POLITECNICO DI TORINO

POLITECNICO DI TORINO

Master Degree course in Master Degree course in Computer Engineering

Master Degree Thesis

Design and Implementation of an Advanced Monitoring System for alert management and log analysis in a mobile payment app

Supervisors
Prof. Riccardo Coppola

Candidate
Alessandro Genco

Academic Year 2024-2025

Abstract

The thesis work has been conducted with the collaboration of Nexi Digital S.r.l., inside a team adopting the SAFe framework for Agile development methodology, which is a methodology used in large organizations with multiple teams cooperating among themselves, to develop complex products in a very quick and continuous way, using a specific set of technologies.

The case study is about the MyPayments application, which manages merchants' data and different kinds of PoS implementations. This application is in the digital payments market and allows to manage physical PoS terminals, but also soft PoS terminals, which are increasing over time also because they work on common smartphones.

Although the application has a very extended backend, including databases, the whole system has some technical issues to be solved, like absences of services or high waiting times inside the application under specific conditions. Specifically, the objective of the work is to identify these anomalies and improve the reaction time to these issues, thanks to a real-time alerting system, also in order to increase the user experience.

The methodology applied in this work consists in analyzing potential solutions with different approaches, and then suitable technologies for the chosen approach; describing how the parsing of the raw log data has been done, the partial output structure, which is used by the machine learning model to be trained, and how the training and execution of the model are implemented, using two different scripts.

The results obtained are positive and respect the given requirements, since the anomalies have been successfully identified with very high recall. The benefits of the results in terms of user experience and reactiveness to issues are evident, because the anomaly alert would be received by the team as soon as an issue is identified, so that it can be solved in the least possible time.

As a conclusion, the proposed system can be implemented inside the Nexi's infrastructure in the future, both inside the Cloud or in any virtual machine. The analysis and design of the entire system consists of a RabbitMQ implementation, including the scripts defined to parse raw logs and to execute the XGBoost model, with the script to send email alerts. It will also be possible to extend the machine learning model to identify new anomalies or malicious patterns which may be potentially insecure for the backend servers and applications.

Contents

Chapter 1

Introduction

MyPayments application is a service provided by the Nexi Group and developed by Nexi Digital S.r.l. This application is supported by a variety of backend services with different purposes. The merchants using the application are experiencing technical issues with both the iOS and Android application, like absence of services or temporary malfunctions.

The objective of this work is to design and implement a solution that supports the team managing some of the backend servers, which needs to identify and be notified of problems as soon as they occur, to reduce the reaction time and improve the user experience of the application.

During the work, different possible solutions are described and compared; the best fitting one according has been chosen, according to the needs and requirements of the team. This solution is described at various levels, from the design one to the actual implementation. Moreover, it is both theoretically and practically relevant due to the possible approaches, such as machine learning algorithms among other ones.

It is important to notice that some information, like services' names, IP addresses, security configurations, additional functionalities, the infrastructure architecture, and so on have been omitted as agreed with the company partner of this work, to not expose the infrastructure and services to security threats and also to not make public, internal information related to the application and the entire ecosystem for several reasons.

A future implementation of the solution has also been discussed, considering and comparing various available technologies, with respect to the requirements of the team involved in the work. The chosen approach has been designed and then described at a more practical level, including the working mechanisms and the expected results.

The chapters of the thesis work are described in the following list:

1. Background: an overview of the team and the company is provided, as well as the software development methodology used in the team involved in this thesis work. Some concepts needed to work inside the team are explained, together with the entire ecosystem of teams working on the company projects, always synchronizing among themselves. In the end, the technologies involved in the study are briefly described;

2. Case study: MyPayments application is described, together with some technical

concepts related to the digital payments domain, like the different types of PoS. The overall architecture of the application and the backend is described, specifically explaining the internal backend architecture, which is directly managed by the team involved in this work. The known technical issues of the application and the backend are also described;

3. Methodology: given the technical issues, some potential solutions are analyzed and compared among themselves, choosing a specific one. After that, this solution has been analyzed and designed at a lower level, until reaching the actual implementation of the final solution, which is described, including all the relevant details about the code and the complexity managed over the work.

4. Results: the numerical results and performance obtained are discussed, split by the objectives reached regarding the solved problem. Moreover, the methodologies used during the various operations done are also described, giving details on how the work has been executed.

5. Conclusion and future implementations: the result obtained can still be extended inside a system with specific requirements needed to be satisfied to meet the needs of the team involved in this work. Possible approaches are discussed and compared with technical details, reaching a choice. The design of the system based on this choice is explained, as well as the working mechanisms inside itself and the expected results from the overall system.

Chapter 2

Background

The company in which the work has been completed is Nexi Digital S.r.l., which is an Italian company, part of the Nexi Group and works on several of its products internally, in order to achieve the best possible results by focusing the development of the applications and services on the business orientation the company has.

2.1 Team description

In particular, I was part of a team called ApeCar team, within a large set of teams, taking part in a much larger project. In my team, there are iOS developers, Android developers, a backend engineer, and two people working on quality assurance for the frontend product, which is the app. There have also been two testing automation developers, who worked in the team until they needed to automate some tests which were not automated yet. Apart from technical personnel, there also is a manager coordinating the ApeCar team and some others within Nexi Digital, a product owner communicating with stakeholders, such as various product managers for the app the team works on, each one for a set of specific aspects of the app, and a scrum master to ensure that the SAFE Agile framework is correctly applied. The team cooperates with nine other teams to contribute to a much larger project, which aims to provide several services and products to Nexi's customers.

2.2 My experience

I had the opportunity to know all the people inside my team, and some other people from other teams, because sometimes our work was not related to the work of some other teams. The team in which I worked has a very positive and optimistic approach, all members cooperate as much as they can to achieve the objectives they have set for the product. I also knew people from other teams and discussed several times with them, in order to solve problems with dependencies or incidents which have happened. These interactions have been very important in order to exhaustively understand how all the other systems to which the MyPayments App backend is connected. In some situations, I have also worked together with developers and members of other teams to complete cross-team user stories or tasks. I think it has been very positive for me to do this,

because I also understood a bit more of the project they work on, and also the extension of the entire infrastructure and all the services inside itself.

2.3 SAFe Agile framework

As earlier mentioned, several teams are cooperating in order to achieve the objectives of a much larger project. All of these teams are part of the ART, which means Agile Release Train. The ART has the objective of providing new features and improvements to the products and services managed by the teams within it, by giving priority to each of them. The features are described by the epics, which contain several user stories to be implemented in order to release the complete feature. In some cases, there are dependencies among the different teams to complete user stories, so if there are some problems during their development, other teams may also be affected by them. All the features and improvements are driven by business choices, requesting to go towards a specific direction within the market among all the products and services, but also specifically for every single one. After having defined these things at the ART level, every team has its own objectives to realize and can work on them, with its software development methodology. Since every team can choose and adapt the best-fitting software development methodology for its work, a common methodology is needed at the ART level, in order to coordinate in a very efficient and effective way all the teams taking part in it.

2.3.1 ApeCar team methodology

The team in which I work chose the software development methodology which fit more to its needs, and in this case, it is the Scrumban, which I have already learned and applied inside the Software Engineering II course, within a team of five other colleagues, apart from some aspects related to the Kanban methodology. Before joining the team, I already felt confident about the Scrum actual application, thanks to the project developed all over the course at University and the support of the professors. The ApeCar team has to do different ceremonies inside the Scrum, at different moments during the software development process. The most frequent ceremony is the Daily Scrum meeting, which lasts around fifteen minutes and all the team participates in it, including the Product Owner and the Scrum Master. Moreover, there also are other ceremonies, as we already know, which are the Sprint planning, the Sprint review or Demo, which is performed at the end of the iteration, the team retrospective, which will be described in details later, and the backlog refinement, which is not mentioned inside the theory of the Scrum methodology, but it is a lot used in practice. The iteration lasts 2 weeks inside the ApeCar team, as theoretically described by the Scrum methodology. About the Kanban methodology and how the ApeCar team uses it mixed with Scrum, obtaining the Scrumban, it will be talked about later on in the thesis work. For all the ceremonies, the team only relates to Scrum, so everything will refer to it from now on.

2.3.2 DoD - Definition of Done

The ApeCar team has a document inside a platform, explaining all the criteria to be respected in order to consider a story as done. In the following they're listed "(from [3])":

Acceptance Criteria

1. **All acceptance criteria are fulfilled:**
   Each acceptance criterion defined in the user story is fully met.

2. **Completion of all subtasks:**
   All associated subtasks (e.g., analysis, implementation, testing, pull request, merge into the development branch) are marked as completed.

3. **Documentation requirements:**
   If documentation is specified in the acceptance criteria, the relevant Confluence page must be created. The page must include a link to the documentation, video, or proof of concept (POC), and the Jira code must be present in the page title.

4. **Jira test case requirements:**
   For user stories, there must be at least one associated Jira test case.

5. **Unit testing:**
   All required unit tests must be implemented and successfully passed.

6. **Firebase tracking:**
   All implemented functionalities must be tracked in Firebase as planned with the same nomenclature for both iOS and Android.

7. **Execution of planned Jira tests:**
   All planned Jira test cases must be executed and marked as PASSED and DONE.

8. **Attachment of additional information:**
   Any supplementary information specified in the description field must be attached, if required.

9. **Resolution field completion:**
   The Resolution field in Jira must be updated with one of the following options:

   - Done
   - Fixed
   - Won't fix
   - No Longer Applicable
   - Not a Bug
   - Out of Scope
   - Cannot Reproduce

10. **Product Owner (PO) approval:**
    Final approval and feedback from the Product Owner (PO) must be obtained → the PO will do the approval once the issue on Jira (of the main ticket) is in Ready for Acceptance status.

2.3.3 Sprint planning

The sprint planning inside the ApeCar team, like in the theory of Scrum methodology, is performed at the start of the sprint, which is called iteration according to the SAFe Agile framework, because it also includes working cycles of Agile methodologies different from Scrum. During the Sprint planning all the team discusses about the user stories to be added to the Spring backlog, in order to be implemented. The team has a certain velocity and capacity which is constantly measured by the Scrum Master, so that the team is very likely to complete all the stories committed inside the Iteration, unless potential blocks due to technical problems or dependencies from other teams inside the ART.

Velocity and capacity

As mentioned above, the Scrum master computes and updates the velocity right after every sprint. Talking about the velocity of the team, it is the average speed of the completed work from the team members, and it is computed on the measures coming from the last six iterations as an average value of the measures themselves. The measure taken inside from every iteration corresponds to the story points completed during it, so, at the end of every iteration the Scrum master can compute the updated velocity, which numerically will be the average of the story points that the team is able to commit and complete in a single iteration. Given a velocity and the number of team members, once fixed the iteration length, it is possible to compute the total number of story points the team is expected to be able to complete all over an Iteration. As everyone knows, some team members may be absent some days, for example, for vacations or other kind of permits, so the actual number of story points that the team is able to complete may be lower than the one computed as previously described. As a consequence of this, the Scrum master also computes the capacity, which is the number of story points the team is expected to complete during the iteration, considering the absences of all the team members. In the end of these computations, the Scrum master gets the capacity, which can be used to choose which user stories to commit or not for the next sprint.

Iteration buffer

The iteration buffer is very important to perform effective sprint planning, for different reasons. It is possible and it regularly happens in software development that software engineers and developers face technical issues, problems related to dependencies from other teams, or incident to be managed, so practically bugs to be fixed, with a variable priority level. Due to these events which can happen with a varying probability, if the team commits inside the sprint a number of story points equal to its capacity for that iteration, it will be very unlikely to complete them all, so the remaining user stories will be moved in further iterations, also according to possible business priority changes

related to them. As a consequence, the Scrum master always keeps an iteration buffer, which is the difference between the actual capacity and the story points committed for an iteration, so that the team also has time to manage technical problems, overhead to communicate with other teams in case of issues due to dependencies, etc. The iteration buffer is also very important to guarantee the possibility of the team to react to incident, without having to leave incomplete some of the user stories already committed, so that the users are constantly provided new value, by keeping their experience at the highest possible level inside the app.

