Transaction monitoring

Final Report

T. Harting
S. Popping
M. Post
D. Swaab

bunq

TU Delft

TI3806 - Bachelor project - 2017

Transaction monitoring

Final Report

by

T. Harting
S. Popping
M. Post
D. Swaab

to obtain the degree of Bachelor of Science
at the Delft University of Technology.

Project duration:	April 24, 2017 – July 7, 2017	
Coaches:	Prof. dr. C. Lofi	TU Delft
	Ir. W. Van	bunq
Bachelor project coordinators:	Dr. O.W. Visser	
	Dr. H. Wang	

TU Delft

Preface

The report before you documents the bachelor project 'Transaction monitoring', which was conducted by Tom Harting, Sven Popping, Mathieu Post and Daniël Swaab. This project is part of the Computer Science Bachelor program at Delft University of Technology. It serves as the final project before obtaining the degree of Bachelor of Science. Over the past three months, we were engaged in conducting this project and writing this report. Within this project, we designed and implemented a system which deals with monitoring transactions for fraud. This project was commissioned by bunq.

We would like to thank everyone at bunq for their open attitude and help over the course of our project. Our questions were never left unanswered and we received all the help we needed. Special thanks go out to Wessel Van (developer) and Ali el Hassouni (data scientist) for their guidance and support.

Finally, we would like to especially thank Prof. Dr. Christoph Lofi from Delft University of Technology for his advice, feedback and clear explanations.

Tom Harting
Sven Popping
Mathieu Post
Daniël Swaab
Amsterdam, June 2017

Summary

Being a bank, bunq deals with transaction fraud on a regular basis. All transactions that are handled by bunq are monitored for these cases of fraud by a transaction monitoring system. When this system flags a transaction as being possibly fraudulent, a bunq employee has to manually check this transaction. The problem with the current system is that it proves to be time consuming and labor intensive. This is caused by the fact that there are a lot of transactions which are falsely flagged as possibly fraudulent, these transactions are called false positives. This resulted in a demand for a system which reduced the number of false positives and thereby the time needed to manually check the flagged transactions.

To fulfill this demand, bunq has been working on creating a machine learning model which classifies transactions as fraudulent or legitamte. This machine learning model has shown promising results during test runs on historical data. However, it was not yet production ready, because it was very slow and there existed no connection with the existing bunq back-end.

During the project, a new transaction monitoring system was designed and implemented. The new system uses a combination of a bunq-made machine learning model and a set of pre-defined rules to flag a transaction as possibly fraudulent or not. The final system implementation consists out of 5 different components: (1) an incoming transaction system, responsible for noticing new transactions and segregating those over different workers so that they can be classified in parallel, (2) an information gathering system, which efficiently gathers large sets of needed information for the classification, (3) a machine learning model server, which enables fast communication with altering machine learning models, (4) a set of pre-defined rules, which check transactions for indicators of fraud and (5) a Grafana dashboard which monitors the performance and statistics of the machine learning model, the pre-defined rules and our system.

The system is fully tested by unit and integration tests. Furthermore, new machine learning models and pre-defined rules can be easily adopted. All the above mentioned components are implemented and fully working. The final implementation is focused on integrating the system in the currently existing bunq back-end, because the system will actually be used in production.

Contents

1 Introduction

Our project took place at a company named bunq, a self-proclaimed 'IT company with a banking license'. bunq is a bank that overthrows the traditional way of banking and focuses on making banking more personal and social. Instead of an unwieldy big company (like most other banks), they are much more agile.

Because bunq is a bank they handle a lot of transactions. An inescapable part of handling so many transactions, is the fact that some of these transactions are fraudulent. At bunq, there are currently fraud detection and prevention systems in place, which are aimed at detecting and preventing those fraudulent transactions. bunq is constantly improving these transaction monitoring systems, because the people who are committing fraud are continuously trying out new strategies to avoid being caught.

The goal of our project is to create a flexible system which communicates with a (changing) machine learning model that classifies possible fraudulent transactions. To do this, the system has to be connected to the systems already present at bunq. Furthermore, our system should be able to use custom monitoring rules to support the machine learning model. Moreover, it needs to provide statistics and visualizations about these models and rules. This way, a new transaction monitoring system based on machine learning can be implemented in the bunq workflow and these models and rules can be tweaked and compared using the statistics.

This report documents our progress throughout the project, the environment in which the project was conducted and recommendations for further improvements of the transaction monitoring system.

2

Research stage

A big part of creating a software solution is research into the problem and it's environment. This research is needed to arrive at the best techniques and solutions for the given project. At the beginning of the eleven project weeks, two weeks are spent on doing the research, which is presented in the following sections. First a detailed description of the problem, the current status and the assignment are formulated. Then a structure for the application is proposed, this application interacts with the machine learning model and two currently existing systems of bunq. The possible solutions for the interaction with each of these systems are discussed and we will substantiate the final decisions regarding these solutions. After this, we discuss other requirements that have to be taken into account when developing this system. Finally, a conclusion is given that shows the complete set-up of the system.

2.1. Problem definition and analyses

In this section we will give a description of the project problem. Furthermore, we analyze the current situation. From this, we will derive the assignment. A more detailed description of these topics is given in our project plan, which can be found in appendix A.

2.1.1. Problem Definition

Transaction handling is a very important aspect of a bank, no less for bunq. Thousands of transactions happen every day. Each and every one of these transactions needs to be checked. There are multiple reasons why a bank should check their transactions. First of all, there is a need for simple (trivial) checks. For example, when a person wants to do a payment, and this person's balance is not high enough, the transaction should be blocked. Next to these checks, a bank is also obligated to monitor their transactions for fraud and report fraud when it is found. This, among other things, prevents money laundering and the financing of terrorism [3]. Fraud leaves typical organizations with a loss of 5% of their revenues each year [2]. In bunq's case, the loss is not only caused by the fraudulent transactions themselves, but also by the time spent on monitoring these transactions.

2.1.2. Current Situation

[redacted]

2.1.3. Our Assignment

Our assignment can be described as follows. Although a machine learning model is being build, there is not yet a system in place to use this model to classify a transaction in production. There is a need

for such a system which allows a machine learning model to work with the bunq back-end. Furthermore, this system should be able to accept updated models, show statistics about the used model and compare the performance of these models. Next to this, our system should be able to use custom monitoring rules to support the machine learning model. [redacted].

To guarantee that the application is working properly, we need to monitor it. For example, are transactions still checked inside the time constraints? We also need to monitor the used machine learning model. How many transaction are flagged per day? How many false positives are found? And is this increasing or decreasing?

In the project plan, requirements were defined. In researching the set-up of our application, these requirements must be considered. The problem description above includes most of these requirements. The following paragraphs recap the remaining requirements to guarantee the quality of the final product.

Functionality The final product needs to work correctly. The number of false negatives should be limited and the number of false positives should decrease. The connection between the systems should be solid and stable.

Maintainability The final product will be used by bunq. Therefore it is important to write maintainable code. It should be clean, documented, tested and comply to the PSR-2 code-style guide [12]. bunq formulates this as follows: “It should not take more time to remove your code from the code base, then how much it took to build the code”.

Efficiency Transaction monitoring deals with a lot of data. While there is a need to monitor transactions in real-time, efficiency is a key aspect. The threshold given to us by bunq to classify one transaction is [redacted]. Besides this, the number of users at bunq continues to grow and therefore the product should be scalable.

Security & Privacy bunq is a bank, therefore the privacy and security of it’s users are of utmost importance. The final product will handle transaction data which is privacy sensitive and the model will be trained and tested on anonymized data. Therefore the application will need to be secure at all times.

2.2. General Structure

Based on the application requirements and the initial information of the systems running at bunq, a first general product structure is given in figure 2.1. This section will give an overview of this proposed structure. The rest of this report is dedicated to presenting a more in-depth research on the different aspects of this structure.

[redacted]

Next to the back-end, the application needs to gather the needed information for classifying from the database. As stated it has to do so in an efficient manner. This data from the database, is then used as input for the model. The details of how the application will connect to the database and which data the application needs, is discussed in chapter 2.5.

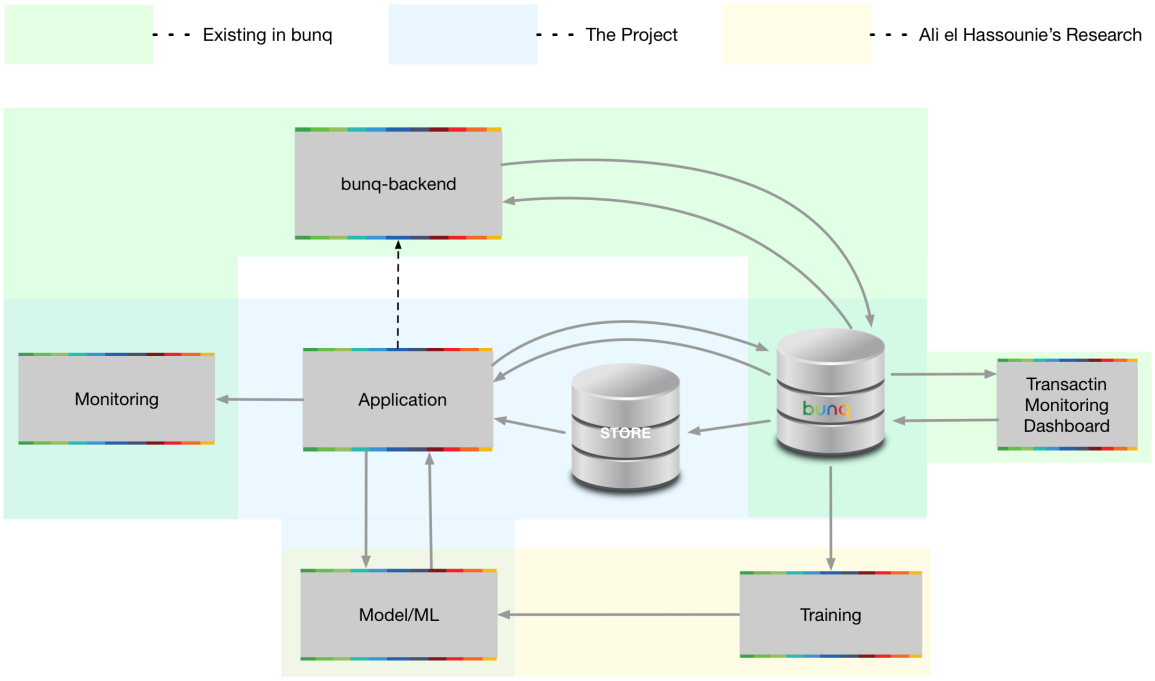


Figure 2.1: Schematic overview of the different systems and their relation to the project

There are two other systems that are related to our end product, the transaction monitoring dashboard and monitoring. The transaction monitoring dashboard is used to manually check the flagged transactions. This web application extracts data from the database. This web interface does not need to be replaced, while our application writes it's results to the same database as the current system. Monitoring (left in the figure) means checking the requirements the of the application itself (speed etc.) and analyzing the used machine learning model. In chapter 2.6, we explain which frameworks are used at bunq, how the application can integrate these and which data is important for monitoring.

In the yellow area the machine learning model and it's training are defined. The initial model comes from Ali el Hassounni's research [8]. Training reassembles a script that is used to train the model with the data from the database. The model will classify transactions using input provided by our application. In chapter 2.3 a more detailed description of the model and the interaction with the model will be given.

2.3. Machine learning model

In the field of machine learning, a lot of big steps have been made in the last couple of years. This is mainly the result of the enormously increasing availability of data, but also of the development of new algorithms [16]. There are a lot of frameworks which all use different algorithms or implementations to work with all this data. In this report we will use the term machine learning model, which is defined as the trained classifier that we use in our system to classify transactions as being possibly fraudulent or not.

2.3.1. Deep learning and GBM

Currently deep learning is a fast growing sub-field of machine learning. It has been shown that deep learning can find more complex patterns in (raw) data than what is currently possible with other machine learning techniques. The key difference between deep learning and other machine learning techniques is that the layers of deep learning algorithms are not created by human engineers, but are learned from the data itself. Prerequisite of using deep learning algorithms is that a lot of data is needed to train a deep learning model that can classify the data.

[redacted]

[redacted]

2.3.2. Frameworks

As stated in this report, our application integrates an existing machine learning model, made by Ali el Hassounni, in the bunq back-end. In his research [8], Ali el Hassounni used the H2O framework [13] to create his machine learning model. However, there are more free frameworks available. These frameworks are used to create the actual machine learning model which we will use in our application. Despite his experience with H20, Ali el Hassounni was willing to look at other frameworks to see if there would be a better fit for our application. This is why we compared different frameworks. We have selected some frameworks that are currently available, open source and widely used. We have compared the frameworks namely on available algorithms, benchmarks [21] and the availability of documentation.

- scikit-learn [6]
  - API: Python
  - Written in Python (some C and C++)
  + Applicable for almost every machine learning algorithm
  + Widely used by programmers
  + A lot of documentation
  + A lot of available libraries
  – Slow
  – Memory inefficient
- R packages [23]
  - API: R
  - Written in R, C and Fortran
  + Applicable for almost every machine learning algorithm
  + Widely used in the academic field
  + A lot of documentation
  + A lot of available libraries
  – Slow
  – Memory inefficient
- mlpack [19]
  - API: C++
  - Written in C++
  + Really fast because it is completely written in C++
  – Not a lot available algorithms (no GBM for example)
  – No native APIs other than for C++
- Apache Spark MLlib [1]
  - API: R, Python, Java, Scala
  - Written in Scala
  – Made to distibute over a cluster of servers
  – Really memory inefficient
- H2O [13]

  - API: R, Python, Java, Scala, REST
  - Written in Java, Python and R
  + Already used at bunq for research purposes
  + Fast
  + Memory efficient
  + Export trained model as POJO
  + Deploy the model as REST API
  + Lots of different options
  + Scalable
  + Also supports deep learning frameworks as backend
- xgboost
  - API: Python, R, Java, Scala, C++
  - Written in C++
  + GPU support
  + Fast
  + Memory efficient
  − Only capable of Distributed Gradient Boosting (GBDT, GBRT or GBM)

Our application has to be scalable, as the amount of transactions will probably increase a lot over time. Also, the hardware set-up should be easily maintainable. Therefore, we can already discard scikit-learn, R packages and Apache Spark MLlib. These three frameworks turned out to be quite slow and sometimes they will crash because of memory consumption on very big (>1M) data sets [21]. Furthermore, Apache Spark MLlib is made to run on clusters, which are hard to maintain and such an advanced solution is not needed for our project..

Another framework that does not meet our needs is mlpack. It turns out to be really fast using the algorithms that are available in the framework. However, since boosting machines are not implemented which perform best for us at the moment, it is not really suitable for us at this point [4, 21].

This leaves us with two frameworks which will probably work the best for our use case: H2O and xgboost. xgboost and H2O are pretty comparable in terms of performance for our purposes at the moment, but H2O offers a lot of more options. H2O has great support for a lot of different machine learning algorithms and offers a lot of parameters that can be used to fine-tune these algorithms. This makes it suitable to easily test different algorithms and parameters to check which ones have the best results and the least errors.

[redacted]

That in combination with the possibility of H2O to deploy the model as a REST API as well as creating a POJO of the model, makes it relatively easy to create a completely new model without editing much of the rest of the code base. It will be possible to deploy a new model which uses the same REST API.

Also not unimportant is the accuracy of the models for the different frameworks, which is almost the same. Depending on the amount and the kind of data xgboost or H2O performs a bit better. This makes that we have chosen to use H2O as our machine learning framework. As stated above, H2O is currently also used internally for research purposes, which is also an advantage. This combined with all the positive points described above makes H2O the best choice for our project. We will start by using the REST API to use the model within our application, if this does not meet our requirements we will look at using the exported POJO.

2.4. Back-end connection

In this section we first describe how the bunq back-end handles payments and where in this process the transaction monitoring should be added. In the second section we describe how the connection between the back-end and our application can be made, the most efficient way to do this and how we are going to implement this.

2.4.1. bunq back-end

[redacted]

2.4.2. Connection options

Our application should be connected to the existing back-end. The most important requirement for this connection is that it should be easy to install, however it should also take the same amount of time to remove.

For this connection there are several options. It could be done via a file system, the back-end writes data to a directory monitored by the application. Another option is via the database, the back-end and the application share a schema and use it to share data. It could also be done via a bridge, so the back-end and the application are integrated. The last option is via a message, the back-end sends a request to application to check a transaction and it will responded with the classification.

File system Using a file system for this problem is simply devious, because the request data should be really small. Furthermore this will create a latency that the application can not afford.

Database The bunq back-end and our application share the same database. Therefore it could be possible to have our application watch over the transaction scheme and go to work when a new transaction has been inserted. However, this will probably result in a big load on the database which it can not afford.

Another way is that the database sends a trigger to our application when a new record is inserted. However, according “The Trouble with Triggers” [18] this should be avoided, because it will lead to a

maintenance headache. “They should be used only when you cannot do something any other way” as stated by Tom Kyte [18].

Bridge Bridges are reliable and fast, but will increase the complexity on both sides of the application. In this case there is only one call from the back-end to the application, so making a bridge for this project is a bit much. Furthermore the requirement is that our application should be easy to add and remove, making bridge might make this to complicated.

Message Sending a message from the back-end to the application is the last option. This means that the back-end will send a message to our application. This has as advantages that it is easy to implement and can be easily added to and removed from the current back-end. [redacted].

We concluded that for this case using a message was the best solution, more specifically by using an API call. We concluded this because this will give the least amount of overhead and it is easy to expand.

The request the back-end needs to send to our application differs, when it is done before or after the transaction is committed to the database. The difference between these two is the body of the request. When it is before the commit, the request should contain all the information about the transaction. [redacted]

2.5. Gathering data

In this section we will first describe the specifics of the bunq database. We will then look at why our application needs a connection with this database. After this the efficiency aspect of the database is described in further detail. Finally, we will look at which data our application needs and where/how this data is stored.

2.5.1. Input features

[redacted]

2.5.2. bunq database

[redacted]

[redacted]

[redacted]

2.5.3. Efficiency solutions

As stated before, there is a need for our application to classify the transaction data in real-time, this is why efficiency is a key aspect. The threshold given by bunq is to keep the time needed to classify one transaction under 0.1 second, to ensure the 'real-time feel' of the application. In this section we will describe different techniques that we can use to achieve this goal.

Directly querying the database The least complex solution is querying the existing bunq database for every feature that is needed. This means that all the needed computations are executed during the classification of a transaction. Although this technique is straight forward to implement, it will result in a large amount of queries on the database per transaction. Also, the computation of the complex features will most likely take too much time to still pass the threshold. This is why we do not think that this solution caters all our needs.

Data store Besides the method described above, the time needed to gather the needed features can also be optimized by looking at the structure of the database itself. In section 2.5.1, we already discussed the problems of complex features.

There is a solution to this problem; using a data store. A data store is a database which is used to store integrated data from other databases. What this means in our application is that the data store contains features which would take to much time to calculate for each transaction. An example is a feature like the number of transactions during the last week for a certain user. This feature would occupy a lot of time if it needs to be calculated for each transaction. That is why this value could be stored in a data store. The process described here is called data denormalization. Data denormalization describes the process of improving read performance of the database, by adding copies of (joined or ordered) data to the database. Although data denormalization increases the complexity of the system, studies have shown that it can enhance query performance when it's deployed with a proper understanding of the needed features [24]. If we want to use a data store, we have to expand the ORM which is currently available to allow the application to gather data from the data store.

ETL Filling the data store with the needed features is done by a process called Extract, Transform, Load (ETL) [31]. Within ETL, the data is first extracted from the data sources, in our application these are the databases which store the tables. After the extraction, the data is transformed. During this transformation, rules and functions are applied to the extracted data in order to transform it to the needed format for the end target. Some data does not require transformation, it is directly copied to the end target. This data is called 'pass through' data. In our application the transformation functions vary from joining data from different databases, aggregating data, sorting and ordering the data, etc. The last phase of ETL is load. During this phase, the transformed data is loaded into the end target, which is the data store.

Traditionally, the ETL process is run in a periodic schedule, for example every night. This means that the data in the data store is not live data, but data from the previous day. For features like the

age of the user this is not a problem, while it changes only once a year. However, [redacted]. For these features, there is a need for (near) real-time ETL. Real-time ETL is the process were the data store is constantly updated, instead of for example once a day.

Real-time ETL example: Santos and Bernardino There are several possible techniques to achieve real-time ETL, here we will describe the real-time data warehouse loading methodology by Santos and Bernardino [25]. This methodology is used as an example of how real-time ETL could be achieved and it could serve as a base for our ETL system later on during the project. There are already a number of open-source (real-time) ETL frameworks available which can be used (e.g. [28] or [30]), the choice to make our own ETL system or use one of these frameworks will be substantiated later on during the project.

The methodology in the paper [25] is mainly based on the principle that row insertion procedures in tables with few or no contents are performed much faster than these procedures in large tables. In general, it is a known fact that data handling in small tables is much less complex and much faster then in large tables. The methodology focuses mostly on optimizing the loading aspect of ETL and the querying of the data store.

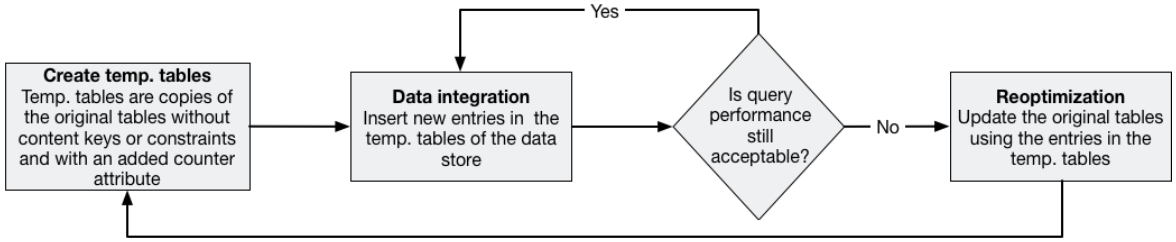


Figure 2.2: Architecture of the proposed methodology (inspired by [25]).

In figure 2.2, the architecture of the proposed methodology is visually explained. The methodology consists of different steps, which we will explain here.

Creation of temporary tables - In this step a temporary table is created for every table in the data store. A temporary table is a structural copy of the original table, which is created empty of contents and with no indexes, keys or other constraints. There is an extra attribute that is added to the temporary table. This attribute is a unique sequential identifier which is related to the insertion of each row in the temporary table.

Santos and Bernardino argue that performance is not the only ETL problem, complexity is a important factor as well. They state that this step guarantees a simple and fast logical an physical support for achieving (near) real-time ETL, while the only change in the structure is adding some simple tables.

Loading procedure - After the ETL procedure has extracted and transformed the needed data, the loading procedure needs to refresh the data store. Within this methodology this is done by executing the following steps every time a new entry is made to the source tables:

1. Extract the needed information from the tables.
2. Transform this information to the needed format.
3. Create a new entry in the temporary table of the data store destination table.
4. Insert the transformed data in this new entry, set the unique sequential identifier to a counter which starts at 1 and auto-increments for every entry insertion.

Santos and Bernardino state that inserting entries in the temporary table is faster then inserting it in the original table. This is partly caused by the fact that the temporary table does not contain indexes, keys or other constraints. So there is no need for time consuming tasks like index updating.

Query adaptation - To take advantage of the use of temporary tables, Santos and Bernardino explain that the queries that are used to get data from the data store need to adapted to the following rule: the FROM clause should join the needed data store tables with their temporary tables, excluding all fixed restriction predicate values from the WHERE clause whenever possible. However, when only the most recent information is required, the query should only use the temporary table. The writers state that this way, their method aids in processing the most recent data, while this data is stored in the temporary table. The small size of these tables makes recent data query processing a lot faster [25].

Reoptimizing - As the temporary tables grow, they lose their advantages. To regain the performance it is necessary to reoptimize the system. The reoptimization is done by updating the original data store tables. The rows of the temporary tables should be aggregated according to the original table's primary key. The row with the highest unique sequential identifier should be added to the original database in the case of duplicate values, because this row represents the most resent entry. Then delete the temporary table content and restore the unique sequential identifier to 1. The time it takes to update the tables is the only time in which the data store is not accessible for the application. So the interval of reoptimizing should by a good balance between temporary table performance and extraction usage.

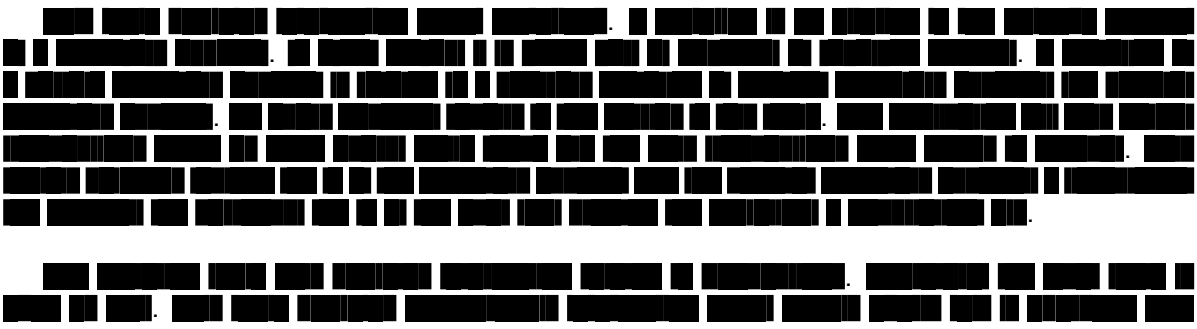
In their paper, Santos and Bernardino show that the methodology works by conducting an experiment on an actual data store. They applied their methodology at various time rates against the execution of various query workloads for data stores with different scale sizes. All these scenarios showed that it is possible to achieve real-time ETL in exchange for an average increase of query execution time.

Features in the data store Until now, we have assumed that we store all the needed features in the data store. However, for a lot of features this is not necessary. Some features are direct copies of attributes from a database (previously mentioned 'pass through' data). Other features require some calculation, but this can be very minimal (like getting age from a date of birth). It is more efficient to just take these features directly from the database, without storing them in a data store first. This is why we will need to look at the execution time of retrieving and calculating each feature. From this data we will determine which features need to be stored in a data store and if there is a need for certain temporary tables as described above. We will begin by using just a few of these features. As the project is progressing we will add more and more features to our application, increasing the need for efficiency.

2.5.4. Used data

In his paper [8], Ali el Hassouni describes the data that is needed for his machine learning model and where he extracted this data. We will use this information combined with our access to the bunq database to describe how the transaction data is stored and structured. We will do this by looking at the database specifics and the relevant tables. Then we will look at the data that needs to be stored in the previously described data store.

Tables The bunq database consists of a large amount of tables, a lot of which are logs of other tables. This is why we will not describe the whole database, but just the parts that apply to transaction monitoring. The features used by the machine learning model are derived from the columns of the tables or combinations of different tables. As stated, the tables are segregated over separate databases.



[redacted]

2.6. Monitoring

[redacted]

2.7. Other aspects

In this chapter we describe the research on the other aspects that are in place and the conclusions we can derive from this research.

2.7.1. Programming language

We need to make an informed decision about which programming language we will use to build our application. At bunq, PHP is used as the main programming language. Although they had a preference

for a PHP application, we were free to choose the language that we thought would fit best.

Our fist choice was Java. The biggest reason to go for Java was that the machine learning model is exported to a POJO (Plain Old Java Object). It would be easy to fit this into our Java application. Furthermore, all the team members had a lot of experience in Java, which was also an advantage to us. However, during the research we found out that the database tables are segregated over separate databases. As stated, bunq uses a custom ORM to access the data from the database. If we were to use Java, we would have to make our own ORM or find a different solution. Because H2O also allows the machine learning model to be accessed via a REST API, we decided that it would be easier to write our application in PHP and use this API. This way we can use the ORM that is currently in place and our application will fit better in the existing code base. If it turns out that the REST API does not meet our requirements, we will have to make our own connection to the exported POJO.

2.7.2. Testing

"Everything that can go wrong, will go wrong sooner or later" - Murphy's law. Many things can go wrong in the process of checking a transaction, often this is because the code simply does not work. bunq wants to be fast with their processes, failing code is only a waste of time and can cost a lot of money. For these reasons good testing is very important.

At bunq, unit testing, API call testing and manual testing are used. Manual testing is done for the parts that can not be tested using PHPUnit, eg. card payment testing. Beside the bunq testing criteria, our application should also meet the Software Improvement Group (SIG) testing criteria. This means we should have a 100% test coverage.

Beside these unit tests and integration tests, which test whether the code works, we also want to do a load test. Load testing is testing under unusually heavy load to determine the range of load the application can handle [26]. We want to know the number of payments which our application can handle without breaking down or causing errors with the existing back-end or database. The reason for this is that the application should be scalable and thus should handle a lot of transactions well.

2.7.3. Dataset

For training the machine learning model, a lot of labeled data is needed. In our case these are transactions that are labeled as fraud or not. Ali el Hassouni labeled a dataset for us that we can use. [redacted] transactions which all have been labeled fraudulent or not. The dataset is real, but anonymized for privacy and security reasons.

2.8. Conclusions

This section contains the conclusions of the research that we conducted. It gives a concise overview of the different parts of our application and it will serve as a base for the creation of our application.

Machine learning model In chapter 2.3 we discussed the differences between deep learning and other machine learning methods. We have also compared different machine learning frameworks that are available at the moment. From these frameworks we have concluded that H2O is the best option for our problem, because it scores good on accuracy, speed, memory consumption, available APIs and extra features such as deploying as REST API and supporting deep learning frameworks for the future. Another a big advantage of H2O is that it is already used for research purposes at bunq.

Back-end connection [redacted]

Gathering data In this chapter, we saw that we can use the ORM that is currently in place to communicate with the bunq MySQL database. Furthermore, we described the importance of efficiency in gathering data from the database. By looking at different features of the machine learning model, we concluded that there was a need to store some of these features in a data store, instead of calculating

them over and over again. We also saw that there is a need for real-time access to some of these features. A real-time ETL technique which makes use of temporary tables was described, this technique is a possible solution to the problem. Later on during this project, we will experiment with this technique and/or already existing ETL frameworks to ensure fast access to the needed data.

Monitoring [redacted]

Other aspects In this chapter we explained that we will use PHP to write our application. This decision was made while it enables us to use the ORM that is currently in place. Furthermore, we will start by using the REST API provided by H2O to use the machine learning model.

The code should be well tested, there are several different techniques to achieve this. The techniques that we are going to use are: unit testing, integration testing and load/scalability testing. The first two are to ensure the functionality of the code, the last is to test if the application can handle a heavy load and thus if the application is scalable.

The data that we will use for the machine learning model is real, but anonymized. It contains around [redacted] transactions.

3

Software development methodology

In this section we will describe the software development methodology by which we worked during the project. We show the structured way in which we tackled the given problem and describe the tools that we used to do so.

3.1. Agile

As we learned over the past years, the agile approach is often the best approach when working on a software development project. Within the agile approach, the project is split up in different parts. We called these parts stages, later on in this report we will describe these stages in further detail.

During these stages we started by looking at what we wanted to do during the stage and how long this would take. The average stage took about one week. We then started designing the architecture of the code by using a whiteboard. When the initial design for that stage was finished we implemented it.

3.2. GitLab

During the implementation of the designed code architectures we made use of GitLab. GitLab is a Git repository manager which equipped us with all the tooling we needed to successfully manage the project.

At the beginning of a stage, after creating the design, we split the bigger architecture in several smaller tasks called issues. