the request signature
    let timestamp = new Date().toISOString();
    let signature = crypto.hmac('sha256', hmac_secret,  payload+timestamp, 'hex');

    // Add signature and timestamp to request headers
    headers.headers['X-SWITCH-SIGNATURE'] = signature;
    headers.headers['X-SWITCH-SIGNATURE-TIMESTAMP'] = timestamp;

    // Make the request
    let card_present_req = http.post(card_present_url, payload, headers);

    // Validate that the card-present transaction was successfully created
    check(card_present_req, {
        'card-present created': (r) => r.status === 201,
    });
}
```

Figure 4.2: Helper method used to create card-present transactions, in the load tests script

Figure 4.2 depicts the method created for generating a card-present transaction using the Acquiring Host Way4 and a Mastercard test card. In summary, the content for the requests is set first, then the signature required for authentication is computed and applied to the request headers, then a POST request is made to the card-present endpoint. Finally, the status code given by the response is validated to ensure that the card-present transaction was correctly created.

E-Commerce Script

In the case of e-commerce, no special endpoints have been created to facilitate this sort of transaction. As a result, a request to both the Charges and the Instruments endpoints is required. Aside from the merchant ID and merchant key, which are shared by both scenarios, no extra authentication is required in this circumstance.

For this scenario, two auxiliary methods were developed: one for creating the Charge and another for creating the Instrument, which are components of a single transaction. Figure 4.3 illustrates the method that was used for creating the Instrument using a Mastercard test card. Similarly to the last test case, the request payload is defined first, followed by a POST request to the instruments endpoint. At the end, the status code returned in the response is validated, to guarantee that the Instrument was correctly created.

```
/**
 * Creates a valid e-commerce instrument with Way4, using Mastercard.
 * @param {String} charge_id - The ID of the charge created
 * @param {String} instruments_url - The url to create instruments
 * @param {Object} headers - The headers used in the request
 */
function createInstrument(charge_id, instruments_url, headers) {

    // Define the instrument payload
    let instrument_payload = JSON.stringify({
        charge: charge_id,
        number: '5188[redacted]',
        expiration_year: '2025',
        expiration_month: '12',
        cvc: '123',
        customer_context: 'ecommerce',
    });

    // Make the request
    let instrument_req = http.post(instruments_url, instrument_payload, headers);