2.3.4 Scrumban

The Scrumban is a hybrid approach derived from the Scrum and Kanban methodologies in order to achieve a higher level of flexibility, inside the structure of the traditional Scrum. The ApeCar team adopts Scrumban, so that it can be able to react faster to incidents coming during the sprints, also depending on their priority. There are different incidents priorities, starting from 4, which is the lowest one, to 1, which is the highest one. The reaction depends on the priority of the incident: for example, in case of priority 2 incidents, "War rooms" are opened, in which several people with different roles, like developers, managers, product managers, etc., join and discuss about how to solve the problem, its actual impact on users, the users impacted by the problem, etc. In this scenario, the developers responsible for the code causing the problem have to work until the incident is not solved, and everyone has to help them as much as he can, in order to reduce the fixing times.

2.3.5 Sprint review

The sprint review, which consists of a demo with all the stakeholders of the product, is performed every Friday after the end of an iteration. During the review, the ApeCar team shows the new features implemented. The stakeholders for the ApeCar team and the MyPayments App are the product managers working across all the teams of the global project, especially the product manager of the MyPayments App. During the sprint review, the main developer of a certain feature usually shows a demo of it, with a video or by showing the functionality in real time on the product. After completing the demo, the presenter asks for any questions coming from the product managers, then he answers them, if there are any.

2.3.6 Sprint retrospective

After the end of each iteration, the members of the team do a retrospective, which is conducted by the Scrum master. During the retrospective, each team member can express his thoughts, by writing them on a board, and then the whole team discusses about everything inside an online meeting. The team usually also gives a mark to the iteration.

2.3.7 Backlog refinement

The backlog refinement is executed with all the team together, although some of the user stories which will be implemented in the iteration aren't related to all the team, since there are three main categories of developers, which are the ones working iOS, Android and on the backend. During the backlog refinement, anyone can express his thoughts or concerns about new user stories to be implemented, and then the developers involved in the implementation of the user stories have to give an estimation of that. As from the literature of software engineering, there are several ways to give an estimation of user stories, like Scrum poker, T-shirt sizing, which simply consists of giving an estimation with the value of the size of a t-shirt, like S, M, and so on, the three-point estimation, etc. The ApeCar team estimates the user stories using the Scrum poker, with Fibonacci values, generally up to a maximum of 13, otherwise, the story should be decomposed into more and smaller stories, manageable inside a single iteration. During this meeting, the developers can also discuss technical aspects of the implementations, in order to provide better estimations for the user stories. After each vote, it is important to get to the same values in order to confirm the estimation for the user story, so if the values proposed by the different developers are not equal, they have to explain themselves the motivation of their vote, so that the voting can be performed again, to verify the new estimation proposed. This process continues until the proposed estimations are the same among all the developers participating in the voting operation. In the end of the refinement, the team has estimated the new user stories, ready to be planned in further iterations.

2.4 Spikes

The team can commit user stories to be completed inside the iterations, regarding the implementation of new feature inside the iOS or Android app, inside the backend. Moreover, the team can also commit Spikes. A Spike is created when a user stories is requested to be implemented, but the time needed is not easily estimable, or the technical possibility to implement it is not granted, so the team has to check and verify if the user story can be implemented and in which way. Spikes usually last one day and are expected to produce a document as output, to explain what has been found in detail, any potential problems, etc. If the spike yields a good result, the developers can estimate the user story more confidently, making it ready to be added to an iteration to be implemented.

2.5 SAFe framework - ART

All teams within the ART adopt the same shared framework in order to cooperate harmoniously among themselves. The adopted framework is SAFe, which means Scaled Agile Framework.

2.5.1 Definition

SAFe is an organizational framework that allows one to apply Agile, Lean and DevOps' principles on a large scale, in a context which several teams work in, with the objective of

developing very complex products, in a very quick and continuous way, as already defined inside DevOps and Agile principles, while being aligned with the strategical objectives of the company the teams work for, all together. In order to achieve these results, the SAFe framework provides a structure, roles, practices and workflows, which allow to extend the traditional Agile application to a very wide context of working teams, by keeping the focus on the cooperation and continuous deployment, as in the DevOps principles, in order to always provide value to the customers, that are fundamental for every business. In the following a scheme containing the principal elements and functions inside the SAFe framework is represented:

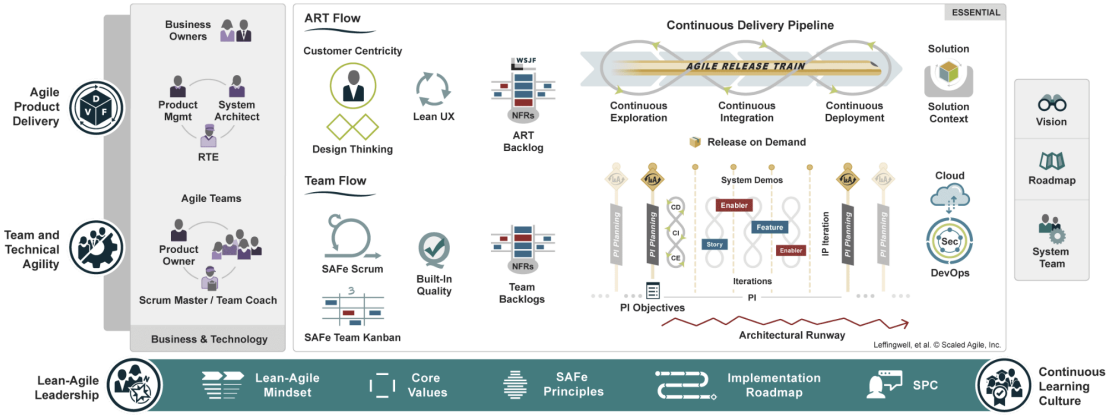

Figure 2.1. Scheme with the main elements of the SAFe framework from (Reproduced from [4])

In the following sections, the most important elements and practices of the ART from the internal point of view of the ApeCar team will be explained and discussed.

2.5.2 Core values

Like Agile, Lean, and DevOps, the SAFe framework also has its own values, which add to those of the methodologies it is defined upon. These values are as follows:

- Alignment
- Built-in quality
- Transparency
- Program execution

Alignment

SAFe framework is used to manage a variable number of teams, in order to let them cooperate in the most efficient and effective possible way. It is very important that all the teams are aligned and work toward a common goal. In this way, all the teams work

in alignment with the defined strategy and objectives defined by the business, producing the best possible results, which is fundamental for every organization.

Built-in quality

Quality is fundamental in the SAFe framework, and it has not to be an add-on to the final product. It has several aspects to be considered, like code quality, external quality of the product, and so on. The objective, as in Agile methodology, consists of working to produce only high-quality products. This can be ensured through accurate testing, which has to be automated in order to automatically execute tests when needed, ensuring no regressions after new implementations or changes to the already existing code. Moreover, practices like pair programming, continuous integration, etc. contribute to consistently maintaining quality high.

Transparency

Transparency is a core value of the SAFe framework, as well as for the Agile methodology. It is very important that all the people working to a project have the complete visibility of what is done, and the trust is fundamental from the stakeholders and product managers, towards all the development teams. Transparency makes it easier to honestly cooperate among all the teams, and inside every single team, in order to earlier identify any problem and to solve them, so that the value provided to the customer is the highest possible. Transparency can be adopted through the usage of several practices, like having visible Agile boards, so that every member of the team and also product managers can see the progresses of every single team inside the ART, sharing metrics related to each team, and having frequent feedback, which is possible thanks to iteration reviews and system demos.

Program execution

Program execution in the SAFe framework consists of continuously providing value to the customer, which is crucial for every business. This value is similar to continuous deployment in DevOps. In order to actually be adherent to program execution it is very important to follow the schedules defined about the development of new features, their releases, etc., while being focused also on any possible improvement to already existing ones.

2.5.3 Ceremonies

Since the SAFe framework is grounded in Agile principles, just as there are some ceremonies defined by the Agile methodology, there are also some within the SAFe. The main ceremonies of the SAFe framework are as follows:

- PI Planning
- Scrum of Scrums (SoS)

- PO Sync
- System Demo
- Inspect & Adapt (I&A)

PI Planning

PI planning is executed at the start of every PI, which means Program Increment. The ART the ApeCar team works in adopts a PI lasting five iterations, each one lasting two weeks. There is only an exception to this structure and organization, which happens only in a PI over the year, and it is that in the PI going over the month of August, the iteration over the second and third August weeks, lasts three weeks instead of two. This choice is due to a lot of people taking vacations in these weeks, so the capacity of the teams is too low to complete the committed user stories for the iteration within two weeks, as it usually happens. Product managers, product owners, system architects, RTEs, which means Release Train Engineer, business owners, and other technical or functional stakeholders, together with all the members of every team inside the ART take part to this ceremony, in order to schedule the work to do in the iterations of the PI, manage dependencies across the different teams, to better plan the activities and the user stories to be committed inside every iteration, otherwise, some teams may commit user stories which are impossible to complete due to some dependencies to user stories of other teams, which may not be completed yet. During PI planning, every team maps all dependencies with others, identifies the main risks related to the activities and user stories to be implemented, and defines the objectives of the whole PI. PI planning typically lasts two days, as in the ApeCar team case inside the ART.

Scrum of Scrums (SoS)

The Scrum of Scrums, also called SoS, is a Scrum meeting that is regularly performed among the scrum masters of all the teams inside the ART. Its objective is to coordinate all the teams inside the ART, to manage and solve problems and dependencies across the teams. Thanks to the SoS, the workflow of the entire ART improves in terms of efficiency, and all the scrum masters can align on the common objectives of the current PI. The participants to this ceremony usually are the scrum masters of every team inside the ART, but also the RTE and sometimes product managers or system architects, if they are needed to provide any kind of explanation to the scrum master, allowing them to better organize the work of all the teams.

PO Sync

The PO Sync, which means Product Owner Sync, is a very important ceremony inside the SAFe framework, because it is fundamental in order to coordinate the products owners from every single team, and also product managers in several situations, in order to guarantee continuous alignment on priorities of the different features committed during inside the PI and how their development progresses trhoughout it. Moreover, some teams

may have related user stories in their backlog, so the product owners can also discuss how to better synchronize their teams' work during the iterations. Product owners during this meeting also discuss functional and prioritization problems, if there are any. The PO Sync is regularly executed, like a Scrum of Scrums, and sometimes also system architects, the RTE, and the business owner take part in it. Thanks to this sync, POs are always aligned about the strategic business direction of the product, while being also able to quickly adapt to problems happening over the PI and feedback received from product managers, etc.

System Demo

The system demo is a ceremony involving all the people already involved inside the PI planning, with the objectives to show the results of the PI obtained by every single team inside the ART. During this demo, every team usually shows the most important features implemented during the PI, in order to receive feedback from the stakeholders, the product managers, etc. The system demo also allows one to verify that the ART is actually producing the expected value for the customers, according to the business strategy already defined.

Inspect & Adapt (I&A)

This ceremony is equivalent to the iteration retrospective at the team level, but it is conducted among all the teams of the ART, together with the RTE, product managers, system architects, business owners, and stakeholders. During this ceremony, the system demo is executed in order to show the results obtained and the features implemented. After that, the metrics chosen at the ART level are analyzed together to evaluate the positivity of the results. Some metrics which may be used in order to execute this analysis are the percentage of PI objectives completed, velocity, quality, etc. In the end, the participants cooperate with the objective of identifying the main problems that appeared or emerged throughout the entire PI, so that they can define the cause of the problems, without having to point the finger on anyone who has any responsibility, but only to improve the performance inside the ART, across all the teams. The technique used to understand the causes of the problems is the RCA, which means Root Cause Analysis. After having identified all the problems and their causes, the participants identify improvement actions to prevent the problem from appearing in future Program Increments.

2.5.4 Epics

Within the ART, across all the teams, it would be too complex and also meaningless to discuss user stories, as each team does this at a lower level. Due to this problem and to the need of being more concrete and adherent to the business strategy point of view, it is necessary to talk about features at a higher level, without making too detailed distinctions among the different platforms on which the product is deployed, etc. An equivalent of user stories at the team level is needed at ART level to achieve this objective, which is why there are epics. Epics are similar to user stories, but may include several implementations

and sub-features which need to be implemented, in order to complete the entire epic, so it is also possible to define it as a collection of user stories, related among themselves from a common objective, which is defined by the business. Every epic has a description of what the business wants to provide to the customer, and then the PO has to align the team about what is needed to be done from his team's side, in order to contribute in the correct way to the epic actual implementation. As imaginable, starting from the analogy of the epics with user stories, there is also a backlog at ART level, so that it is possible to track and prioritize the features across all the teams, in order to let them be able to coordinate and cooperate in the most efficient and effective way, if needed.

2.6 Considered technologies

During this work, several technologies are considered, each one of them for different and specific reasons, according to the objectives to be reached at the end of the work.

In this section, these technologies will be briefly described, in order to provide a better overview of the entire technological stack.

2.6.1 Python

Description

Python is a high-level programming language that is interpreted and object-oriented. It is also a very easy to read programming language.

Thanks to these characteristics, it is well-suited for data analysis and machine learning. It has a very extended set of available libraries and frameworks, such as NumPy and Pandas, providing tools to manipulate data in several ways, to implement machine learning algorithm, and so on.

This is one of the most used languages in these fields, so it is supported by a very active and extensive community.

There are also several studies and articles highlighting its relevance in machine learning and AI field, such as "Machine Learning in Python: Main Developments and Technology Trends in Data Science, Machine Learning, and Artificial Intelligence" from [14]

2.6.2 XGBoost model

Description

XGBoost is a very powerful machine learning algorithm, due to its efficiency and performance with limited resource availability.

This algorithm is based on gradient boosting, optimized for speed and performance.

XGBoost is very effective in classification and regression problems, since it is highly capable at managing heterogeneous data, also because it is very robust, and it rarely happens that overfitting occurs when using it. This model also manages missing or incomplete data, which is frequent in real-world data because of errors or other issues which may occur.

There are several studies that demonstrate the effectiveness of this machine learning model in different contexts, as well as its integration with the scikit-learn library, as demonstrated in the study "Gradient Boosted Regression Trees in scikit-learn" from [13]

Reasons to use it

There is a very important article, named "XGBoost: A Scalable Tree Boosting System" from [6], which highlights the performance of the model, describing it as better and more efficient rather than other models, on several machine learning benchmarks. This is very important for this thesis work, for the reasons analyzed later on.

Thanks to this information, it is easy to understand why it makes sense to use this machine learning model, due to its characteristics, but also its performance, which is satisfying for several potential applications.

2.6.3 scikit-learn library

Description

This is a machine learning library for Python language. It provides a very wide set of tools for different purposes, such as data mining, statistical modeling, etc.

This library is very easy to use, as well as scalable and efficient in its possible applications.

scikit-learn library supports several machine learning models, which can be trained and executed right after the training operation is done.

It is important to note that it is backed by a very active community, which continuously contribute to its development over time.

Usages and algorithms supported

Going into further details, it includes support for supervised learning algorithms, both for classification and regression purposes. Some examples of these algorithms are Support Vector Machines (SVM), k-Nearest Neighbors (k-NN), Random Forest, etc., for classification purposes, but there are also Linear Regression, Support Vector Regression (SVR) and others as regression algorithms, as highlighted and explained in the book "Mastering Machine Learning with scikit-learn" from [7].

This library also provides features to train machine learning models without supervision, like clustering algorithms, but it also supports dimensionality reduction algorithms, like PCA, which means Principal Component Analysis, that are fundamental to solve different kind of problems. The clustering algorithms supported are k-Means, DBSCAN and Hierarchical Clustering.

Moreover, additional support is included, for example, to preprocess data, by providing tools to normalize, standardize, and also transform data with techniques like the One-Hot Encoding and Polynomial Features. These techniques are widely used in data preprocessing to prepare the dataset in the training phase of a machine learning model.

Scikit-learn also provides features to evaluate trained models with an extensive set of metrics, which can also be configured depending on the needs. The available metrics

include accuracy, precision, f1-score and ROC-AUC, also with different variants available for the f1-score. Cross-validation is also supported to evaluate the machine learning models more robustly, reducing the likelihood of bias.

This library is also widely used because it is integrated with other Python libraries like NumPy, SciPy, etc. Thanks to these integrations, it is easy to implement in any scenario, providing useful tools for several needs.

Reasons to use it

scikit-learn is also widely used because of the tools it provides for training machine learning models and preprocessing data for the training. There are several articles highlighting the completeness of the tools provided by this library for data preprocessing, specifically for normalizing and transforming data, often surpassing other libraries providing similar functionalities in terms of efficiency. Moreover, another reason to use is its seamless integration with other libraries, which enhances the entire project workflow, as supported by a study entitled "Scikit-learn: Machine learning in Python" from [12].

2.6.4 RabbitMQ

Description

RabbitMQ is an open-source message broker that uses the AMQP protocol, which stands for Advanced Message Queuing Protocol. It works with an asynchronous paradigm in order to let the different applications to exchange messages through message queues, in a reliable way.

This broker supports different messaging mechanisms, such as the publish/subscribe one, where there are topics and subtopics. Applications can subscribe to specific topics to read or publish messages, sending messages to other applications that are subscribed to the same topic. It is fundamental to note that the messages stay in their queue until they are consumed by a subscriber. There are also other mechanisms supported, like the routing and the request/response ones.

The request/response paradigm is used to facilitate bidirectional communication between application instances.

Reasons to use it

RabbitMQ is acknowledged for its scalability, reliability, and simplicity of integration with different programming languages.

It is important to note that it supports clustering to balance the workload and automatically manage failures, as highlighted by the study "Evaluation of highly available and fault-tolerant middleware clustered architectures using RabbitMQ" from [15]. These characteristics are very important for certain implementations that need to be scaled, although they may not be considered when there is a single instance consuming messages inside a system.

Moreover, considering other alternative technologies in the same category, like Kafka, it is important to notice that RabbitMQ is very powerful at managing clusters in a

reliable way, at replicating queues among the different instances, which is very important to obtain the expected results from the overall system, and it is also easy to integrate among applications using different programming languages, as also stated by another study entitled "A study on Modern Messaging Systems: Kafka, RabbitMQ and NATS Streaming" from [16]. This last analysis also details RabbitMQ's capability to maintain very low latency while supporting different communication protocols, which is crucial in terms of flexibility for this technology.

2.6.5 Apache Kafka

Description

Kafka is an open-source platform providing tools to manage high volumes of data flowing in real time. It is designed to be scalable, in order to reliably handling ever-growing data volumes.

It is also known for its capability to effectively manage persistent data and operate with low latencies, as described in the study "Kafka: a Distributed Messaging System for Log Processing" from [11]. Due to these characteristics, Kafka is well-suited for real-time processing in several contexts.

Moreover, Kafka can easily be integrated with different systems used to process data, such as Apache Hadoop, Apache Spark, and so on.

Reasons to use it

There are several reasons to use Kafka, such as the very good capability to work with persistent data, while maintaining low latencies, which are crucial in order to execute real-time processing.

It is also important to know that it is scalable at a very high level, specifically in an horizontal direction, so it may be the perfect choice if the system being designed or redesigned is planned to grow to manage a very high data volume, always keeping throughput very high respect to other similar technologies with the same data amount. These results have been obtained using Kafka to execute an experimental implementation, which is documented in a study named "A Study of Apache Kafka in Big Data Stream Processing" from [9].

2.6.6 REST APIs

Description

This approach is based on the HTTP protocol. It consists of a specific way of implementing APIs, according to the REST paradigm, which means REpresentational State Transfer.

REST APIs have different characteristics: such as the stateless nature, meaning that every request is independent from every other one, so it needs to include all the needed information to execute the procedure on the server side.

Moreover, this API flavor is scalable, thanks to its architecture, allowing it to handle a high rate of requests.

It is very important to know that REST APIs expose a uniform interface for the possible operation which can be called, so that they are compatible with other systems of different natures.

Reasons to use it

REST APIs are widely used due to the ease of implementation they provide, keeping a very high flexibility in terms of applications, but this approach fits best web solutions, thanks to the improved scalability and efficiency for them, as also highlighted by the study "Choosing the Right Communication Protocol for your Web Application" from [8], which also states that sometimes, despite limited performance in certain scenarios, REST APIs are still the chosen solution for web applications due to their simplicity of integration and interoperability with other systems they have.

Although REST APIs are very flexible and scalable, when implementing microservices with high workloads or real-time systems, technologies like RabbitMQ manage the workload volume better due to its asynchronous nature and the publish/subscribe paradigm, as demonstrated by the study "Performance Analysis of RESTful API and RabbitMQ for Microservice Web Application" from [10].

Chapter 3

Case study

In this chapter, the existing iOS and Android application which the ApeCar team works on is described, with technical details on its architecture and on both versions of the mobile applications. Moreover, some problems of the project are introduced, in order to effectively motivate the solution proposed with its implementation.

3.1 MyPayments App

MyPayments is an application available for both iOS and Android devices, dedicated to the markets of these countries: Germany, Switzerland, Austria, Denmark, Sweden, Norway, Finland, Latvia, Lithuania, Estonia, France, Faroe Islands, Iceland, and Liechtenstein.

The application has been designed to streamline the management of digital payments for merchants of all kinds, by providing them several features which make it easier to manage their terminals, to enable tips, for example, to receive assistance in various situations, and more. Users are also able to access a detailed transaction history, with curious insights computed on their data, and users can also accept NFC card payments directly from their smartphones, working as a Soft PoS, without the need for any external terminal. The application has some different features specific for every country, because some of them haven't still be released in some countries or aren't available at all, due to several reasons.

As previously mentioned, MyPayments application is available for both iOS and Android, providing the same functionalities but in different ways in some cases, because of the characteristics of operating systems and already existing implementations in both versions of the app.

The application is integrated with a very extended backend system to retrieve transactional data, to authenticate and get authorized to access different other systems, to access detailed reports and analytics about transactions of every merchant, etc.

In the following, there are some screenshots of the user interfaces of the application:

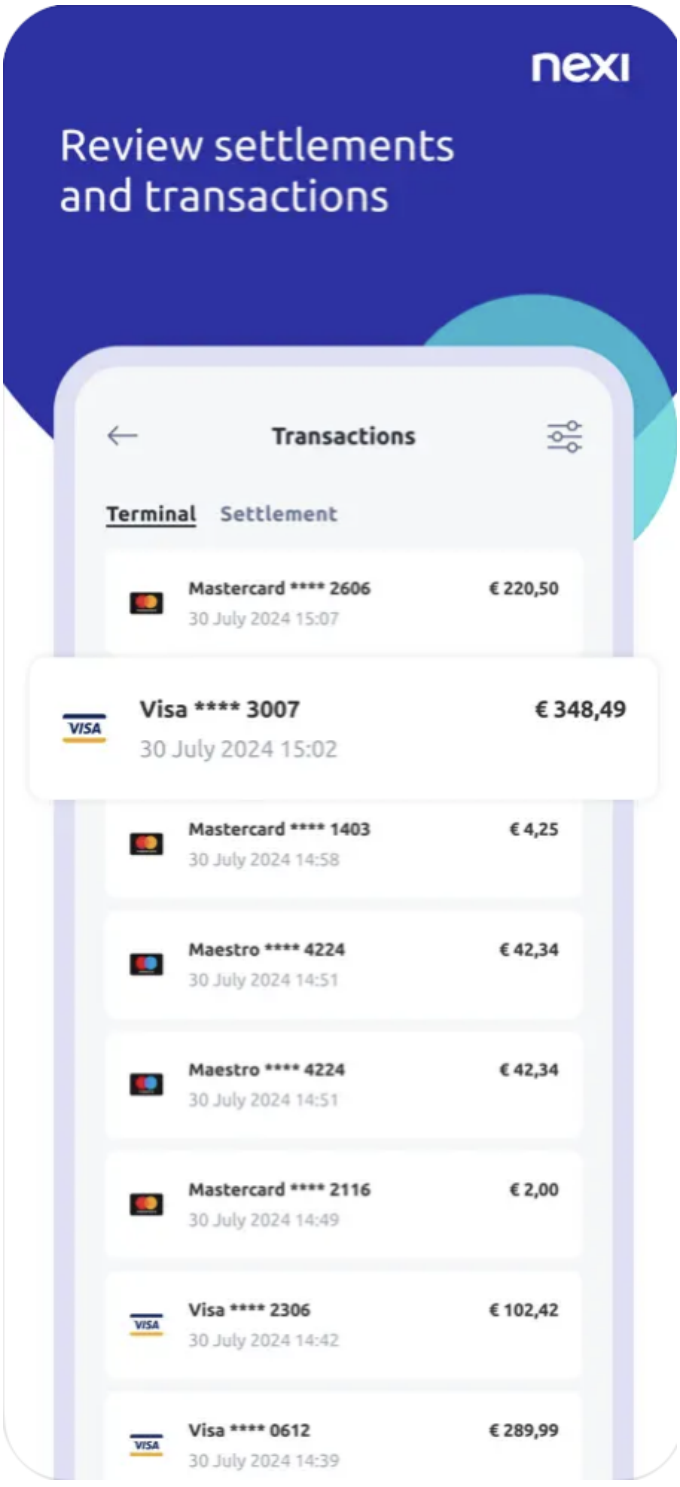

Figure 3.1. Transactions and settlements interface (Reproduced from [5])

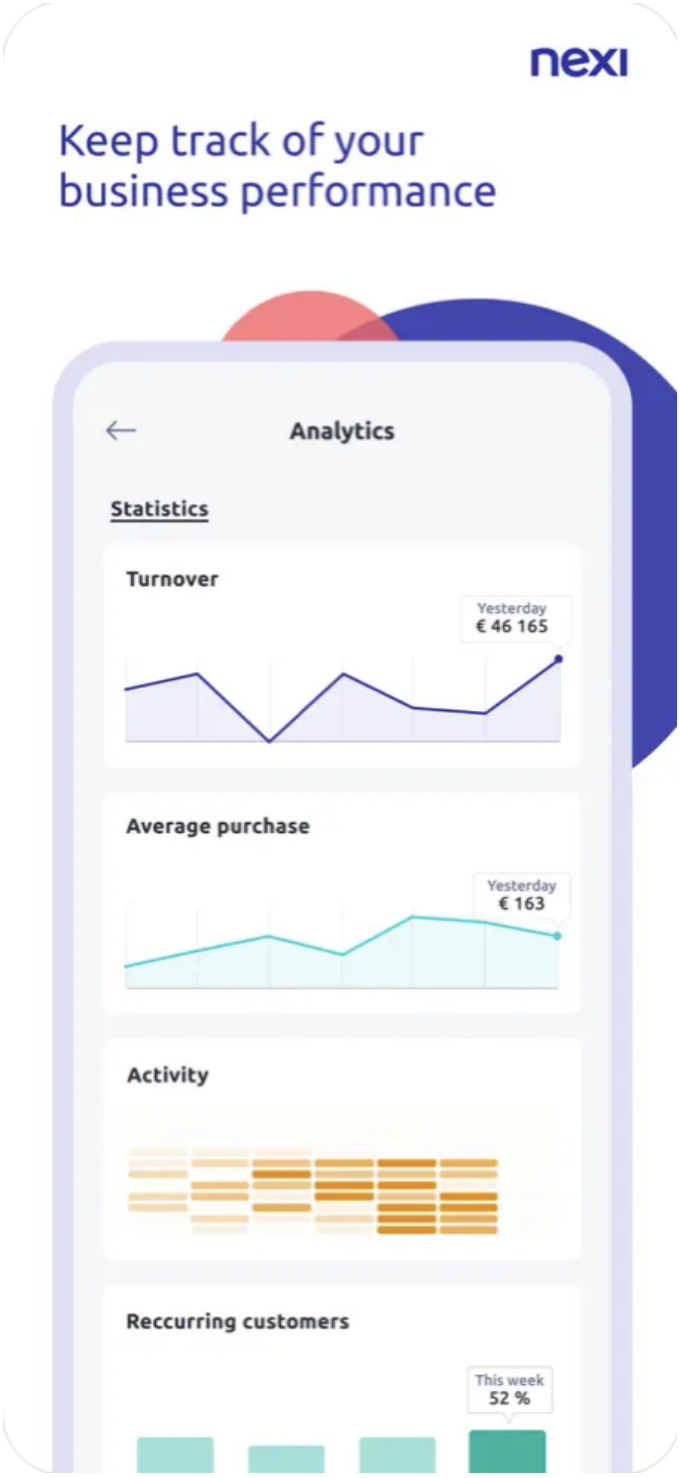

Figure 3.2. Business statistics interface (Reproduced from [2])

3.1.1 PoS categories

Different kinds of PoS terminal exist, and the MyPayments application manages them in different ways, from the physical PoS, to the virtual PoS. Before going into details, let's briefly describe the characteristics of every terminal category and how they're integrated into the application.

Physical PoS

A physical PoS is a device composed of a screen, a keyboard, which may be physical or implemented using a touchscreen, potentially also a printer, which is used to print receipts to be given to customers, and also a reader for cards. There are different kinds of readers, like the magnetic stripe reader for cards, the EMV chip reader and the NFC reader, which means Near Field Communication and it is compatible with almost every smartphone on the market nowadays.

Soft PoS

A Soft PoS, meaning Software PoS, consists in using a physical device like a smartphone, a tablet, or any other of their kind, which is able to perform digital payments on itself, without the need of any additional physical hardware. Merchants only need their device to perform contactless payments, using the NFC technology in order to retrieve all the information from the cards used to pay for the receipts.

Virtual PoS

A virtual PoS is a software platform, like a web one, which is able to elaborate payments, without the usage of any physical device. Virtual PoS platforms are used on the web to execute payments remotely, just by securely sending the card information data needed to process the payment. Virtual PoS can be integrated into various applications, supported by any platform, thanks to integration with the web service of the virtual PoS.

3.1.2 Security

The application follows the rules defined by PCI-DSS, which means Payment Card Industry - Data Security Standard, about the protection and security of all the data managed by the application working in the payments market. Moreover, the My Payments application also implements end-to-end encryption in any communication done with the servers, thanks to the TLS protocol used over HTTP.

3.1.3 Translations

MyPayments application is translated into eleven languages to provide the best user experience to any merchant in any country where the app is released. Managing translations can be done in several ways in apps, for example in iOS by using Localizable.string files, so there is a file for every language we translate phrases and words into, but this method

has not been used for this project. The methodology used to manage translations involves using Lokalise, an online tool allowing to manage translations to any language, in a centralized way for every version of the application, so for both iOS and Android, by accessing the translations through its dedicated SDK, directly within the applications. Lokalise associates a translation with a specific key, which has to be unique and can be named in any way. In the MyPayments project there is a standard for naming new keys in Lokalise, which consists in inserting the string "ms" before any key name, in order to recognize that the corresponding translation is related to the MyPayments applications. As already mentioned, both applications access translations thanks to the Lokalise SDK, by retrieving the texts in the locale language of the device running the application. In order to place the translations on the right label, a Lokalise key is associated with every single component, so that the translation retrieved from the Lokalise system will automatically be set in the correct position. In some situations, the locale language of the device, which is used to request translations, has not any mapping on Lokalise, so the default language, which corresponds to English in this case, will be used.

3.2 Overall architecture

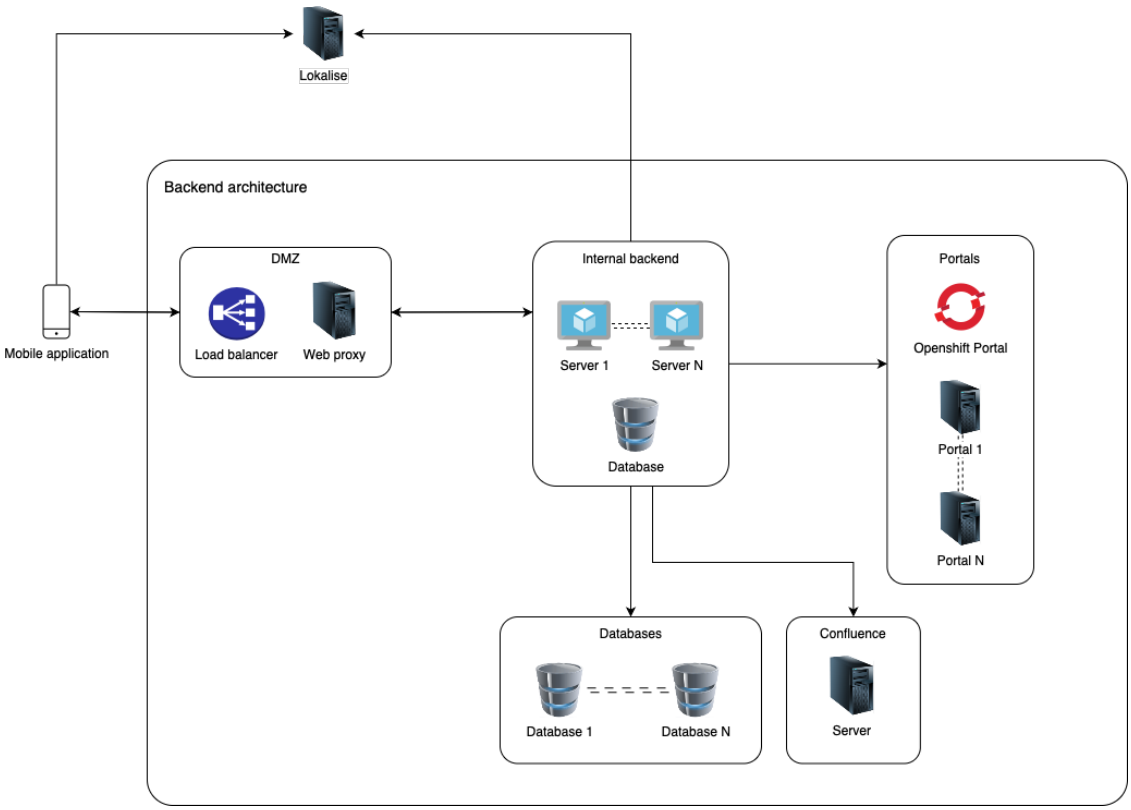

Figure 3.3. Overall backend architecture (Reproduced from [1])

The overall architecture of the project's backend is composed of several elements, which will be simplified to their functions within the architecture, without going into deep details, as requested by the company. Let's start from the services which are directly managed by the ApeCar team. These services have multiple instances, to achieve better performance in providing the best possible service to the users of the MyPayments application. They are inside the central block of the scheme, and will be discussed in detail later on in a dedicated section, because of their importance for the MyPayments application.

At the left of the central block, there is a DMZ, which contains a load balancer, to better distribute the incoming requests for the internal servers of the central block, and a web proxy, which can be used to safely exit outside the central block. The security configuration of the web proxy isn't known by the ApeCar team, because it is managed by a different team, working on the infrastructure. The load balancer configuration is also unknown to the ApeCar team for the same reasons.

At the right of the scheme there are several servers, exposing services in order to access the several payments portal for the merchants. Since the MyPayments app allows merchants from different portals to access the application, its services then retrieve data directly from the portal on which the different merchants are registered, in order to show it in the application. The backend of the MyPayments app also requests some operations to be performed on these portals, by exposing APIs to the app, which can request to execute these operations at any time.

These portals are related to the merchants of different countries, so there is a portal for German merchants, for example, another one for Nordic merchants, and so on. There also are some portals related to the same merchants, with different versions, because they're being migrated, but the previous version cannot be disabled until the migration isn't completed, so the MyPayments app also manages merchants part of a migration, by accessing their data in the correct way, with specific strategies, according to the requirements, business rules, etc.

Since some data is present on Confluence, about the ApeCar team and also other teams inside the ART, the central block is also connected to the confluence server, in order to be able to directly upload automatically generated reports, etc.

At the bottom of the scheme there is a database block, containing some of the information accessed by the MyPayments app, which has not access all the existing information on these databases. This block isn't described in much detail, for several reasons.

The central block also communicates with the Lokalise servers, in order to implement the translation functionality described in the MyPayments application section.

In the end, the mobile application instances of the MyPayments app communicate with the central block, through the DMZ, to implement security controls and balance to load among all instances inside the backend managed by the ApeCar team. The mobile application instances also directly communicate with external systems not present in this scheme, because they do not interact with the central block, so they are not displayed here.

Talking about security, all the communications among the different server instances use https as a protocol, in order to achieve confidentiality, integrity, and authentication.

3.3 Internal backend architecture

The MyPayments App backend is composed of a limited set of servers with related services, essentially two, plus an additional external service managed by the ApeCar team only to perform maintenance and incident management on it. This backend system is completely developed with Java Spring Boot framework, and there are two environments as is usually done according to development best practices, which are the pre-production environment and the production environment. The pre-production environment is used to allow developers and stakeholders to perform tests on new feature or on already existing ones, in order to verify the presence of bugs, issues, etc. The production environment only executes the tested and released code, which has already been approved by at least one system architect of the ART.

3.3.1 External service

The External service is the outer one managed by the ApeCar team, and it communicates with a data warehouse, containing all the data about transactions of the user on the different systems. It exposes several APIs providing statistics and detailed information coming from the analysis of the very large amount of data stored in the data warehouse. The ApeCar team isn't responsible of any development of this service, so hasn't many details about how it works, technical implementations, specific business requirements, etc. The backend of the MyPayments app communicates with this service to retrieve statistics and other data associated with app users, as well as general data which is available to any user of the application.

3.3.2 Internal services

The two internal services, which will be called BE1 and BE2, are managed and developed by the ApeCar team, which also manages all the incidents involving them. These two services have different roles inside the backend of the MyPayments app.

BE1 is the frontend service, which takes all of the requests from the mobile applications, and then it directly communicates with the services connected in the entire architecture for the majority of the requests, or with BE2 for the remaining ones, which are related to specific use cases. As imaginable, BE1 exposes a lot of APIs to the frontend applications.

BE2 has a lighter implementation, because it only provides some specific functionalities, which are the registration, authentication, and authorization of the users trying to access the MyPayments application, but it also provides other business-analysis-related functions. These last functions are about tracking all the accesses of the users from every country, in order to create country-focused report for logins, and also overall reports, considering combined data from all the countries where the app is released. Since there are more instances of these servers, both in the pre-production and in the production environment, distributed lock mechanisms have been implemented to perform some operations in the correct way, and only once, avoiding repetitions for different reasons, like having duplicated data, and so on. Moreover, BE2 also provides other tracking functionalities,

which will not be discussed in this thesis work.

3.3.3 Profile configurations

All these server instances operate in two different environments, as previously mentioned. Also the servers providing external services to the ones managed by the ApeCar team have different URIs to be identified. Due to this architecture, all the pre-production instances and production instances, communicate with resources in the web, which have different URIs, depending on the environment they are into. For example, in the pre-production environment the BE1 service communicates with a service called SRV-PP to perform this explanation, which has a specific URI in the web. The same BE1 implementation inside the production environment communicates with SRV-P, which is the production replica of SRV-PP, in order to provide the correct service to the frontend mobile applications, for example by accessing the production database, and so on. A possible solution to implement this mechanism in BE1, BE2, and external server instances may be to have two different builds of the service code, with different URIs for the services they communicate with, according to the environment they're released into. This solution isn't the best one, because some mistakes may be done and also because for every service different builds would be needed. In order to solve this problem in an efficient way, BE1, BE2 and External server instances have profile configurations, containing all the information related to the environment they run in.

The actual implementation consists of having different ".yml" files with properties containing the URIs of the other services in the corresponding environment, and also other configuration values used by the server instances at runtime. The properties files have an ordered hierarchy to be accessed until a certain property hasn't been found, so that the property requested by the server instances is at first searched in the most specific properties file, which is configured in the execution configuration of the machine the code runs on. If the requested property is not present inside the specified properties file, then it is searched in the previous one of the ordered hierarchy; this recursively happens until the property gets found.

The main properties file configured are three, which are the "p" properties file, where "p" stands for production, the "pp" properties file, where "pp" stands for pre-production, and simple properties file, which is the lastly accessed properties file, if the requested property is absent in all the other ones. This last file usually contains common configuration properties, which are both used inside the pre-production and production environments.

3.3.4 Identity Management

As previously briefly explained, the MyPayments app allows merchants from different portals to access their data through the app. The app backend does not know the credential for all the merchants, in every portal, since it only retrieves data from them, to show it in the frontend application, after being requested. A strategy is still necessary to authenticate and authorize merchants requesting some data, which are retrieved by the various portals.

This property is satisfied with the following implementation: every merchant has to

register initially, so that the MyPayments backend, particularly BE2 service, checks if the user is already registered on at least one of the available portals; if this condition is verified, then the registration is successfully completed, and the MyPayments backend can authenticate and authorize the merchants at every login future within the app.

When retrieving data, the BE1 and BE2 services have to communicate with specific portals, depending on which one the merchant performing the request belongs to. In order to perform these requests, BE1 and BE2 are authorized to make any request for any value to any portal, to have no blocks in accessing any data. It is very important to properly manage the privacy aspect in compliance with GDPR law, otherwise. some merchants may see data from others, actually violating the privacy law in EU, which is very strict and severe.

It is also important to say that the business logic complexity of the backend services, BE1, BE2 and the mobile applications is increased because a single merchant may be present on one or more portals, but every portal has different data and also features, which have to be correctly displayed in the mobile applications. As a consequence, the application appears and behaves differently according to the kind of merchant logged in and to the requested operations.

In the following paragraphs, some technical aspects of identity management will be discussed, adding more details in order to explain the complete flow implementing the mechanism just described at a higher level.

3.3.5 Merchant login mechanism

Let's discuss the complete mechanism implemented to provide authorization and authentication to merchants on the MyPayments application. The three main operation needed to register for the MyPayments application and to be authorized for any access: registration, token and authentication, which in a single operation actually implements both authentication and authorization.

Registration

At the beginning, right after the app download, the merchant has to register on the MyPayments app through the registration process. In order to perform the registration, the merchant has to provide his/her email address or CVR, which have to be already registered on at least one of the portal connected to the MyPayments internal backend, otherwise, the registration process will fail. CVR is the equivalent of the VAT number for Denmark. Moreover, an application identifier is needed, in order to associate a single email or CVR with a specific mobile application instance. After the mobile application sends the requested information, the BE1 service receives the request and forwards it to the BE2 service, which communicates with different databases and portals to verify that the email address or the CVR present in the request is already registered on at least one source system.

In the end of this check, if the value is absent in every source system, the registration operation will fail, returning a 404 error, indicating that the requested email address or

CVR was not found; otherwise, the BE2 service will generate an OTP and send it by email to the email address in the request.

If the value in the request is a CVR, the mechanism is different, because the OTP is sent to the email address of the administrator associated with the sent CVR. In this way, the administrator can authorize people working in his organization to access the MyPayments application, although they have no access to the organization email, just by communicating them the OTP.

Right after these operations, the merchant has to insert the OTP in the mobile application and send the request to perform the next operation of the flow. The registration allows to bind an application identifier to a specific merchant or CVR on the portals. Thanks to this implementation, it is also possible to have multiple instances of mobile applications accessing the same merchant information from any of the portals.

Token

The next operation of the flow concerns the obtaining of a token. Right after the registration, the next operation to perform consists of sending the OTP code, together with the application identifier and the email address or CVR used to perform the registration.

Right after that, the BE1 service will receive the request and forward it again to the BE2 one, which implements the identity management mechanism, which will verify the OTP against the saved one for the corresponding email address or CVR and application identifier. If the check successfully ends, the BE2 service will return a persistent token to the mobile application instance, which will permanently save it for future accesses. The persistent token has an expiration period, to ensure a higher security level to the app users. If the check fails, the BE2 service responds indicating that the OTP is wrong, so the merchant can try to send the OTP again, in order to get the persistent token.

Authentication and authorization

The last operation of the login flow is authentication and authorization. This operation uses the values obtained from the previous ones, particularly the persistent token and the application identifier.

The mobile application sends these values to the BE1 service, which forwards the request to the BE2 service. This service authenticates the user if the persistent token and the application identifier are bound in the database; otherwise, it responds with an error. If the authentication is successfully executed, the BE2 service generates a JWT token, which is used to authenticate the merchant in every request of the session, until it expires. The duration of this session token has to be low, in order to increase the level of security, protecting from any potential MITM (Man-in-the-middle) attack.

After generating the JWT token, the BE2 service responds including it in the body and also collects other information related to the authenticated merchant. This implementation is not standard or structured as it should according to the software development best practices and theoretics, because together with the token, the BE2 service also returns all the information about the organization of the merchant, so all the shops inside the organization, the enabled features for the specific merchant and portal, etc. This

information should be returned by independent APIs to keep the corresponding methods inside the code independent, to improve maintainability, reduce network load, and also increase modularity. This implementation was developed by the previous external team, before the project began being managed internally by the company.

In the end, the BE2 service sends the response to the BE1 service, which forwards it to the mobile application instance, and the login will be completed, with the merchant having access to all his/her information inside the app.

After a certain amount of time, the JWT token expires for security reasons, so any new request to APIs requesting authentication will return an error, indicating to perform the authentication operation again, in order to get a new JWT token, to start a new session.

A problem in this implementation is that the authentication operation may take a high number of seconds to be completed, so performing it more than once in a short time period is neither scalable nor efficient, and the overall user experience in the application does not benefit from this mechanism.

For these reasons, the BE1 server instances have a cache storing the authentication responses for a certain period, associated to the corresponding application identifier. Thanks to this implementation, the mobile application instances can call the authentication API multiple times, without having to use all the resources needed as in the first call. This is also possible because data is unlikely to change in a short time, so the user experience gets improved, and the data consistency is not affected. Since data is unlikely to change in a short amount of time, there is no need to retrieve the same copy of it multiple times from the merchant's portals, generating higher network loads, resource usage, and so on.

3.3.6 Databases

All data displayed within the MyPayments mobile application comes from several source systems.

Among all the databases accessed to retrieve the requested information or to provide the correct service to the merchants, there are some only used to retrieve and update statistics, while others are accessed to check if merchants are already registered or not on some portals. For example, for the portal which is being migrated at the moment, there are both the new implementation and the old one in the current production environment, with the related databases.

The internal backend, managed by the ApeCar team, communicates with different databases for different purposes. For example, the External server accesses statistical data from a separate database which is used for different purposes, while the team also manages two databases, containing all the information directly related to the mobile application.

Almost all the databases use Oracle Extended SQL as query language, which means that they are Oracle databases. There are also some instances of Cassandra databases used for a specific purpose, but are planned to be dismissed in the future.

It is also very important to retrieve data properly from the different databases; otherwise, there may be inconsistencies or missing data, which would affect the user experience

of merchants impacted by the issue.

3.3.7 Known technical issues

As described up to this section, the application is extensive and manages a lot of data, so there may be issues or bugs in different implementations, in the backend services and in the frontend applications. The objective of the ApeCar team is to improve app quality to provide the best possible user experience to merchants, and this can be achieved in several ways.

Regarding the backend and the exposed APIs, the team has already noticed different issues due to external providers or potential internal bugs, which may be solved more quickly by monitoring them by constantly analyzing logs and receiving an alert when issues occur. Some known issues are related to an absence of service or to malfunctions, distinguishable by problems related to the response time, that may be too low or too high for a specific API, or implementations not working as expected, detectable when an API returns a 400 status code, for example, instead of a success status code under normal conditions, etc.

In order to improve the quality of the mobile application, the proposed solution will be discussed in the next chapter, giving further details about the overall problems and the actual implementation.

Chapter 4

Methodology

The objective of this thesis work is to notify the ApeCar team of issues as quickly as possible, so that it is possible to react in the shortest possible time to prevent or minimize the impact on merchants.

4.1 Solution analysis and choice

In this section the possible solutions to the problem will be analyzed, with their characteristics, downsides, etc., in order to choose which of them to actually implement to satisfy the objective of the ApeCar team regarding the MyPayments application.

4.1.1 Possible solutions

In order to react in the shortest possible time, it is needed to analyze logs in real-time, so that a problem is detected right after happening, and an alert can be sent by the system analyzing logs.

There are several possible solutions to analyze logs, like implementing some detailed controls within the code, in the API methods. With this approach, if a problem occurs, the corresponding backend server instance can directly send an alert, by using the email or any other method.

This solution requires significant development effort to be implemented, and it also increases the overall complexity of the backend code, because additional logic is added to every API called.

It is also very important to consider that every API and method may present issues under different conditions, also due to the implementations from external services and portals with which the API method interacts, making this solution very onerous to implement.

Moreover, there are other possible solutions, like developing a script which can be run on demand, or by scheduling a chronologic job. This solution would need to receive log files as input to analyze them, but the effort required to define all the possible issues for every API in a deterministic way would be too high, and there would not be any warranty

of covering every possible problem, so also this solution has been discarded, as well as the previous one.

Another possibility is to use a machine learning algorithm. For this solution, it is necessary to choose one of the available ones to train it and improve its performance, maximizing recall and accuracy.

This choice allows to train the model basing on the detailed logs collected from the internal backend servers and let the model autonomously learn from them, analyzing and considering all the situations actually happening at the moment, so no issues will be ignored or mistakenly not considered. Furthermore, this model can be run continuously to receive logs and analyze them in real-time, so that an alert is generated and sent in the chosen form right after a problem occurs.

In the end of this analysis, the choice was to use the machine learning model, because it promises more accurate and comprehensive results, characteristics which align with the objective of the ApeCar team for the MyPayments application.

4.1.2 Model potential choices and evaluation

Before analyzing the potential choices and evaluating them, it is important to define the resources available to train the model, also according to the amount and complexity of data in the log files of the backend service considered.

Talking about the available resources, the chosen model will be trained on a portable PC, so it would not be sustainable and feasible in a reasonable amount of time to train a large-scale model or even a medium-scale one, but in this case it depends on the model characteristics.

In this analysis, several potential models will be briefly described to determine whether they are suitable for solving the previously described problem.

XGBoost

This model is a Gradient Boosted Decision Trees algorithm, that does not support categoric values, but it is possible to define them by the usage of the one-hot-encoding technique. The computational cost of this model is medium, but the model is very optimized and in between a lightweight and a medium-scale model. The ideal number of features is maximum of two thousands, but the model works better on tabular data, which are the data the ApeCar team can generate directly from logs.

LightGBM

LightGBM is another Gradient Boosted Decision Trees algorithm, but it also uses leaf-wise growth, rather than the level-wise approach used by XGBoost. This model performs better on large datasets with more than a thousand features, natively supports categorical data, and is highly optimized in terms of computational cost required for training. Moreover, LightGBM is not very robust on a small number of features, also because the robustness is affected by the tuning executed. It is also more sensitive to overfitting if not properly tuned, which requires additional effort.

CatBoost

CatBoost is another Gradient Boosted Decision Trees algorithm, which works very well with data containing a high number of categorical features, without needing one-hot encoding to use this kind of features. The computational cost is medium, but higher and less optimizable compared to XGBoost, and this model works well with any number of features.

This model is quite expensive to execute with a large set of data.

Random Forest

Random Forest is a bagging of decision trees machine learning model, which is a Random Forest algorithm variant, with a very low computational cost to train and execute the model, but it is less scalable than boosting models.

Moreover, it performs very well with a few hundred features inside the data set, but it has poorer performance compared to other boosting machine learning models. It does not require much effort for tuning, so the training time is slower than other models, but the performance is affected by this characteristic.

4.1.3 Chosen Model

Starting from the Random Forest model, this is very easy to implement, but its performance and scalability are lower than the ones of the other models described, so it has been discarded.

LightGBM is a very efficient machine learning model which works very well with large datasets with a high number of features. However, it is foreseen to not have a high number of features, which is needed to get optimal performance using this model, which is around a thousand at least. Moreover, this model is robust only if the tuning is very well executed, and this operation requires some additional effort.

In the end of the analysis, the most suitable models were CatBoost and XGBoost, due to their capability to manage categorical and numerical features, the low-to-medium computational cost for training and executing the model with the available resources, and their efficiency.

The final choice is XGBoost because it performs better with large datasets, like the one the ApeCar team has, containing all the logs from multiple instances of internal backend servers.

4.2 Implementation

In order to implement the chosen model, including its training and execution, it is important to select a programming language with the corresponding library to access the XGBoost model. After several considerations based on availability of the model, versatility, completeness of functionalities provided by the currently available libraries, the final choice for the programming language has been Python.

Moreover, according to the chosen solution, the log analysis is needed to be executed on all the instances of two out of the three internal backend services.

Before implementing a specific solution for each of the two different services and their multiple instances, which would require more effort than doing it for just one service, the ApeCar team chose to start by implementing the solution for just a service, including all its instances. This approach allows for validation of the implemented solution, and then replication on the other instances.

This methodology is feasible thanks to the log structure on the two services, which is very similar, allowing the logs to be parsed with slight changes to the parsers for the two different services.

4.2.1 Project configuration

In order to train and execute an XGBoost machine learning model with the chosen programming language, which is Python, it is important to include all the libraries allowing to do these operations, which are the "xgboost" library and other ones allowing data manipulation, hyper-parameters tuning, etc. These libraries have to be included in the scripts to train and execute the XGBoost model.

Folders

The project has been divided into different folders to separate different types of files, scripts, etc. Going in further details, in the project there are the following folders:

- logs: contains all the logs used to train the XGBoost machine learning model;
- logs_for_tests: contains all the logs used to test the trained XGBoost model and evaluate its performance;
- utils: contains the script created to parse logs, producing the parsed logs used to train the model and the ones to test its performance;
- main: contains the script to train the machine learning model, the script to execute it and the file containing the model trained in the .pkl format.

4.2.2 Logs

The choice of the ApeCar team is to implement the solution only for a single service of the MyPayments internal backend, so the one considered will be the service on the BE2 server instances.

In order to obtain the detailed logs, it is necessary to access the different instances of the server and to activate debug mode for the new logs. In this way, the model will have more information to use during the training process.

After activating the logs in debug mode, it was necessary to wait several days to collect enough data usable to properly train the machine learning model and to parse them, considering almost the majority of the possible APIs called by the mobile application instances.

Once the log files have been produced, I accessed the machines, moved them in a folder allowed to download files from it, and downloaded them. After downloading the log files, I put them into the "logs" and "logs_for_tests" folders of the project, in such a way that each log file is only in a folder.

With this approach, the data used for testing are different from those used during the training process, so the performance measures of the trained model executed on the test dataset are more accurate and reflective of a real usage scenario.

4.2.3 Log analysis

After adding the log files inside the project folders, I started analyzing the log structure to identify all the information which may be useful to train the XGBoost machine learning model, also by computing additional features using data within the logs.

Here is a log example to clearly explain the structure of the log files:

```
2025-03-12 15:53:14.964 [20ca3541-9e83-44a1-8875-6365785684dd] INFO - ======>
    Incoming Request | Flow: AUTHENTICATION | URI: /BE2/auth | Method: POST |
    Request Body: {...} | Content-Type: application/vnd.nets.v6+json

2025-03-12 15:53:14.984 [20ca3541-9e83-44a1-8875-6365785684dd] INFO - Outgoing
    Request ======> | URI: https://outerservice.com/auth | Method: POST |
    Headers: [Accept:"text/plain, application/json, application/*+json, */*", X-
    Target-Portal:"SP", Content-Type:"application/json", Content-Length:"81", X-
    Trace-Id:"20ca3541-9e83-44a1-8875-6365785684dd"] | Request Body: {...}

2025-03-12 15:53:15.534 [20ca3541-9e83-44a1-8875-6365785684dd] DEBUG- Incoming
    Response <====== | URI: https://outerservice.com/auth | | Status Code: 200 |
     Headers: [Date:"Wed, 12 Mar 2025 14:53:15 GMT", Server:"", Strict-Transport
    -Security:"max-age=3600; includeSubDomains; preload", Content-Type:"
    application/json; charset=utf-8", Content-Length:"1465", Keep-Alive:"timeout
    =5, max=300", Connection:"Keep-Alive"] | Response Body: {...}

2025-03-12 15:53:19.469 [20ca3541-9e83-44a1-8875-6365785684dd] INFO - <=====
    Outgoing Response | Flow: AUTHENTICATION | URI: /BE2/auth | Status Code: 200
```

As seen in these logs, some requests perform additional requests to external services, like the portals, to provide the correct service to the mobile application calling the API.

Initially, the choice was to analyze the entire sequence of requests needed to provide a response for a single API, also considering the status codes of each request between the incoming request and the outgoing response, in order to have very detailed data to train the XGBoost model.

This approach led to several complexities which may would have not provided any advantage, so the analysis was simplified by only considering all the requests in the sequence in terms of message length, response times, etc., without considering the URI, incoming responses status codes and other information, which would have made explode the number of features used to train the machine learning model.

This choice has also been made to train the model in a very efficient way, keeping a high level of performance.

The objective is to monitor API calls independently, so it is necessary to group all logs according to something which is unique for every incoming request. This is crucial to keep track of the request execution and analyze its data to identify any anomalies and their type.

The unique possibility to group different log rows together based on a single API call is by using the X-Trace-Id, which is the value inside the square parenthesis, to analyze all the requests and responses as part of the same server-side request execution.

Looking at the log file example, it is possible to identify some features which can be directly extracted by reading them. Some of these features, like the date and the X-Trace-Id, are meaningless for any analysis, since the X-Trace-Id is unique for every request sequence, since it is a UUID. The date is also unique, but it may be useful to extract the day of the week from it to help the machine learning model detect potential patterns during the training phase.

There are some other features which are more interesting than these first ones, so they can be parsed to train the model. These features are the URI, which is a categorical feature since there are a certain amount of different URIs associated to the APIs exposed by the service under analysis, the hour of the day, the status code of the outgoing response, which allows one to detect malfunctions or absences of service for a specific API, so that the ApeCar team can deeper investigate to better understand what happens with the involved API, the API method, and the flow associated with the request, that indicates a specific functionality provided by the service.

Thanks to the grouping by X-Trace-Id of the log rows, it is also possible to compute other features from the logs. These features are different from those directly extractable form the logs, like the URI and so on.

The derived features which have been considered meaningful are the average message length over the requests sequence, which may indicate the presence of errors if its value is too high, or the absence of the expected data in the responses if its value is too low, the day of week, for the same motivation of the hour of the day, and the total response time, computed considering the timestamps of the log rows with the same X-Trace-Id, starting from the incoming request and ending with the outgoing response.

After analyzing the logs and identifying the interesting and meaningful features, it is time to parse the log files and produce data in tabular form, so that the XGBoost model can be trained using the clean dataset produced.

4.2.4 Log parsing

In order to be able to use the data contained inside the log files to train the machine learning model, it is needed to convert them from the raw format to a tabular one, after having identified the features and their possible values.

Starting from the log files, it is needed to understand their structure to be able to parse them properly. Since the amount of logs needed to train and test the XGBoost model is very high, it is necessary to perform this operation in an automatic way, for example by writing a script to produce a file containing data in a tabular form, like a .csv file.

Continuing with this approach, the single rows of the log files have to be properly

parsed, extracting from them the values of the identified features to be used by the XGBoost model.

In order to extract them, the most efficient method to be used consists of defining specific regular expressions for the different elements of a log row, so that the elements can be correctly identified and inserted inside the file containing the dataset in tabular form.

Specifically different patterns and regular expressions have been used, because it was needed to identify the single log row at first, and then also to identify different features inside it, like the URIs, containing a string in a specific format, the status code string with the numeric value, needing another kind of pattern to be identified, and also method and flow from the request.

The parser script also has to be able to group all the log rows with the same X-Trace-Id, so it is possible to group the different requests and responses of the same sequence thanks to its value.

Moreover the script also has to compute some derived features, like the total response time from the incoming request to the outgoing response; this has been computed by performing the proper subtraction between the timestamp of the outgoing response and the incoming request with the same X-Trace-Id.

In the end, also the anomaly type has to be computed, in order to train the model, and to verify the performance of the already-trained model, right after testing it on the test dataset. The trained model will not consider the anomaly type feature when performing the predictions, otherwise, this solution would be pointless.

In the following table some examples of parsed sequences of requests and response for different single API calls are shown, in order to provide more details about the output of the log parsing operation:

URI	**Method**	**Flow**	**Status code**	**Hour**	**Day of week**	**Message length (Avg)**	**Response time (ms)**	**Anomaly type**
/auth	POST	AUTH	200	17	2	204.98	1236	0
/events	POST	AUTH	204	14	4	111.00	14	0
/auth	POST	AUTH	204	14	1	175.25	3676	0

After the log parsing operation, the training and test datasets are ready to be used in training the machine learning model, and testing it right after the training phase to evaluate its performance and make changes if needed to improve them.

4.2.5 Training

For the training operation, it was necessary to write a new Python script, as well as for the other operations.

This script went through different versions throughout the thesis work for several reasons. All versions of the script have been realized to improve the trained model's

performance as much as possible, until achieving satisfying and meaningful results to resolve the ApeCar team problem.

Both the initial version and final one will be described, along with all the increments added to the initial script to reach the final version.

Features changes and filtering

In the initial version, there was no Flow feature, which enables the machine learning model to distinguish the APIs also based on the logic flow they belong to within the backend. After considering the flow value as a new feature to train the model, it became also possible to detect additional patterns, so that the overall performance was improved rather than before.

Moreover, in the initial versions of this script, all requests and responses sequences were considered, even though some of them did not conform to the standard characteristics. Since these cases occurred, I chose to not include these sequences because they caused performance loss if used during the training operation.

Some of these sequences have a response time of zero, which is physically impossible. They may correspond to some log rows with internal messages or exceptions which are not associated with any time interval, but only with a single timestamp in the row, making it impossible to identify a complete sequence.

Missing values management

During the first training executions, some features had no values for specific sequences related to some APIs. It was necessary to manage these situations to train the XGBoost model properly, considering all the available data in the correct form.

There were various features which may contain no value, so it has been set to unknown in some situations, or to the most meaningful one for that feature in other ones, as done in the following image:

```
data['uri'] = data['uri'].fillna('unknown')
data['method'] = data['method'].fillna('POST')
data['combined_flow'] = data['combined_flow'].fillna('unknown')
```

Figure 4.1. Filling missing values

Data standardization

After the first operations related to the data contained inside the dataset, now, operations needed to improve the machine learning model performance will be discussed.

Starting with data standardization, it is crucial to ensure that all the features have the same weight for the model, without having one of them being more relevant and dominant than other ones, because this situation would cause a biased model training associated with the involved feature.

Moreover, having standardized data improves the model's convergency speed, because the steps needed to train the model are less compared to non-standardized data. This is very important in this implementation, because the available resources are limited. Reducing the convergency time decreases the resource usage to train the model, as the time needed to train it with the available hardware, which consists only of a portable PC.

This technique also allows easier comparison of all the features' values, since they are on the same value scale, and this also increases the interpretability of the trained model.

To achieve these results, the operations done during this phase consist of removing the average value from all the values of a specific feature, so all of them will be centered around zero for every feature inside the dataset, and in scaling all these values so that the unit variance is one and the features' values are uniformly distributed over their scale.

Dataset splitting

Given the training dataset, it is also important to test the model during the training phase to measure errors and other metrics, so that the machine learning model's performance can be improved. In order to achieve these results, it is fundamental to correctly split the dataset into training and testing sets.

Dataset splitting also helps reducing the overfitting, because in some situations the model may learn too well on the training dataset, without properly generalizing analyze new data and negatively affecting the model overall performance metrics.

In this implementation, the whole dataset has been split, dedicating the eighty percent of itself to train the model and the remaining twenty percent to test it.

It is important to remember that there is another entire dataset dedicated to model testing; it will be discussed in the following paragraphs.

Once the dataset is properly split to train the model, it is important to ensure that it is balanced to avoid biases, increase the overall performance of the model, generalize correctly on the new data and recognize anomalies or rare events. In this implementation, for example, it is fundamental to identify anomalies with the highest possible recall, especially for the absence of service anomaly class.

Dataset balancing

The dataset obtained from all the collected logs, despite their volume, is very unbalanced considering the requests and responses sequences with an anomaly in the "absence of service" class.

Numerically, the number of sequences with this anomaly is less than ten out of several thousands. This imbalance may have led to a lack of performance in the trained machine learning model, so it was necessary to balance the dataset to have more occurrences of this anomaly class. Thanks to this operation, the XGBoost model could also predict this anomaly with a satisfying accuracy level.

The objective is to obtain a balanced dataset: this operation may be done in different ways, for example, by collecting more log files. This approach is the simplest, but it may not have led to a solution, since it is impossible to predict how many occurrences

of requests and responses sequences with a specific anomaly will occur within a defined time period.

Due to these conditions, it would have taken an undefined time to collect a certain amount of samples with the anomaly type "absence of service" directly from the log files.

In the end, the logs could not be collected as described earlier to balance the dataset, so it was needed to do it with a different technique, for example, by generating valid sequences to train the model properly and increase its performance.

```
smote = SMOTE(random_state=42, k_neighbors=2)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)
```

Figure 4.2. SMOTE technique application

In this script the usage of SMOTE, which means Synthetic Minority Over-Sampling Technique, was needed, to increase the number of samples for the "absence of service" anomaly class.

This technique generates synthetic samples of the minority class within the dataset, not by simply duplicating the already existing ones, but by creating new samples which are similar to the real ones.

Thanks to the implementation of this technique within the script, the trained XGBoost model will more accurately detect the sequences with the anomaly class involved, and it will also be less biased toward the most dominant class, as all classes will have a balanced amount of samples.

Although the "absence of service" anomaly is rare, it is important to have the highest possible recall, so that the model will recognize almost all the occurrences of the event.

In conclusion, the model will detect more accurately this class of sequences, which is fundamental since its impact is extremely high on the merchants of the MyPayments application.

Hyperparameter tuning

As already well known, every model needs to be tuned to reach a higher performance level, given the same dataset and other conditions. There are different approaches which can be followed to tune the hyperparameters of a machine learning model, for example by testing random values. Although it may look strange or useless, it can be a good solution in some cases for several reasons.

For this XGBoost model the approach I chose to use consists of using GridSearchCV, which allows to explore all the possible hyperparameters combinations within a defined set, so that the search is exhaustive and complete.

With GridSearchCV it is possible to define a set of hyperparameters with their possible values to execute the exploration. This set is defined through a parameters grid, which is used during the search operation.

Moreover, also the GridSearchCV needs some input parameters to be inserted to effectively execute the search operation. Among these parameters, there are some interesting

ones to be discussed, like the scoring method, which is used to evaluate the performance of a certain set of hyperparameters tested, and the validation method, which is the stratified K-fold in this case.

About the scoring method, there are different values for it, like accuracy, precision, and so on. In this implementation the chosen one has been the f1 macro, which is based on the harmonic mean computed on the precision and the recall.

Using this scoring method is fundamental if a dataset is unbalanced, even after balancing with SMOTE technique or other ones, because this combination of techniques ensure better robustness of the model and a correct performance evaluation during the training phase.

In further detail, the f1 macro scoring method computes the F1 score for every predicted class. The harmonic mean is computed by calculating the average of the F1 scores of all the classes, treating them equally, independently from the amount of samples analyzed for each of them.

Regarding the other interesting parameter, it is the validation method, which corresponds to the stratified K-fold in this implementation. This method was chosen because the dataset is unbalanced, so other possible methods like the standard K-fold, the Leave-One-Out method, etc. would not have been effective for this dataset, since they have better performance when the dataset is balanced.

Thanks to this choice, the evaluations done during training operation are more reliable, as the results are more stable and the model will be evaluated in a more appropriate way. Moreover, the overfitting is prevented, because the model is evaluated after splitting the dataset in different ways, increasing the generalization level. Furthermore, all the classes are proportionally represented in every split datasets, because their distribution is kept the same as in the original dataset, and it cannot happen that a fold is unbalanced, which may cause a wrong evaluation of the trained model.

Model training and exporting

After all this preparation and these operations to properly prepare the dataset and tune the hyperparameters of the model in the best possible way, the model is trained on the machine and then exported in a file in the format .pkl.

The file is inside the main folder of the project, so it can be used during the model execution phase to run it with other datasets, different from the ones used for training and testing.

4.2.6 Model execution

Once the model is ready, it is possible to execute it on the test dataset. Another Python script is needed to execute the model at this moment. This new script has to import the trained model, the dataset and all the other elements needed to execute the model. After the import operations, the response time ratio is computed, as in the training script, and the features are encoded with the One-Hot-Encoding technique, since the XGBoost machine learning model only works with numeric values. Moreover, also the response time ratio thresholds are computed as in the previous script.

Other minor operations are performed to align features, sort columns, etc., but they will not be described in detail.

The XGBoost model can be executed at this moment, right after preprocessing and preparation operations. Its execution returns the column with the anomaly class prediction for every row in the test dataset.

After the model execution, it is very important to evaluate the performance of the trained model to understand the results produced by the training operation and identify potential problems of the model.

The performance evaluation operation has been included in the script executing the model, and it consists in computing relevant metrics like recall, accuracy, precision and F1 score, so that they can be printed and shown as results of the model execution.

4.2.7 Post-execution logic (Results cleaning)

Post-execution logic consists of deterministic algorithms executed after the model prediction is completed to remove false positives or false negatives which can easily be identified and removed. Thanks to this operation, it is possible to provide more accurate results and increase the overall performance of the model together with all the logic surrounding it.

The logic introduced is crucial to increase the precision of the trained model based on the actual data inside the dataset, ensuring the prediction is coherent with it. It also removes false positive predictions, since there are some scenarios in which the model classified some requests and responses sequences as anomaly, although they were not, for example with the absence of service anomaly class, for which the recall was maximum, but the precision and accuracy were very low.

Since the log rows for the class with this anomaly type were too few, having more of them would have been very useful to increase the accuracy of the model without any post-execution logic, but new logs have not been collected for different reasons.

One reason is that it would have been needed to activate logs in debug mode for some other days on the machines, but this would have used more resources of the server instances, so this solution was not chosen. Another reason is that the same final results could be obtained by adding some easy post-execution logic, analyzable, and implementable in a very short amount of time, so this solution has been chosen to improve the final results provided by the script executing the machine learning model.

Here it is the code logic implemented:

```
new_data.loc[(new_data['predicted_anomaly'] == 2) &
             (original_columns['status_code_final'] >= 500) &
             (original_columns['status_code_final'] < 600), 'anomaly_type'] = 'Absence of service'
```

Figure 4.3. Post-execution logic for absence of service anomaly class

In the end of this added operations, the recall is still maximum, but also the accuracy and the precision are much higher than before.

It is very important to consider that the objective of the ApeCar team was to have the highest possible recall. This objective was already accomplished without the post-execution logic, but there would have been too many false positives to check, making the alerts useless in this situation.

The model has produced the final outcome, properly identifying the different possible anomalies, with a very high level of accuracy and recall, meeting the needs of the ApeCar team.

Chapter 5

Results

Now let's discuss all the results obtained from the implementation done, including the log parsing script, the training script and the script to execute the chosen machine learning model after analyzing several ones.

It is important to always consider the requirements to properly evaluate the results of any work, especially in this case where the ApeCar had specific needs for the product it manages.

In the next paragraphs, the results of the log parsing script will be analyzed, with a qualitative approach since no numbers are available for the operation it executes. For the XGBoost model trained, the results and metrics will be described in both a qualitative and quantitative way.

These results have been obtained thanks to the log data collected by the internal backend server instances of the MyPayments application, although it was not possible to collect a very high volume of data for several reasons, as previously explained.

5.1 Numerical results and performance

Let's analyze the results, distinguishing between the log parsing and everything related the XGBoost model.

5.1.1 Log parsing results

Considering this part of the project, it has produced the expected results. After collecting a sufficient number of logs, in terms of tens of thousands log rows, all the different row types have been correctly identified and parsed with the developed script.

Another positive result is that also during the execution of the model with a completely disjunct dataset from the one used to train the model, all the log rows were parsed correctly and no information has gone lost.

In the end, this part of the project has satisfied the requirements, being able to successfully parse and analyze all the occurrences of the log rows of the BE2 service.

Since the log structure is the same for both BE1 and BE2 services, this implementation can easily be applied also to the BE1 service as well.

5.1.2 XGBoost model results

These results are crucial to meet the requirements defined by the ApeCar team, and also to realize the log-analysis system properly.

The metrics obtained can be split by anomaly class, because the model may be more performing on some anomalies rather than on others.

Absence of service anomaly

Starting with this anomaly, the XGBoost model initially did not reach a very high accuracy. However, thanks to the post model execution logic implemented, the model accuracy has been increased by removing all the false positives, and the recall is maximum.

As mentioned, this result for this type of anomaly respects the ApeCar needs, and it is the desired result for this log analysis work.

In the end, it is crucial to identify this kind of anomalies because it allows for their immediate detection, as they may be sign of something impacting very much on the merchants using the MyPayments application. Therefore, it is necessary to react quickly to minimize any potential downtime or partial absence of service for any feature.

Response time anomaly

Considering the other anomaly class, which is the response time anomaly, the table with predictions is displayed after executing the machine learning model, and it is shown in the following to better explain the obtained results:

Table 5.1. Prediction details of test dataset - example

ID	Response Time	Msg Length	Status	Anomaly Type
1	3362	199.25	200	None
7	3	288.00	200	Response Time
13	1234	187.69	200	None
15	17076	231.87	200	None
2217	24815	223.48	500	Absence of service
2310	86836	242.78	204	Response Time
2705	155707	251.79	200	Response Time

As shown in the prediction table for the test dataset, which is completely disjunct from the training one, several sequences of requests and responses have very high or low response times, comparing them to the values considered as normal by the machine learning model. The normal values were obtained after having grouped them by URI, status code and method, in order to have more specific values for every possible scenario.

In this way, thanks to these results, the ApeCar team will be able to receive an alert when an anomaly occurs, in order to check if everything is working properly, or if there are any kind of issues, like network ones, overload problems on external services, and so on.

This is very useful because sometimes there may be problems on external services, but the teams responsible of them may not be aware of the issues. This may degrade the user experience within the app and slow down reaction times compared to having an alert system.

In conclusion, after getting this kind of information, the reaction by any team involved in the problem will be faster, so that the overall user experience of the merchants inside the MyPayments app will be better, as requested and needed by the ApeCar team.

5.2 Methodology

After having displayed the results, it is important to also describe how they were obtained, so the methodology used in each phase.

5.2.1 Log parsing

During the log parsing operation, when writing the script, a sufficient number of logs needs to be collected, to be able to consider all possible log rows. If this is not done, the parsing would be incomplete and information would be lost after executing the parsing operation.

After having collected log files, it is necessary to explore them, understand the structure of the single rows, identify and define some strategies to parse them to get a result usable as training or testing data for a machine learning model. A possible choice is using the regex approach, though there may be other equally valid possibilities.

This activity requires various changes and adjustments to realize a script capable of identifying all possible occurrences within the log files, as well as in any other kind of data source.

5.2.2 Machine learning model training

This is a critical phase for this work, because all the results of the entire project depend on how well the model is trained, using the correct methodologies, which must be based on the data collected for training.

In this implementation the usage of some techniques has been needed as already described, like the SMOTE and so on, but with other datasets others may be needed, in order to satisfy the requirements and obtain the highest possible performance for the trained model.

It is fundamental to consider the value distribution for all the features within the dataset to avoid biases in the final model.

Moreover, the testing strategy has to be chosen appropriately according to the data distribution of the training dataset, otherwise, the metrics obtained may be unrealistic or simply wrong.

After doing these considerations, training can be executed and results have to be analyzed, to understand if performance can be improved through additional training executions, considering any changes which could potentially benefit the model to better satisfy the given requirements.

When the model satisfies the requirements, it was tested with a completely different dataset in this project to evaluate its performance more realistically than during the training phase.

5.2.3 Machine learning model execution

After training the machine learning model, it was tested with a complete disjunct data set to simulate a real use case as accurately as possible in the actual environment.

This operation was executed several times because, although the training phase produces very good metrics and performance values, the model may have very different performance when tested. Therefore, something needs to be changed during the training phase or the parsing one, such as considering new features, etc., in order to reach the objective and satisfy the given requirements.

As described, it is always important to consider and evaluate all the aspects affecting the final results, to be able to make changes at any time to reach the defined goal.

Chapter 6

Conclusion and future implementations

The practical thesis work has been successfully completed, since the model works as expected and required by the ApeCar team.

After the already described implementation, it should be integrated into a running system inside the Nexi Group's infrastructure to be actually used in real-time to analyze logs and send alerts if any of the anomalies defined occur.

This implementation would have taken too long because changes to the company IT infrastructure are required. These changes need different stakeholders and technical staff analyzing and approving them, as well as investment in new machines, configurations, and everything else necessary. Moreover, some other teams should have also worked on this implementation, in order to correctly integrate the new machines inside the infrastructure, configure the network to properly work, and so on.

Due to these factors, it is currently only possible to design a possible implementation meeting the requirements defined by the ApeCar team, explaining its working mechanisms in some detail.

Before going on to the possible solutions, the main requirements will be briefly summarized.

The main requirements consist in analyzing the logs in real-time, to react quicker to incidents or become aware of them before being notified from the merchants using the MyPayments application, and sending alerts immediately after any of the anomalies is detected.

The work completed until now allows the team to detect the anomalies, as earlier explained and shown in section 5, which describes the results obtained. With a small and simple add-on, it is possible to send an email alert to a defined recipient list, but it is not possible yet to analyze logs in real-time.

In the following sections, different potential solutions to achieve this objective will be discussed, at least by designing the system at a high level.

6.1 Possible implementations

After identifying the requirements for this kind of implementation, it is time to consider and evaluate different potential implementations based on their characteristics.

6.1.1 Requirements

Since the new system which has to analyze logs in real-time, has to be implemented in a data center or on the cloud, and to receive data by the MyPayments internal backend through the network, it is necessary to consider architectural solutions compatible with this scenario. Moreover, it is also important to consider that its implementation has to be done with Python programming language to run the log parsing scripts and the script to execute the XGBoost model.

Given this information, it is possible to identify compatible solutions for this implementation, analyze them, and choose one to design the new system.

6.1.2 Possible approaches

There are possible approaches for these implementations, for example, using REST APIs to send the logs and then process them, or by using a broker like RabbitMQ, implementing a message queuing system, or by using a streaming platform like Kafka. The last two approaches are quite similar under different aspects, which will be discussed later on in detail.

6.1.3 REST APIs approach

This is the most common approach on the web to execute Remote Procedure Calls by sending HTTP requests.

REST APIs could be chosen to implement the already described system, but it is fundamental to consider their main characteristics and the given requirements.

REST APIs would need to send an HTTP request containing the log data in this implementation, with the objective of obtaining a response, but the MyPayments internal backend does not need it, because it only has to send the logs to the new system which will analyze them. Analyzing other characteristics, it is important to remember that HTTP requests generate some overhead due to their nature, so managing a large amount of them would be consume a significant amount of resources, resulting in an overload of the log-analysis system. This overload could anyway be managed by introducing a buffering or queuing mechanism, in order to not overload the HTTP server directly.

After analyzing these characteristics, this approach does not seem to suit the requirements described earlier very well.

6.1.4 RabbitMQ and Kafka approaches

RabbitMQ and Kafka are grouped in the same paragraph because their implementation would be conceptually similar. However, there are some important technical differences

between them which have to be considered to make the most appropriate choice for the final implementation.

RabbitMQ is based on a queue mechanism, so all incoming messages are enqueued until consumed by the system. Moreover, it ensures that the messages reach their destination, as also Kafka does, but in a different way.

Kafka is based on a streaming mechanism, it is more scalable than RabbitMQ, but it needs more effort to be implemented.

Delving into technical details, Kafka is designed to handle a very high number of connections including their messages, ensuring a very high throughput. RabbitMQ would be overloaded under these conditions, but this is not expected to happen in this scenario.

The maximum number of connections from the MyPayments internal backend servers to the log-analysis system would be around a hundred. Thanks to this value, the implementation with Kafka would may be over-dimensioned for the system to be implemented, but it remains a very good choice.

There are other factors to consider, like the configuration effort necessary to implement them: RabbitMQ is easier and faster to implement and set up rather than Kafka, so it would be more cost-effective in economic terms to the company. Moreover, Kafka is very good also at managing persistent data, which is not needed in the system being discussed, since it is possible to temporarily store the logs in variables until they are analyzed, to be removed immediately after this process.

Talking about resource consumption, which would be very important if the system is implemented in the cloud, Kafka is much more efficient at managing a high volume of data, while it may be less efficient than RabbitMQ at managing a lower data volume.

In conclusion, as previously discussed, the approaches of Kafka and RabbitMQ are similar in terms of functionalities, but the described differences are important to consider to choose the most suitable approach for this system.

6.2 Chosen approach for future implementation

In the end, three approaches have been analyzed, including technical details and some comparisons for Kafka and RabbitMQ.

The REST APIs approach is not the best one and does not adapt very well to this implementation, since it could need some extra mechanisms in order to work as it should, but the overhead generated by the HTTP requests cannot be avoided in any way.

After these considerations, this approach is discarded in favor of one of the remaining ones.

Considering Kafka and RabbitMQ, they work in a similar way as described above. There are some technical differences, like the efficiency, the number of connections they can manage at the same time, and so on, that suggest to choose RabbitMQ, because although it is less scalable than Kafka, it is correctly dimensioned to the requirements of the ApeCar team, and it also needs less effort to be implemented. Talking about the scalability of these two solutions, Kafka is the most scalable one, while RabbitMQ is less scalable at very high data volumes due to the complexity needed to manage the message queues.

In conclusion, the choice goes on RabbitMQ, because the amount of logs to be analyzed is not very high, for sure under millions of log rows per day.

Given these considerations, the log analysis system may only need a single instance in order to execute the analysis. The designed system can also be scaled up to manage more logs simultaneously up to a certain limit in case of necessity, which is unlikely to be reached in the medium term.

6.3 Working mechanism

Currently, it is possible to define how the implementation with the chosen approach should work, including more technical details.

The mechanism is very linear, so the solution should use queues in RabbitMQ to receive logs in real time from the different server instances within the MyPayments internal backend. These logs should be stored inside some variables until a final response log is received for a certain sequence.

Once this log is received, the complete sequence will be retrieved from the variables and passed to the log parser. In this way, all data is converted and formatted as needed by the XGBoost model, which will then make the prediction on it. At the same time, the requests and responses of the sequence will be removed from the variables containing them, since they will no longer be accessed or used. Thanks to this approach, the resource usage will be more efficient and optimized, which is crucial for running any kind of service onto the cloud, for example, but also on any server machine to make it capable of managing a higher volume of requests and responses sequences.

The parsed sequence of logs has to be passed to the script executing the machine learning model to execute the prediction on the anomaly. When the prediction is ready, it can be refined using the previously described post-prediction logic to obtain the final prediction.

Once the prediction is complete, it has to be processed by an algorithm that analyzes it to send an email alert if necessary. This feature can be configured and implemented in different ways, which will not be discussed here in detail.

In conclusion, the requests and responses sequence has been analyzed, sending the proper alerts to the configured email addresses, as in the needs of the ApeCar team.

6.4 Expected results

After completing this implementation, the expected result is a log analysis system which works as described and satisfies the needs of the ApeCar team for real-time analyzing logs.

Moreover, the predictions made by XGBoost are expected to be performed keeping the same performance metrics as observed during the testing phase of the machine learning model, which was conducted using a completely disjunct dataset from the training one.

In addition, if any anomalies are detected, the system is expected to send an email to all the addresses configured within the new system, so that the relevant people become aware of the anomalies as soon as possible. This allows to react them in a shorter possible

time than before and increase the user experience and value provided to the merchants using the MyPayments application, by minimizing absences of services and malfunctions.

Bibliography

[1] Backend architecture. page 1. MyPayments App Information Folder, 2023.

[2] Business statistics. page 1. MyPayments App Store Page, 2025.

[3] Dod for apecar team (story level). page 1. ApeCar team, 2025.

[4] Safe framework inside art. page 1. Scaled Agile, Inc, 2025.

[5] Transactions and settlements. page 1. MyPayments App Store Page, 2025.

[6] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016.

[7] Gavin Hackeling. *Mastering Machine Learning with scikit-learn*. Packt Publishing Ltd, 2017.

[8] Mohamed Hassan. Choosing the right communication protocol for your web application. *arXiv preprint arXiv:2409.07360*, 2024.

[9] Bhole Rahul Hiraman. A study of apache kafka in big data stream processing. In *2018 International Conference on Information, Communication, Engineering and Technology (ICICET)*. IEEE, 2018.

[10] Xian Jun Hong, Hyun Sik Yang, and Young Han Kim. Performance analysis of restful api and rabbitmq for microservice web application. In *2018 International Conference on Information and Communication Technology Convergence (ICTC)*. IEEE, 2018.

[11] Jay Kreps, Neha Narkhede, and Jun Rao. Kafka: A distributed messaging system for log processing. In *Proceedings of the NetDB*, volume 11, 2011.

[12] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Mathieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[13] Peter Prettenhofer and Gilles Louppe. Gradient boosted regression trees in scikit-learn. In *PyData 2014*, 2014.

[14] Sebastian Raschka, Joshua Patterson, and Corey Nolet. Machine learning in python: Main developments and technology trends in data science, machine learning, and artificial intelligence. *Information*, 11(4):193, 2020.

[15] Maciej Rostanski, Krzysztof Grochla, and Aleksander Seman. Evaluation of highly available and fault-tolerant middleware clustered architectures using rabbitmq. In *2014 Federated Conference on Computer Science and Information Systems*. IEEE, 2014.

[16] T. Sharvari and K. Sowmya Nag. A study on modern messaging systems-kafka, rabbitmq and nats streaming. *CoRR*, abs/1912.03715, 2019.