    // Validate that the instrument was successfully created
    check(instrument_req, {
        'instrument created': (r) => r.status === 201,
    });
}
```

Figure 4.3: Helper method used to create instruments, in the load tests script

However, given that it is essential in this situation to make two separate API queries to different endpoints, the sequence of the requests is important. The Charge is created first, and the ID of the resulting object is injected into the code that only then will be creating the Instrument. A valid Charge is one of the properties needed to create an Instrument.

4.3.3 Metric Dimensions

Different observability metrics have been utilised to analyse the current state of the services that collectively constitute the Payments Gateway and Acquiring Host, which are the centre of this study, and inside the critical route of interaction by the physical terminals.

In addition to offering charts, graphs, and alerts, Grafana, as described in the preceding paragraphs, enables management of many data sources. Loki and Prometheus should be emphasised among the different data sources that SaltPay's Grafana instance uses. Loki is a multi-tenant, highly available, and horizontally scalable log aggregation system (Grafana Labs 2023a), and Prometheus is an open-source event monitoring and alerting system (Prometheus 2023).

Along with standard log and metric queries, Grafana also enables the construction of dashboards, which have an additional significance and enable a more thorough analysis of the item being studied.

Figure 4.4: Example of the dashboard created to measure the execution times of the different methods in a transaction's critical path

As seen in figure 4.4, percentile values such as the P50 (average), P75, P90, P95, and P99 are excellent examples of parameters that aid in understanding how the numbers are distributed in a sample, in this case, how the execution time of the methods is distributed. These values are calculated based on the execution time logs that have been included in the scope of the source code of the methods in analysis.

Furthermore, the output report generated by K6 offers a summary of the load test results. The end-of-test summary provides aggregated statistical information such as median and average values, minimum and maximum values, and percentiles, in addition to allowing for the definition of custom metrics and setting the statistics that are shown.

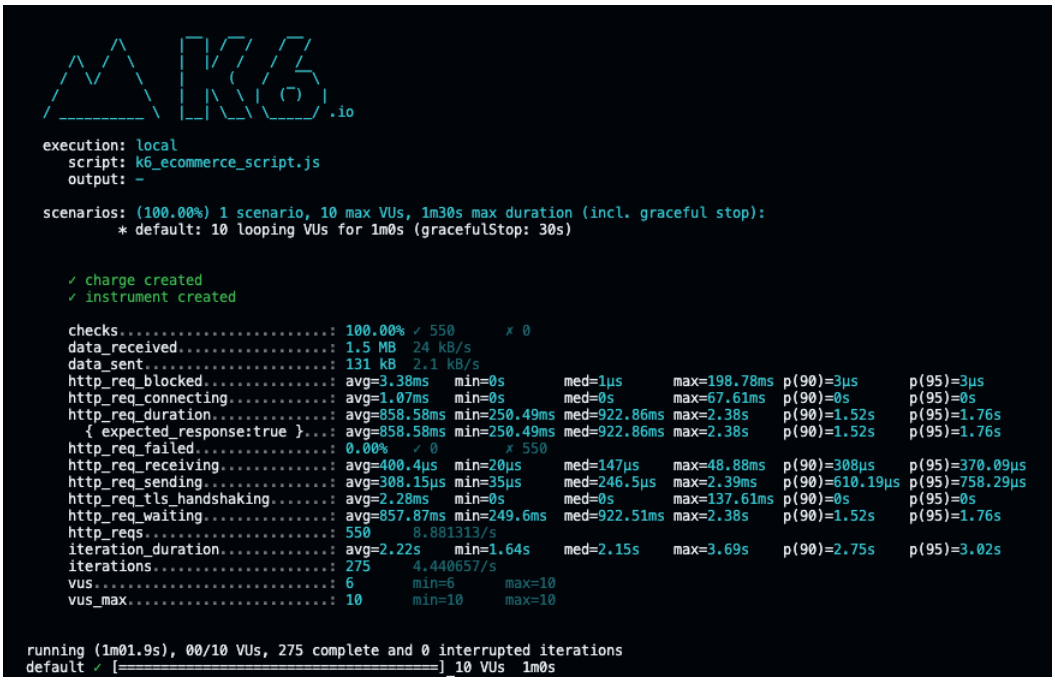

Figure 4.5: Example of an output report generated by K6

Figure 4.5 depicts the output report generated for a sample run of the load tests designed within the scope of this dissertation. The two metrics that will receive the greatest emphasis in the output summary will be the HTTP request duration and the number of HTTP requests.

4.3.4 Metric Criteria

It must be emphasised that the processing times seen in the production environment — the one with which the terminals in use by the merchant interact with — are significantly lower than those seen in the development environment, where the load tests will be run and the analysis will be done.

The SaltPay cloud infrastructure, which is allocated in Amazon Web Services (AWS), contains the Payments Gateway services. In terms of system resources, such as CPU power and Random Access Memory (RAM) memory, as well as instance storage and network bandwidth, the infrastructure configurations of the Payments Gateway development environment are inferior to those in the production environment. While in the production environment there are 12 replicas with 4 CPUs, there are only 2 replicas with 1 CPU in the development environment.

Apart from the restrictions that apply to the Payments Gateway environment, the Acquiring Host environment also has limitations that are related to the use of card-scheme simulators (i.e., the Acquiring Host will communicate with a simulator of the environments for Visa and Mastercard rather than directly with those environments).

The Payments Gateway services were migrated from their own "external" cloud infrastructure to SaltPay's cloud infrastructure while this dissertation was being prepared. Following the change, several challenges arose, mostly with databases and networking differences, resulting in a minor increase in transaction processing time, namely in P50 and P90.

Taking the above criteria into account, the transaction processing time percentiles P50 and P90 in 2023 were roughly 1.5s and 3s, respectively. In terms of gateway performance, the current highest number of transactions per second is 8.

Given that the goal of this dissertation is to assess the current limitations and identify others of the Payments Gateway services and Acquiring Host, as well as provide suggestions on how to improve those limitations, it is necessary to achieve values of at least 8 transactions per second and processing times of around 1.5 seconds on average while performing the tests. By doing so, even if the tests are run in settings that differ from those in the production environment, valid findings will be obtained, allowing for higher-fidelity suggestions and improvements.

4.4 Results

To collect as much data as possible and hence have more trustworthy measurements and findings, it was scheduled to execute load tests every 15 days in collaboration with the Payments Gateway development team. Even though the load tests would be performed in the development environment, so as not to embarrass any team that used it (for example, the Payments Gateway team or the Terminals team), and so as not to interfere with the Acquiring Host simulators, they would be performed during offline hours, when no one would be using it.

K6 test execution configuration parameters were altered several times to create distinct load peaks. This allowed for distinct conclusions for the numerous scenarios under consideration.

The results of the load tests were gathered and analysed over a period of weeks, allowing for a collection of output values that may be taken as a reference, given the continuous and repeated execution of these tests.

Table 4.1: Results obtained in load tests with 20 VUs and 200 iterations

	Issuer	Requests Per Second	HTTP Request Duration				
			Avg. (P50)	Min.	Max.	P90	P95
Card-Present Transaction	**Visa**	7.41	2.07s	1.13s	3.41s	2.74s	3.02s
	Visa*	7.2	2.14s	958.11ms	3.78s	3.03s	3.22s
	Mastercard	1.29	14.24s	2.28s	16.52s	14.96s	15.01s
	Mastercard*	7.44	2.07s	1s	3.97s	2.85s	3.09s
Card-Not-Present Transaction	**Visa**	10.6	1.52s	266.09ms	4.91s	2.97s	3.43s
	Visa*	10.62	1.52s	251.23ms	4.74s	2.8s	3.12s
	Mastercard	5.5	4.2s	256.8ms	8.02s	6.32s	6.51s
	Mastercard*	10.3	1.56s	256ms	6.5s	3.5s	4.07s

Table 4.1 presents the results obtained in the initial iteration of the load tests, which consisted of 20 VUs executing 200 concurrent iterations - that is, there will be 20 instances simulating a user performing the requests made in the script, which will create 200 transactions in total. The asterisks(*) next to the issuer name in the table correspond to test conditions where the simulator was switched off, allowing transactions to run much faster than if communication with the simulator was required.

The findings of this initial iteration show that, overall, Mastercard transactions had worse outcomes than Visa transactions, specially when it comes to card-present transactions. This is an identified limitation of the simulator, which only allows for one card-present transaction per second, as opposed to over 30 in Visa. The results achieved with Visa cards in both card-present and card-not-present transactions with the simulator switched on or off are almost identical, therefore it is irrelevant whether the tests are performed with the simulator turned on or off. The same cannot be said with Mastercard, with the discrepancies being rather noticeable given the aforementioned constraint.

Card-present tests perform worse than card-not-present tests, as demonstrated by the data in the table. This is due to the fact that, in addition to a higher number of parameters (many of which are linked to terminal capabilities), a transaction conducted in person often entails additional parameter enrichment processes. Among these, the queries made to the HSM to decrypt data related to the card's track2[1] and the retrieval of data linked to the Personal Identification Number (PIN) should be highlighted, because they are among the requests that take the most time to execute, aside from actually delivering the message to the Acquiring Host over the sockets.

Given the very short duration of the tests (40 seconds to 2 minutes, depending on the scenario), this initial run should not be regarded as a load test. Instead, the initial set of data should be taken as a guideline, demonstrating how the development environment generally behaves.

[1] Track2 refers to a specific data format used in the magnetic stripe of a payment card, and contains encoded information about the cardholder's account.

When it comes to building test scenarios, personalising test executors, and establishing custom thresholds, K6 provides a plethora of customisation choices. An executor controls how the tool will schedule VUs and iterations, and a threshold can be seen as the pass or fail criteria that is defined for the test metrics (Grafana Labs 2023a).

One of the executors available is the *constant-arrival-rate* executor, which is particularly beneficial for the purposes of this research. It allows for the definition of a predefined number of iterations over a certain time period, as well as the enforcement of criteria such as the number of requests per second.

With the goal of offering an improved understanding of the current state of the APIs, conducting a test with a *constant-arrival-rate* executor performing 10 iterations per second (that is, 10 requests per second) for a period of 10 minutes proves to be quite interesting in this context. Although the 10 minutes fixed duration of the test is still a relatively quick test, when combining its execution time with the amount of fixed requests per second, it becomes possible to emulate peak usage of the services in question and validate whether or not the services continue to operate as they would normally do with regular load.

Table 4.2: Results obtained in load tests with a constant-arrival-rate executor of 10 requests per second

	Issuer	Failed Requests	Requests Per Second	HTTP Request Duration				
				Avg. (P50)	Min.	Max.	P90	P95
Card-Present Transaction	**Visa**	212	6.80	31.61s	251.30ms	59.39s	52.37s	55.37s
	Visa*	78	6.94	29.59s	266.54ms	60s	45.38s	48.98s
	Mastercard	1999	4.19	54.88s	267.94ms	60s	60s	60s
	Mastercard*	347	6.97	30.18s	272.02ms	60s	57.08s	60s
Card-Not-Present Transaction	**Visa**	83	10.09	22.45s	167.79ms	59.90s	43.87s	47.89s
	Visa*	567	10.28	21.67s	192.06ms	60s	59.99s	60s
	Mastercard	1176	7.41	32.48s	173.32ms	60s	55.90s	57.50s
	Mastercard*	279	10.13	22.06s	164.64ms	60s	49.85s	57.7s

Despite the fact that an executor was used that forced the number of requests per second to be 10, the data shown in table 4.2 show that this figure was not fulfilled in the majority of the cases. Apart from being unable to withstand the enforced amount of requests per second, there were several failed requests, particularly in Mastercard testing, owing to simulator limits.

Similar to the previous test run, card-present tests performed worse than card-not-present tests, and Mastercard performed substantially worse than Visa in terms of the number of unsuccessful requests, requests per second, and average request time. The request durations maximum, P90, and P95 for Mastercard are extremely close to 60 seconds, which is the timeout imposed on the APIs for the received requests. Considering this, it is reasonable to deduce that a large portion of the queries made to the Mastercard simulator timed out, causing the request to the Payments Gateway API to time out and the transaction to fail to be created. Even though Visa behaved better than Mastercard, overall, there was an increase in processing times and a slight decrease in the number of requests per second that were performed.

The examination of the metrics accessible through Grafana, in addition to the dashboards created by the Payments Gateway development team to support in monitoring how the machines are operating, it is possible to conclude that during the process of completing this load test, the machines handling the incoming requests - in particular, all instances of Processing-API, the "core" responsible for the creation of the transaction elements - were under a significant amount of load.

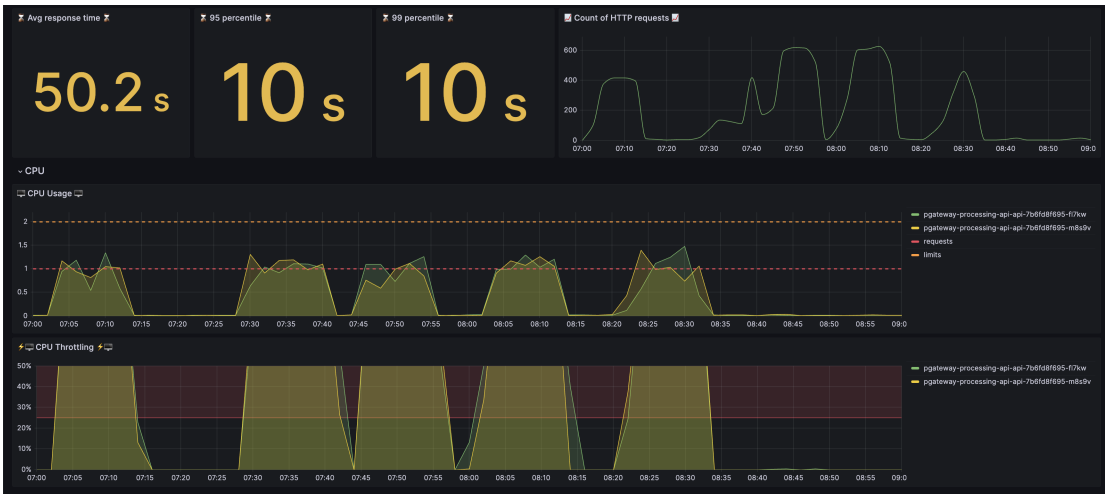

Figure 4.6: Processing-API response times and CPU metrics in a time span during which tests were run

Figure 4.6 features one of the time segments during which the tests were carried out. Substantial spikes are seen in the areas pertaining to "Count of HTTP requests", "CPU Usage", and "CPU Throttling", corresponding to times when the test script was executing.

CPU throttling refers to *Dynamic Frequency Scaling*, a technique that modifies CPU power usage by altering the processor's operating frequency, lengthening the time it takes to process huge volumes of data. It is a fundamental control mechanism in processors that safeguards them from damage (Bao et al. 2016). As the number of transactions requested rose, so did processor use, and CPU throttling reached 100% during peak periods.

Further examination of the traces and other dashboards that provide information about the amount of time it takes for each step in a transaction lifecycle reveals that, while there was a slight increase in the average time it took for requests to reach the Acquiring Host and the response to be returned, as well in the enrichment of the transaction parameters, these findings are insufficient to explain the increase in total processing time. This increase in the execution time of the methods that comprise the transaction lifecycle can be seen in figure 4.7.

Figure 4.7: Execution time in some of the steps that are included in a transaction lifecycle

The metrics that have been reviewed earlier, as well as the information obtained from the reports generated by K6, indicate that, along with to the performance of the methods that enrich the parameters of an instrument not being the most successful, the Processing-API constitutes the greatest barrier to supporting higher transaction numbers.

4.5 Assessment Completion

It is reasonable to infer that the load tests performed were effective in reaching the system limitations based on the findings obtained and examined from the load testing using K6, as well as the metrics and dashboards provided through Grafana. The study of the dashboards, metrics, and traces reveals where latency is introduced into the process.

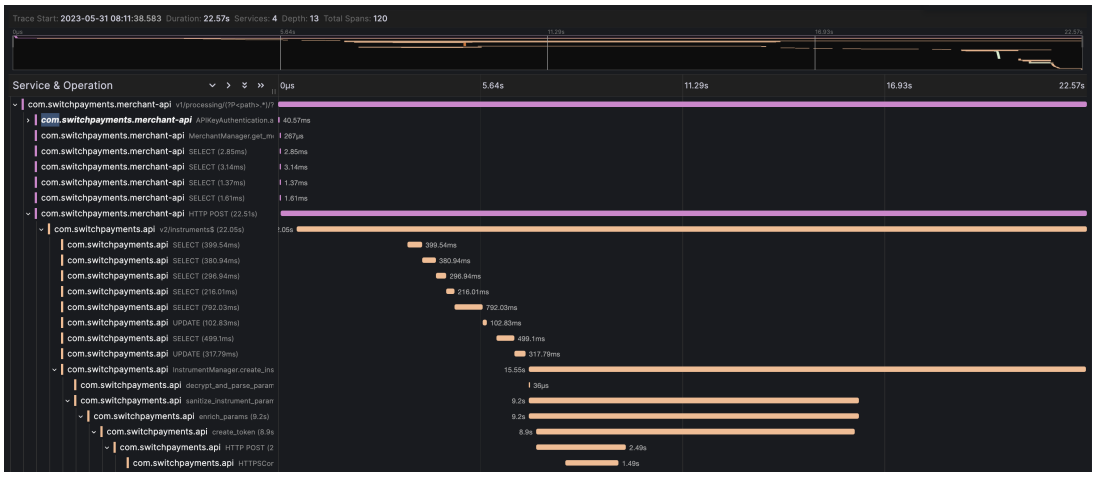

Figure 4.8: Trace of a transaction executed in one of the load tests

The variations between production and development environments, the usage of simulators, and the fact that the tests are conducted from a personal computer rather than the cloud, may all have an influence on the findings achieved. Nonetheless, there is enough information to show that several crucial paths still require modification, and that improving them would possibly result in faster transaction processing times. That said, if it is deemed acceptable

to implement a modification, the trade-offs should not be overlooked, because concurrency is critical in a transaction, and an improvement might become a deteriorating action.

Chapter 5

Optimisation Strategies and Implementation Suggestion

Technology has been evolving at a blistering pace over the last few years, whether it's the emergence of previously unknown domains like block-chain or the advancement of established technology like Artificial Intelligence (AI). This growth has also resulted in substantial breakthroughs in the payments sphere, revolutionising the way transactions are handled.

Whatever technological branch is being discussed, there is a shared increasing desire across all to outperform rivals - enhancing system performance while minimising production costs. As a result, engineers have been pushed to investigate ways of product optimisation in order to meet this demand.

According to Rao (2009), optimisation is the process of determining the conditions that result in the greatest or minimal value of a function. There is no one answer to all optimisation challenges; instead, a variety of strategies have been devised to address the issues that may occur depending on the subject.

This chapter will explore optimisation techniques for dealing with the challenges mentioned in the previous chapter, based on the design of the services and the infrastructure settings. Suggestions for implementation based on the techniques that were investigated, and the potential outcomes of their implementation, will also be explored.

5.1 Optimisation Strategies

As noted in the introduction to this chapter, there are numerous approaches to exploring optimisation in a single service or field. Several premises must be presented due to the variety of services and containers present in the Payments Gateway stack, in order to define the appropriate optimisation solutions for them

5.1.1 Services Heterogeneity

The Payments Gateway is formed up of multiple distinct micro-services that interact with one another to allow transaction processing to take place and aggregate results for providing internal statistics and analytics with the data resulting from the transactions.

The vast majority of these services have been created in Django, a Python framework noted for its scalability, speed, and security, as well as Flask, a Python micro web framework (Django Software Foundation 2023a; Pallets 2023). In addition to the micro-services, the databases are also a component of the stack, with PostgreSQL dealing with the core of the

services (such as Processing-API and Merchant-API) and MongoDB handling the data in Transactions-API, Lifecycle-API, and Analytics-API (responsible for aggregating data pertaining to a transaction's lifecycle events and the analytics of those). Lastly, RabbitMQ, Redis, and Kafka are utilised by the services for communicating with one another or to publish messages that may be read by other SaltPay services, and OpenSearch is used to conduct full-text searching over MongoDB data.

As it is possible to see from the preceding paragraph, Payments Gateway services include a wide range of products. All of those services are deployed on the SaltPay cloud infrastructure, and while they're all containerised using K8s, the optimisation methods that should be utilised vary.

When it comes to optimisation strategies, there are various topics that ought to have covered, ranging from fine tweaking K8s configurations to database queries and the source code itself.

5.1.2 Transaction Parameters Enrichment

Treating the parameters is one of the initial steps in the creation of an Instrument. It consists of processing the request parameters used to create an Instrument object so that the output can be persisted in logs or databases - therefore removing sensitive data that cannot be retained (for example, the card number and Card Verification Value (CVV)) and adding new information (for example, the card's last four digits and a token ID). This procedure is divided into two parts: sanitation and parameter enrichment. Various API calls must be conducted to enrich the parameters and include details pertaining to the card Bank Identification Number (BIN), the terminal used in the transaction, and the tokenisation of the cardholder data.

In the Processing-API source code, where the Instruments originate and thus the parameters are handled, there are three data enrichers: the Card BIN data enricher, the Terminal data enricher, and the Tokenisation data enricher. Only if the *is_enabled* property evaluates to *True* are the data enrichers called and run. This call was made sequentially, that is, one enricher at a time, assessing the property and executing it if applicable.

```
# Enrich data
enriched_data = {}
for enrichment_provider in enrichment_providers:
    enricher = enrichment_provider()

    if enricher.is_enabled():
        data_to_add = enricher.enrich_params(
            params=params or {},
            object_id=object_id,
            object_type=object_type,
            charge_request_log=charge_request_log,
        )
        enriched_data.update(data_to_add or {})
```

Figure 5.1: "Old" call to data enrichers, sequentially

Although the calls to data enrichers are routed to services inside the same cloud network as the Processing-API, and while they are very compact requests (with small request bodies and

payloads), they add some latency to the transaction processing process. Given that those external requests were executed sequentially, which can be seen in figure 5.1, the transaction flow could only continue after all of them were completed. If there was an issue with any of them - for example, a timeout due to a system malfunction - all of the overhead would be added up to the total time it took to process the transaction.

To prevent the possible complications that these queries may cause, as well as to reduce the latency injected into the operation, a simple option is to conduct these requests asynchronously, using threads. Threads are often defined as mini-processes, or processes within a process, and are the smallest sequence of instructions that can be managed independently (Tanenbaum and Bos 2014). They execute their own piece of code independently from others, and multi-threading often leads to performance gain (Tanenbaum and Steen 2006), making it an attractive solution for the issue presented above.

```
# Enrich data
enriched_data = {}
execution_start = time.time()
enabled_enrichers = []
for enrichment_provider in enrichment_providers:
    enricher = enrichment_provider()

    if enricher.is_enabled():
        enabled_enrichers.append(enricher)

try:
    with concurrent.futures.ThreadPoolExecutor() as executor:
        futures = []
        for enricher in enabled_enrichers:
            futures.append(
                executor.submit(
                    trace_wrap(enricher.enrich_params, capture_active_span=True, create_new_span=False),
                    params=params or {},
                    object_id=object_id,
                    object_type=object_type,
                    charge_request_log=charge_request_log,
                )
            )
        for future in concurrent.futures.as_completed(futures):
            enriched_data.update(future.result() or {})
except Exception:
    logger.exception('Unexpected error while running enricher')
    raise
```

Figure 5.2: Updated call to data enrichers, concurrently

Thread Pools are a restricted set of threads that may be assigned to execute concurrent tasks. Maintaining a pool of threads helps improve performance since there is no overhead with thread creation and joining, which can be considerable (Garg and Sharapov 2001).

The call to the data enrichers was changed to use a thread pool, as shown in figure 5.2, and therefore this enrichment process would run simultaneously. Following the implementation of this optimisation strategy, processing times fell slightly, and therefore the overall transaction process performance improved.

5.1.3 Transaction Flow Simplification

Aside from the sanitation of the transaction's input parameters, risk analysis and dynamic routing are the processes that follow, possibly being within the phases that take the longest, disregarding the payment provider call.

Risk analysis is a security step that is part of the transaction lifecycle. Its objective is to ensure that a merchant's transactions are secure by outlining how they are processed and settled. This is accomplished by defining risk rules, that can be set based on specific amounts, locations, currency, or other relevant parameters, which will determine whether a transaction will be accepted, blocked, set for review, or require 3DS authentication (SwitchPayments 2023). These rules are defined and stored in Risk-API, which is in charge of assessing transaction metadata against the stated rules and returning the determined action based on this.

```
# 2.1) Get Risk Result from Risk API (only if it is enabled for this merchant and it is not external sources)
risk_result = None
if (
    settings.RISK_API_ENABLED is True
    and external_operation is None
    and charge.merchant.get_settings().get(MerchantMetadataKeys.RISK_ANALYSIS_ENABLED) is True
):
    execution_start_risk = time.time()
    with tracer.start_as_current_span('get_instrument_risk_result'):
        risk_result = RiskAPI().get_instrument_risk_result(
            merchant=charge.merchant,
            instrument_id=instrument_id,
            charge=charge,
            sanitized_params=sanitized_params,
            request_log=request_log,
            instrument_fingerprint=instrument_fingerprint,
        )
    execution_time = time.time() - execution_start_risk
    observe_processing_method_latency(execution_time=execution_time, method='get_instrument_risk_result')
    performance_logger.info(
        'Execution Time',
        metadata={'method': 'get_instrument_risk_result', 'execution_time__number': execution_time},
    )
```

Figure 5.3: Risk-API call in Instrument creation flow

Although the call to Risk-API happens only if the merchant conducting the transaction chooses to enable risk analysis and Risk-API is configured as enabled in Processing-API settings, as shown in figure 5.3, the aforementioned call still reflects a POST request to another "external" service. With these calls, more latency is added to the entire transaction duration, that can reach at most 5 seconds, corresponding to the set timeout amount.

This call's evaluation and optimisation is not as simple as the last one. Removing this instruction entirely is unlikely to be viable, as this additional safety validation layer would be lost, and merchants would lose more granular control over their transactions. Using asynchrony, and a thread solely to this single request, could hardly be applicable, because the risk analysis result is required to determine whether or not the instrument will be created and the request to the payment provider will be made.

Apart from removing the risk analysis, which would simplify the transaction flow and hence reduce processing times, another viable optimisation technique would be to move the risk rules and analysis into Processing-API, eliminating any network latency associated with the request to Risk-API. The latter method, on the other hand, increases the complexity of Processing-API and broadens its scope beyond what it was designed to achieve, which is create and process transactions - the separation exists to allow for greater readability and maintainability of both services.

It is important to note that risk analysis was disabled by default in all SaltPay merchants, which in turn lead to a slight improvement in processing times.

5.1.4 Source Code Optimisation

All Payments Gateway services are written in Python, with the great majority utilising the Django framework, as mentioned in the *Services Heterogeneity* section. There are various optimisation strategies that may be used to improve speed in Python and Django, that should be investigated in the structure of this dissertation.

Django, and Python code in general, may be optimised using a set of best practises inherent in every programming language, as well as recommendations included in the official documentation. Considering the amount of the code that has been created for all Payments Gateway services, the optimisation strategies mentioned in this section are not going to be targeted at particular sections of code, but rather generic recommendations that should be investigated and perhaps added to the backlog of tasks as technical debt.

Among some of the best coding practises is eliminating duplicate code and implementing the DRY principle (Do Not Repeat Yourself), utilising the correct data structures (such as sets for fast searching), going async whenever it is possible, and using built-in functions. These practises apply to all programming languages, including Python, as pointed out by Python Software Foundation (2023).

In respect to Django-specific optimisation tips, in addition to what was previously mentioned, Django Software Foundation (2023b) recommends reducing and optimising the number of queries used (as querying the database is one of the most expensive requests to process), caching predictable data, and improving database queries. Regarding database access optimisation, the *explain* method, which returns the execution plan of a QuerySet, may be used to analyse the performance of certain queries.

```
>>> print(Blog.objects.filter(title="My Blog").explain(verbose=True, analyze=True))
Seq Scan on public.blog  (cost=0.00..35.50 rows=10 width=12) (actual time=0.004..0.004 rows=10
loops=1)
  Output: id, title
  Filter: (blog.title = 'My Blog'::bpchar)
Planning time: 0.064 ms
Execution time: 0.058 ms
```

Figure 5.4: QuerySet.explain() example (from: Django Software Foundation (2023a))

To potentially put any of these recommendations into action, it would be essential to completely review all of the source code, determining what sections may be improved and then implementing those enhancements. Such could end up in a highly complete, though complex, optimisation process that, in the end, could end up saving just a handful of milliseconds and turn out to be an unnecessary usage of resources.

Known Limitation - Socket Proxies

It is fundamental to maintain a constant connection with the Acquiring Hosts, Way4 and Solar in order to communicate with them. It is not desirable to have any API do this since it can slow transaction requests. Instead, each of them has its own proxy server that maintains and manages this constant connection. To connect with the providers, all proxy servers that have been created use sockets (TCP or SSL).

Currently, the proxies only do two things:

- Serve as a proxy:
  - Receive message requests from Processing-API and proxy them to the payment provider;
  - Receive message responses from the payment provider and proxy them back to Processing-API.
- Manage connections:
  - Open and close connections (send sign-on/sign-off messages required by the provider, to open/close the connection);
  - Keep the connections open;
  - Reconnect the connections when there's read/write timeouts.

The proxy socket servers have been developed from scratch up so they can communicate with the Acquiring Hosts and hence enable for transaction processing.

There have been a number of issues found with these, such as memory leaks and deadlocks, which have already been addressed and resolved. However, considering the complexity of this piece of software, the necessity to rebuild them in a more simplified, and maintainable way has been discussed multiple times. While this proposal will require a significant amount of time and resources, it remains on the drawing board and may be put into effect in the future.

Identified Limitation - Processing-API

Through the examination of the dashboards with data related to the usage of Processing-API resources during the time frame where the load tests were executed, it was possible to assert that the CPU utilisation of this service had a considerable impact, as previously evidenced in figure 4.6.

With traffic increases, CPU spikes are to be expected - the amount of requests received could hinder service performance, especially if it is not adequately optimised, and in extreme cases, entirely shutdown the service. The development environment has considerably fewer resources than the production environment, and when the simple load test was run with a constant request arrival rate of 10, a rather modest amount, the API stopped responding, causing the transaction requests to fail.

However, the limits of the environment should not be the reason why the load testing is able to achieve just a few transactions per second. According to information provided by Way4, their service is unaffected by the load testing, however Payments Gateway services, particularly Processing-API, which may be the bottleneck of these inadequate rates. As a result, the ability to handle a high number of transactions is solely dependent on the quantity of resources available.

Although increasing resources might fix the low transaction rate problem momentarily, it would not be a perfect optimisation method, especially considering the substantial infrastructure costs involved. As a result, the analysis of critical transaction processing paths should be equated in order to try to develop a solution that allows for concurrent connections. In addition to this, optimising the service's settings could also improve transaction processing times and the number of transactions achieved

Identified Limitation - RabbitMQ Connections

A trace may be viewed as a series of actions that together describe a transaction handled by a service and the services with which it connects. By examining the trace of a transaction produced during a load test, it is possible to determine how long each operation takes and where Processing-API spends the most time.

Processing-API produces audit-log events, which are records that reflect what has occurred within the application - such as data generation or modifications, such as the creation of a new transaction, identifying who did it. These events are broadcast to any systems downstream of the transaction authorisation that need near-complete synchronisation, including Event-Sourcing, Lifecycle-API, and Transactions-API, among others. These events are transmitted to a RabbitMQ queue, which is read by the previously listed services, allowing the transaction information to be shared.

RabbitMQ is one of the most popular open-source message broker software that implements the Advanced Message Queuing Protocol (AMQP) (Ionescu 2015). It is presumed that there is a Publisher who generates the message and sends it to the message queue, an Exchange who receives messages from the Publisher and routes them to the correct Queue, a Message Queue which is a buffer where the message is stored until it is read, and Subscribers who subscribe to the Message Queue and receive messages (VMWare Inc. 2023).

In the case of a transaction lifecycle, an event message is sent to RabbitMQ after the Instrument is formed. Analysing the traces revealed that there is a period in every transaction dedicated to initiating the connection with RabbitMQ, which takes an unreasonable amount of time.

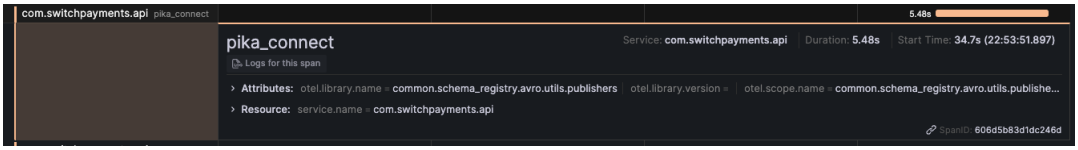

Figure 5.5: Span with the connection to RabbitMQ

Figure 5.5 demonstrates a span of the RabbitMQ connection in a transaction made during one of the load tests, during which 5 seconds are spent just opening the connection with RabbitMQ. Analysing the source code reveals that the connection is closed after publishing a message to the message broker, implying that a new connection must be created before publishing any message. When CPU utilisation hits maximum levels, the time required establishing the connection rises proportionally.

According to VMWare Inc. (2023), connecting to RabbitMQ is a fairly costly action that requires a number of stages - which validates the higher figure. To eliminate the need to start a new connection each time a message is to be published, reusing the connection for several requests can possibly be implemented - potentially reducing the amount of time spent on this step and the time needed to carry out a transaction.

5.1.5 K8s Optimisation

Kubernetes has been widely used by the industry, particularly for container scalability and scheduling. It is a Docker-based open-source container management system that provides a whole management and orchestration ecosystem, which includes the capacity to deploy

containers across many server nodes and schedule and extend these containers throughout the cluster, resource monitoring, log collection, service discovery and container networking (Hu and Wang 2021; Huo et al. 2022).

Following the migration to the SaltPay cloud infrastructure, the Payments Gateway services had to be modified to accommodate K8s, has they previously only had Docker support. One of the tasks necessary to complete the use of K8s in the services was to include the right configuration files, which would allow for the accurate configuration and deployment of resources.

Cloud services are becoming more widely available, which additionally implies that cloud providers must increase their capacity to avoid losing Quality of Service (QoS) or violating Service Level Agreements (SLAs) (Coutinho, Gomes, and Souza 2015). Elasticity, one of the most important aspects of cloud computing, may be described as the system's ability to alter its computing resources on demand as workload changes, by provisioning or unprovisioning resources automatically (Herbst, Kounev, and Reussner 2013). To fully utilise the elasticity of the cloud, a system that enables resources to be automatically provisioned needs to be implemented, as overprovisioning will result in wasted resources and, as a result, extra costs, and underprovisioning will result in performance degradation and SLA negligence, which is defined as auto-scaling, as suggested by Qu, Calheiros, and Buyya (2018).

Kubernetes provides auto-scaling tools at two different levels, application/container level, and infrastructure level. The Custer Autoscaler is a highly adjustable infrastructure solution that automatically adapts the size of the cluster by dynamically modifying the number and size of identical nodes. At the container level, HorizontalPodAutoscaler (HPA) changes the number of pod instances depending on metrics like as CPU and memory utilisation, while VerticalPodAutoscaler (VPA) modifies pod CPU and memory requests based on past and present resource utilisation (Tamiru et al. 2020; The Kubernetes Authors 2023).

```
template:
  common:
    autoscaling:
      enabled: false
      minReplicas: 1
      maxReplicas: 100
      targetCPUUtilizationPercentage: 80
      targetMemoryUtilizationPercentage: 80
```

Figure 5.6: Processing-API auto-scaling configuration

Processing-API currently does not support auto-scaling, as seen in figure 5.6, therefore it will only deploy the amount of replicas specified in the configuration files, regardless of CPU consumption. Enabling HPA and VPA - raising the number of pods until a particular limit (for example, until there are 20 replicas) when the CPU and memory utilisation hits 80% and then adjusting the replicas resources based on their utilisation - may be an interesting optimisation technique to be employed and studied.

5.1.6 AWS Optimisation

AWS is a provider of cloud computing solutions, that is the most trusted provider at the moment, and offers on-demand services, flexibility and quick development and deployment

(Narula, Jain, and Prachi 2015). AWS hosts the SaltPay cloud infrastructure, where all Payments Gateway services have been set up and made available.

Amazon Elastic Compute Cloud (EC2) is one of the services provided by Amazon. It is a web service that provides resizable computational power, making web-scale computing easier, whose primary features are elasticity and scalability, interaction with other AWS services, and a pay-as-you-go pricing mechanism.

Similar to what was described in regards to Kubernetes, auto-scaling is a feature that is also offered by EC2 that dynamically modifies the number of instances available based on the demand of the applications, and plays a crucial role in cloud computing.

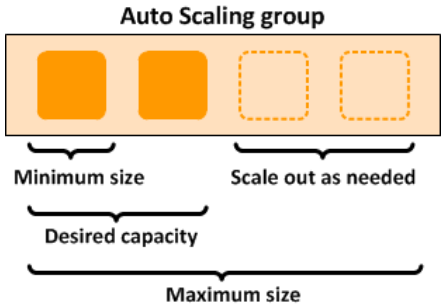

Figure 5.7: AWS Auto-scaling Group (from: Amazon Web Services (2023))

A few key components must be set up in order for this functionality to take effect.

- **Auto-Scaling Group**: A collection of instances with identical characteristics and scaling policies, the size of which is determined by the number of instances specified as the necessary maximum capacity. An example can be seen in figure 5.7;
- **Scaling Policies**: Determine how the resources are scaled, it can be maintaining current instance levels at all times, scaling manually, scaling based on a schedule, scaling based on demand and using predictive scaling;
- **Configuration Templates**: Configurations that specify the settings required for the instances

AWS EC2 auto-scaling, like Kubernetes auto-scaling, is not enabled in any of the Payments Gateway services. Predictive scaling, which employs machine learning to estimate capacity requirements based on historical data from CloudWatch, is an intriguing alternative for EC2 auto-scaling and should be examined for all services, particularly Processing-API (Amazon Web Services 2023). The difficulty is determining how and when to activate the auto-scaling mechanism. Defining the thresholds is not a precise process, and it needs much planning and investigation.

Liao, Kuai, and Leau (2015) proposes two auto-scaling strategies, Dynamic Threshold Adjustment Mechanism and Dynamic Threshold Adjustment Strategy, the results of which show that they may have a significant impact on web-based CPU-intensive applications that consume a large amount of CPU resources per request - as is the case when transaction load increases. The study of these strategies, as well as the options offered by AWS should be equated, in order to improve the performance of the services while optimising resources and their cost.

5.2 Implementation Suggestion

The above-mentioned solutions could possibly be employed and the load tests repeated to measure the improvement (or worsening) in the transaction processing time. All suggested optimisation strategies implementation are described in depth in the sections that follow.

5.2.1 RabbitMQ Connection Reuse

Processing-API, as described in section 5.1.4, sends audit-log events to RabbitMQ so that they may be sent to and read by other services, allowing for nearly full synchronisation across all services. At the moment, every time an event must be published to RabbitMQ, a new connection is established and then closed, which calls for the establishment of a new connection each time an event message must be published.

Considering that creating a new connection is a fairly costly step in the RabbitMQ connection lifecycle, the client's present implementation should be improved so that connections may be reused for multiple requests.

```
class PikaClient(object):
    """
    Wrapper around the RabbitMQ library in order to add features such as support for multiple hosts
    """

    def __init__(self, exchange_name, rabbitmq_user, rabbitmq_password, rabbitmq_broker_hosts, rabbitmq_broker_port,
                 rabbitmq_socket_timeout=10.0, rabbitmq_connection_attempts=1, rabbitmq_retry_delay=2.0,
                 rabbitmq_heartbeat=60, rabbitmq_ssl=False,
                 exchange_type='fanout', durable=True, auto_delete=False, routing_key='', queues=None,
                 event_bus_logger=None, message_ttl=None, queues_arguments=None):
        """
        Constructor initializes a connection with the broker
        """
        credentials = pika.PlainCredentials(rabbitmq_user, rabbitmq_password)

        # Instance variables
        self.auto_delete = auto_delete  # Auto delete the queue ?
        self.channel = None  # The RabbitMQ connection channel.
        self.connection = None  # The RabbitMQ connection object.
        self.durable = durable  # Is a durable queue ?
        self.exchange_name = exchange_name  # The RabbitMQ exchange name. Always declare the exchange.
        self.properties = None  # Publishing properties.
        self.queues = queues  # The queues to declare and bind to the exchange.
        self.exchange_type = exchange_type  # Queue type. By default is fanout.
        self.routing_key = routing_key
```

Figure 5.8: Snippet of the current Pika client implementation

The current client implementation supports a single connection object, as seen on figure 5.8. A novel solution that takes use of connection pooling might be interesting for reusing connections. Rather of making a new connection for each incoming request, the overhead of opening the connection might be decreased by building a connection pool that maintains a group of reused connections.

```python
class PikaPoolClient(object):
    """
    Wrapper around the RabbitMQ library in order to add features such as support for multiple hosts
    """

    def __init__(self, exchange_name, rabbitmq_user, rabbitmq_password, rabbitmq_broker_hosts, rabbitmq_broker_port,
                 rabbitmq_socket_timeout=10.0, rabbitmq_connection_attempts=1, rabbitmq_retry_delay=2.0,
                 rabbitmq_heartbeat=60, rabbitmq_ssl=False,
                 exchange_type='fanout', durable=True, auto_delete=False, routing_key='', queues=None,
                 event_bus_logger=None, message_ttl=None, queues_arguments=None):
        """
        Constructor initializes a connection with the broker
        """
        credentials = pika.PlainCredentials(rabbitmq_user, rabbitmq_password)

        # Connection pool
        self.connections = Queue()

        # Instance variables
        # Declare instance variables here
        # ...
```

Figure 5.9: Suggested implementation of a Pika client with a connection pool

Figure 5.9 shows a small modification to the Pika client implementation, in which a Queue object named *connections* was created to keep the reusable connections to RabbitMQ. In addition, as illustrated in Figure 5.10, a method for acquiring the connection and a method for releasing the connection to the pool must be defined.

```python
    def get_connection(self):
        if not self.connections.empty():
            return self.connections.get()
        else:
            connection = pika.BlockingConnection(
                pika.ConnectionParameters(
                    host=self.host,
                    port=self.port,
                    credentials=pika.PlainCredentials(self.username, self.password)
                )
            )
            return connection

    def release_connection(self, connection):
        self.connections.put(connection)
```

Figure 5.10: Connection pool Pika client methods that acquire and release the connection

The first method, *get_connection*, checks to see whether there are any available connections in the queue, and if so, retrieves one and returns it. If the queue is empty, a new BlockingConnection with the connection configurations given in the constructor is established. The BlockingConnection is a blocking operation, which means it will not proceed until the connection is established. After that, the newly established connection is returned. The *release_connection* method simply returns the connection to the queue, making it ready for use in the following request.

Instead of generating a new connection each time a message from Processing-API needs to be transmitted, the pool might be utilised to reuse connections, which will enhance performance and efficiency. Figure 5.11 depicts the recommended adjustments regarding the call for message publishing.

Figure 5.11: Suggested change to reuse connections to RabbitMQ

5.2.2 Processing-API Resources Adjustments

The results of the load testing in Chapter 4 revealed that the Processing-API was unquestionably the bottleneck of all the services involved in a transaction lifecycle. Analysing CPU utilisation and throttling, which reached 100% at times, revealed that this service had a significant influence at the number of requests reached and the time it took to process them.

As discussed in chapter 4 and section 5.1.4, the fact that the development environment's resources are more modest than those of the production environment should not explain why just a few transactions per second were achieved. Rather, the source code might not be as optimised as it could be.

The production environment currently has 12 four-core pods, whereas the development environment has just two single-core pods. The development environment pods have the potential to be upgraded to six four-core pods, resulting in half the processing capability of the production environment.

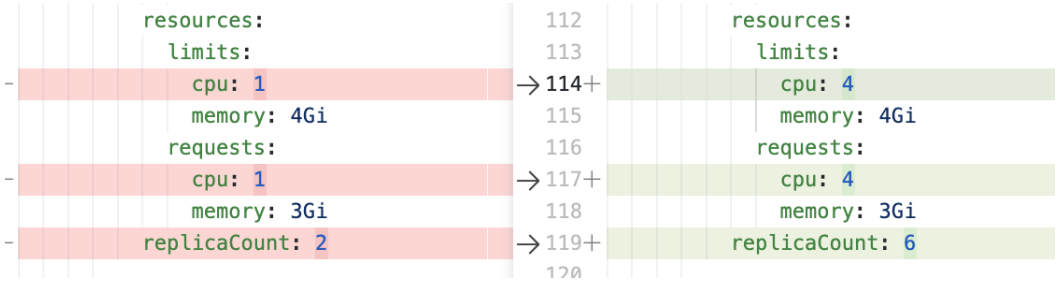

Figure 5.12: Suggested upgrade to Processing-API resources

With the upgrade proposed on figure 5.12, the results obtained in the load tests should be better than the ones recorded and explores in chapter 4, as the service would theoretically have half the processing power of the production environment.

In addition to increasing the resources available, another implementation suggestion is switching from Web Server Gateway Interface (WSGI) to Asynchronous Server Gateway Interface (ASGI). The primary deployment platform of Django is WSGI, which is a standard for web and application servers to connect and interoperate with other web applications and servers (Django Software Foundation 2023a; Thomas 2008). ASGI is the successor

to WSGI, and aims to provide a standard interface between async-capable Python web servers, being more suitable for building high-performance asynchronous web applications (ASGI Team 2018).

The changes necessary to perform in order to accommodate ASGI are rather simple. Currently, Gunicorn, a WSGI server, is used to deploy the application. In addition to replacing the current configuration file, *wsgi.py* with a ASGI configuration file, *asgi.py*, the server used to deploy the application must have its configurations changed, in order to accommodate the appropriate web server. Uvicorn, an ASGI web server implementation for Python, is backwards compatible with Gunicorn, and can be used to deploy the application.

The modifications required for the implementation of ASGI are relatively simple. Gunicorn, a WSGI server, is being utilised to deploy the application. Aside from replacing the present configuration file, *wsgi.py*, with an ASGI configuration file, *asgi.py*, the server used to deploy the application's settings must be adjusted to accommodate the proper web server. Uvicorn, an ASGI web server implementation for Python, is backwards compatible with Gunicorn, and may be used to deploy the application.

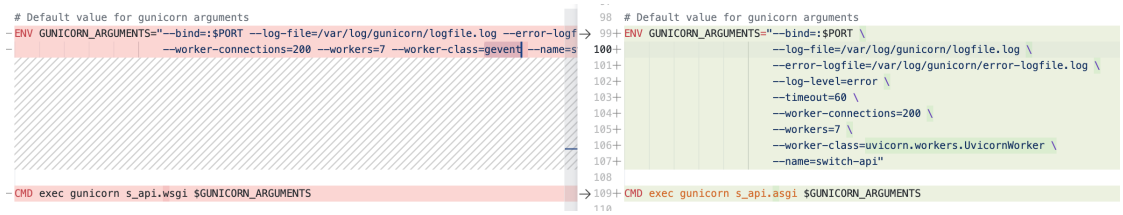

Figure 5.13: Suggested changes to Processing-API Dockerfile, to support Uvicorn

As shown in figure 5.13, changing the worker class in Gunicorn is all that is required to utilise Django with Uvicorn and successfully deploy the application. Additionally, optimising Gunicorn settings could also be a strategy to be implemented, in order to fully enhance the performance of the web application.

5.2.3 Auto-Scaling

Explored in section 5.1.5 and 5.1.6, auto-scaling adjusts the resources available based on current and past metrics results, and it is available both in K8s and AWS. Defining the threshold when resources should be scaled up is not a trivial task, and is one of the biggest challenges currently faced in cloud computing (Qu, Calheiros, and Buyya 2018).

In terms of K8s, the HPA should be activated in order to automatically update the number of available pods.

Taking into account that the auto-scaler should only be enabled when resources reach a certain threshold, the target average CPU utilisation percentage and target average memory utilisation percentage should be set to 80 percent - which signifies that HPA will scale the target whenever the resources available reach average CPU and memory utilisation percentages above 80 percent. Additionally, there are currently 12 pods in the production environment; the maximum number of replicas to which the auto-scaler can scale up could possibly be set to 20. The configurations for the auto-scaling to be applied can be seen in figure 5.14.

```yaml
common:
  autoscaling:
    enabled: true
    minReplicas: 1
    maxReplicas: 20
    targetCPUUtilizationPercentage: 80
    targetMemoryUtilizationPercentage: 80
```

Figure 5.14: Suggested Kubernetes auto-scaling configuration

Moreover, AWS auto-scaling enables the automated modification of EC2 capacity depending on demand, ensuring that the application can manage variable amounts of traffic while optimising resource utilisation and costs.

To enable this feature, administrators must initially create an auto-scaling group and then define the right thresholds that trigger this behaviour, among other elements, as stated in section 5.1.6. In the production environment, there currently exist 6 instances of Processing-API. An auto-scaling group could be setup with a minimum of 6 instances and a maximum of 10. AWS auto-scaling will ensure that the group never falls below or exceeds these numbers.

Pairing resource modification along with both auto-scaling techniques might be a potential strategy for improving application performance and increasing the number of requests serviced.

Chapter 6

Conclusions

The project documented in this dissertation had, as its primary objective, the identification of existing limitations in SaltPay's internal Payments Gateway and Acquiring Host, as well as the proposal of potential solutions to these issues, in order to improve system performance, in general.

This last chapter will analyse all of the work completed for this project and provide final thoughts on it. The goals attained, issues experienced throughout the course of action, and future tasks indicated will all be addressed. Last but not least, final and personal thoughts on the project in question will be provided.

6.1 Objectives Completed

The research conducted throughout the course of this project, as well as the analysis of the findings acquired from the load testing, allowed for a positive contribution to the main goal of identifying the points of latency introduced within the scope of a transaction.

Given the project's organisational regulations and the magnitude of the systems that were examined, the results obtained allow it to safely be stated that, in addition to the solid foundation of work established and suggestions for improvement explored, the vast majority of the objectives mentioned in chapter 1.3 were achieved.

Nevertheless, based on the research and sufficient evidence provided in this dissertation, there is space for improvement and further development in general.

Overall, notwithstanding the fact that not all of the goals were fully satisfied (such as implementing all of the proposed improvements and testing again the systems), the overall assessment may be regarded successful.

6.2 Challenges Encountered

Although it was considered that the major objectives of this dissertation were met in general, there were some restrictions that made obtaining and analysing the data challenging.

The presence of two Acquiring Hosts in SaltPay's domain, Way4 and Solar, was indicated, although the tests only took place on Way4, as reported in chapter 4. Given the inconsistent behaviour of Solar's development environment, it was chosen not to run the tests with this Acquiring Host and instead use Way4, which has a stabilised development environment.

While Way4's development environment is reliable, there are limits owing to the usage of card issuer simulators, which involves Visa and Mastercard. In accordance with Way4 information, Mastercard's simulator can only take one transaction per second, but Visa's can accept between 80 and 90 transactions per second. These figures are related to the fact that the Mastercard simulator is in a machine separate from the production environment, with fewer resources, and is controlled by Way4, whereas the Visa simulator is in the same production environment and is managed by Visa, resulting in greater processing power and resources.

Additionally on the topic of the disparities and limitations of the environments, it is worth noting that the Payments Gateway's development and production environments are quite distinct - in particular, the Processing-API, the "core" responsible for creating transactions and communicating with Acquiring Hosts, has 1/6 of the existing pods in production in the development environment, along with to the machines' processors being substantially inferior as well. This implies that the development environment will always operate slower than the production one, thus transactions will take longer to complete and fewer transactions per second will be created. The findings reached in this study are the product of extrapolating and forecasting results, which is a dangerous strategy in any scenario.

Along with to the concerns listed above, the Payments Gateway infrastructure was moved towards the end of February 2023, becoming part of the SaltPay infrastructure. While the migration had been successful, with only a few minutes of downtime, due to the differences in infrastructure, there were some incidents in transaction processing, which resulted in an increase in the time that each transaction took to process - specifically, MongoDB was used in the old infrastructure, whereas DocumentDB is used in SaltPay's infrastructure, among other nuances. Transactions experienced a slight impact while the new environment settings were not adjusted. These defects, nevertheless, were addressed, and the system was working properly once more.

Last but not least, considering the load tests were conducted on a personal computer (MacBook Pro M1 2021), which has far fewer resources than a cloud machine, the findings obtained may have been influenced by the source machine's lack of computing capability. Nonetheless, the findings are regarded as a foundation for future study, with conclusions extrapolated in accordance with predictions.

6.3 Future Work

As a whole, it is considered that this dissertation was successful in both getting conclusions and fulfilling the intended goals. Yet, there always exist ways that can enhance and complement what has been achieved. Although the proof of concept and proposals have been fulfilled, the magnitude of the company's aims involves a plethora of additional activities for performance enhancement. The key enhancements noticed relate to the implementation of proposed solutions and the repetition of tests to carry out additional data analysis.

While this proof of concept may be utilised as a foundation for the product, it is recommended that other approaches should be explored - the re-writing of the socket proxies, software module identified with the anti-pattern *Spaghetti Code*, that is, a remarkably high complexity for comprehension and maintenance; general simplification of calls made within the scope of a transaction, in order to remove potential unnecessary instructions, such as the decoding of EMV data, which will be encoded again in later steps. As a complement to the preceding

approaches, it is recommended that load testing should become an automated and scheduled procedure, performed directly in the cloud, to establish a certain consistency and facilitate the analysis of critical points in the system, as well as to eliminate any limitations in personal devices.

Lastly, it should be mentioned that an internal initiative is underway to restructure the Payment Gateway. The primary goal of this project is to create a new gateway that is capable of high performance, completing transactions in less than one second, and has a lower cost of maintenance compared to the existing gateway. Even though this project is still in the early stages of development, it represents a promising initiative, having the same aims as this dissertation - reducing transaction processing times while the system usage grows every day.

6.4 Final Considerations

On a technical level, this dissertation offers useful details on the present status of the Payments Gateway micro-services architecture, most notably its crucial components. The discovered concerns are critical areas for developers who are presently maintaining the project and encourage further research.

Ideally, this dissertation presents the necessary evaluation and recommendations so that any system with an architecture based on micro-services comparable to the Payments Gateway can be optimised, and most importantly, provide the appropriate data for the organisation's developers to be able to adapt the current solution and hopefully improve transaction processing.

On a personal level, completing this dissertation and all of the research it required enabled me to further develop technical and personal skills, as well as gain a more thorough and formulated understanding of all the systems with which I had to interact and study in order to carry out this study and achieve the intended objectives.

To put things into perspective and conclude, I believe that the development of this dissertation benefited not only SaltPay, as well as myself, allowing me to be pleased with the outcomes that were obtained, despite the constraints that existed in the implementation of the suggestions.

Bibliography

Amazon Web Services (2023). *AWS Documentation*. Last accessed 20 May 2023. url: https://docs.aws.amazon.com/.

Apache Software Foundation (2022). *Apache JMeter™*. Last accessed 25 January 2023. url: https://jmeter.apache.org/.

ASGI Team (2018). *ASGI Documentation*. Last accessed 20 May 2023. url: https://asgi.readthedocs.io/.

Atlassian (2022). *Scrum - Learn how to scrum with the best of 'em*. Last accessed 01 December 2022. url: https://www.atlassian.com/agile/scrum.

Bagchi, Sugoto and Bill Tulskie (2000). "E-business models: integrating learning from strategy development experiences and empirical research". In: *20th Annual International Conference of the Strategic Management Society*, pp. 15–18.

Banco de Portugal (2022). *Report on Payment Systems 2021*. Tech. rep. Banco de Portugal.

Bao, Wenlei et al. (Dec. 2016). "Static and Dynamic Frequency Scaling on Multicore CPUs". In: *ACM Trans. Archit. Code Optim.* 13.4. issn: 1544-3566. doi: 10.1145/3011017.

Beizer, Boris (1984). *Software System Testing and Quality Assurance*. USA: Van Nostrand Reinhold Co. isbn: 0442213069.

Bowman, Cliff and Véronique Ambrosini (2000). "Value Creation Versus Value Capture: Towards a Coherent Definition of Value in Strategy". In: *British Journal of Management* 11.1, pp. 1–15. doi: https://doi.org/10.1111/1467-8551.00147.

Brytfmonline, Andrea Hargraves (2021). *monopoly? Foreign 'Fintech' files complaint with Banco de Portugal against SIBS – Observer*. Last accessed 20 December 2022. url: https://www.brytfmonline.com/monopoly-foreign-fintech-files-complaint-with-banco-de-portugal-against-sibs-observer/.

CGD (2023). *CGD Website*. Last accessed 21 January 2023. url: https://www.cgd.pt/Empresas/Gestao_corrente/Servicos/Pages/Solucao-TPA-Caixa.aspx.

Chacon, Scott and Ben Straub (2022). "Pro Git". In: 2nd ed. Apress. Chap. 1.

Chan, Lai-Kow and Ming-Lu Wu (2002). "Quality function deployment: A literature review". In: *European Journal of Operational Research* 143.3, pp. 463–497. issn: 0377-2217. doi: https://doi.org/10.1016/S0377-2217(02)00178-9.

Coutinho, Emanuel Ferreira, Danielo Gonçalves Gomes, and José Neuman de Souza (2015). "An Autonomic Computing-based architecture for cloud computing elasticity". In: *2015 Latin American Network Operations and Management Symposium (LANOMS)*, pp. 111–112. doi: 10.1109/LANOMS.2015.7332681.

Davidson, E Jane (Jan. 2004). *Evaluation Methodology Basics: The Nuts and Bolts of Sound Evaluation*. isbn: 0761929304. doi: 10.4135/9781452230115.

Django Software Foundation (2023a). *Django*. Last accessed 20 May 2023. url: https://www.djangoproject.com/.

– (2023b). *Django Documentation - Performance and optimization*. Last accessed 20 May 2023. url: https://docs.djangoproject.com/en/4.2/topics/performance/.

European Central Bank (2022). *Payments Statistics*. Tech. rep. Eruopean Central Bank.

Flutterwave (2023). *Flutterwave Website*. Last accessed 21 January 2023. url: https://flutterwave.com/us/.

Garg, Rajat P. and Illya Sharapov (2001). *Techniques for Optimizing Applications: High Performance Computing*. USA: Prentice Hall PTR. isbn: 0130934763.

Grafana Labs (2023a). *Grafana*. Last accessed 17 April 2023. url: https://grafana.com/.

– (2023b). *K6*. Last accessed 25 January 2023. url: https://k6.io/.

Haksever, Cengiz, Radha Chaganti, and Ronald G. Cook (Feb. 2004). "A Model of Value Creation: Strategic View". In: *Journal of Business Ethics* 49.3, pp. 295–307. issn: 1573-0697. doi: 10.1023/B:BUSI.0000017968.21563.05.

Herbst, Nikolas Roman, Samuel Kounev, and Ralf Reussner (2013). "Elasticity in Cloud Computing: What It Is, and What It Is Not". In: *10th International Conference on Autonomic Computing (ICAC 13)*. San Jose, CA: USENIX Association, pp. 23–27. isbn: 978-1-931971-02-7.

Hu, Tengfei and Yannian Wang (2021). "A Kubernetes Autoscaler Based on Pod Replicas Prediction". In: *2021 Asia-Pacific Conference on Communications Technology and Computer Science (ACCTCS)*, pp. 238–241. doi: 10.1109/ACCTCS52002.2021.00053.

Huo, Qizheng et al. (2022). "Horizontal Pod Autoscaling based on Kubernetes with Fast Response and Slow Shrinkage". In: *2022 International Conference on Artificial Intelligence, Information Processing and Cloud Computing (AIIPCC)*, pp. 203–206. doi: 10.1109/AIIPCC57291.2022.00051.

Ionescu, Valeriu Manuel (2015). "The analysis of the performance of RabbitMQ and ActiveMQ". In: *2015 14th RoEduNet International Conference - Networking in Education and Research (RoEduNet NER)*, pp. 132–137. doi: 10.1109/RoEduNet.2015.7311982.

Jiang, Zhen and Ahmed E. Hassan (Nov. 2015). "A Survey on Load Testing of Large-Scale Software Systems". In: *IEEE Transactions on Software Engineering* 41, pp. 1–1. doi: 10.1109/TSE.2015.2445340.

Jiang, Zhen Ming and Ahmed E. Hassan (2015). "A Survey on Load Testing of Large-Scale Software Systems". In: *IEEE Transactions on Software Engineering* 41.11, pp. 1091–1118. doi: 10.1109/TSE.2015.2445340.

Koen, Peter et al. (2001). "Providing Clarity and A Common Language to the "Fuzzy Front End"". In: *Research-Technology Management* 44.2, pp. 46–55. doi: 10.1080/08956308.2001.11671418.

Koen, Peter A., Greg M. Ajamian, et al. (2002). "Fuzzy Front End: Effective Methods, Tools, and Techniques". In: *The PDMA ToolBook for New Product Development*.

Koen, Peter A., Heidi M. J. Bertels, and Elko Kleinschmidt (2014). "Managing the Front End of Innovation—Part I: Results From a Three-Year Study". In: *Research-Technology Management* 57.2, pp. 34–43. doi: 10.5437/08956308X5702145.

Lavrakas, Paul (2008). *Encyclopedia of Survey Research Methods*. SAGE Publications, Inc.

Liao, Wen-Hwa, Ssu-Chi Kuai, and Yu-Ren Leau (2015). "Auto-scaling Strategy for Amazon Web Services in Cloud Computing". In: *2015 IEEE International Conference on Smart City/SocialCom/SustainCom (SmartCity)*, pp. 1059–1064. doi: 10.1109/SmartCity.2015.209.

Lindgreen, Adam and Finn Wynstra (2005). "Value in business markets: What do we know? Where are we going?" In: *Industrial Marketing Management* 34.7. IMP Conference, Rotterdam, 2004, pp. 732–748. issn: 0019-8501. doi: https://doi.org/10.1016/j.indmarman.2005.01.001.

Locust (2023). *Locust*. Last accessed 25 January 2023. url: https://locust.io/.

Menasce, D.A. (2002). "Load testing of Web sites". In: *IEEE Internet Computing* 6.4, pp. 70–74. doi: 10.1109/MIC.2002.1020328.

Mention, Anne-Laure (2019). "The Future of Fintech". In: *Research-Technology Management* 62.4, pp. 59–63. doi: 10.1080/08956308.2019.1613123.

Narula, Saakshi, Arushi Jain, and Prachi (2015). "Cloud Computing Security: Amazon Web Service". In: *2015 Fifth International Conference on Advanced Computing & Communication Technologies*, pp. 501–505. doi: 10.1109/ACCT.2015.20.

Neap, Halil Shevket and Tahir Celik (1999). "Value of a Product: A Definition". In: *International Journal of Value-Based Management* 12, pp. 181–191.

O Jornal Económico, Revista de Imprensa JE (2021). *Fintech estrangeira acusa SIBS de monopólio com queixa no Banco de Portugal*. Last accessed 20 December 2022. url: https://jornaleconomico.pt/noticias/fintech-estrangeira-acusa-sibs-de-monopolio-com-queixa-no-banco-de-portugal-769265.

Osterwalder, Alexander and Yves Pigneur (2003). "Modeling Value Propositions in E-Business". In: *Proceedings of the 5th International Conference on Electronic Commerce*. ICEC '03. New York, NY, USA: Association for Computing Machinery, pp. 429–436. isbn: 1581137885. doi: 10.1145/948005.948061.

– (2010). *Business model generation: a handbook for visionaries, game changers, and challengers*. Vol. 1. John Wiley & Sons.

Pallets (2023). *Flask Documentation*. Last accessed 20 May 2023. url: https://flask.palletsprojects.com/en/2.3.x/.

Prometheus (2023). *Prometheus*. Last accessed 17 April 2023. url: https://prometheus.io/.

PWC (2021). *Navigating the payments matrix - Charting a course amid evolution and revolution*. Tech. rep. PWC.

Python Software Foundation (2023). *Python Wiki - Performance Tips*. Last accessed 20 May 2023. url: https://wiki.python.org/moin/PythonSpeed/PerformanceTips.

Qu, Chenhao, Rodrigo N. Calheiros, and Rajkumar Buyya (July 2018). "Auto-Scaling Web Applications in Clouds: A Taxonomy and Survey". In: *ACM Comput. Surv.* 51.4. issn: 0360-0300. doi: 10.1145/3148149.

Rao, Singiresu S. (2009). *Engineering Optimization: Theory and Practice*. John Wiley & Sons. isbn: 9781119454816. doi: 10.1002/9781119454816.

Rich et al., Nick (2000). *Value Analysis, Value Engineering*. Tech. rep. INNOREGIO Project.

SaltPay Learning Team (2022). *Introduction to Card Payments*. Last accessed 20 January 2023. url: https://saltpayco.atlassian.net/wiki/spaces/NJS/pages/521798032/Introduction+to+Card+Payments.

Schwaber, Ken and Jeff Sutherland (2020). *The Scrum Guide - The Definitive Guide to Scrum: The Rules of the Game*.

Senart, Tomás (2022). *Vegeta*. Last accessed 25 January 2023. url: https://github.com/tsenart/vegeta/.

Statista Research Department (2022). *Total value of investments into fintech companies worldwide from 2010 to H1 2022*. Last accessed 2 February 2023. url: https://www.statista.com/statistics/719385/investments-into-fintech-companies-globally/.

Switch (2021). *Who's Who: the Players in the Card Payments Industry*. Last accessed 15 January 2023. url: https://switchpayments.com/learn/609ceb4366cfb30013d58cdb.

SwitchPayments (2023). *Risk - Detect and block any fraud*. Last accessed 20 May 2023. url: https://switchpayments.com/docs/risk.

Tamiru, Mulugeta Ayalew et al. (2020). "An Experimental Evaluation of the Kubernetes Cluster Autoscaler in the Cloud". In: *2020 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pp. 17–24. doi: 10.1109/CloudCom49646.2020.00002.

Tanenbaum, Andrew S. and Herbert Bos (2014). *Modern Operating Systems*. 4th ed. Pearson. isbn: 978-0-13-359162-0.

Tanenbaum, Andrew S. and Maarten van Steen (2006). *Distributed Systems: Principles and Paradigms*. 2nd ed. USA: Prentice-Hall, Inc. isbn: 0132392275.

Teya Help Center (2023). *Pay by Link*. Last accessed 17 April 2023. url: https://help.teya.com/articles/payments-on-card-machines/pay-by-link/6405f6699068fb44e4589d70.

The Kubernetes Authors (2023). *Kubernetes Documentation*. Last accessed 20 May 2023. url: https://kubernetes.io/docs/.

Thomas, Mary P. (2008). "Using the Pylons Web Framework for Science Gateways". In: *2008 Grid Computing Environments Workshop*, pp. 1–9. doi: 10.1109/GCE.2008.4738447.

Tohang, Valentina, Ahmad Seiichi Ramadhan, and Vincentius Djajadiningrat (2021). "An Empirical Study on Customer Acceptance of FinTech 3.0 in Private Banking". In: *2021 International Conference on Information Management and Technology (ICIMTech)*. Vol. 1, pp. 105–110. doi: 10.1109/ICIMTech53080.2021.9535074.

Ulaga, Wolfgang and Andreas Eggert (2006). "Value-Based Differentiation in Business Relationships: Gaining and Sustaining Key Supplier Status". In: *Journal of Marketing* 70.1, pp. 119–136. issn: 00222429. url: http://www.jstor.org/stable/30162077.

VisualParadigm (2022). *What is Scrum's Three Pillars?* Last accessed 01 December 2022. url: https://www.visual-paradigm.com/scrum/what-are-scrum-three-pillars/.

VivaWallet (2023). *VivaWallet Website*. Last accessed 21 January 2023. url: https://www.vivawallet.com/pt_en/.

VMWare Inc. (2023). *RabbitMQ Documentation*. Last accessed 20 May 2023. url: https://www.rabbitmq.com/documentation.html.

Yoco (2023). *Yoco Website*. Last accessed 21 January 2023. url: https://www.yoco.com/za/.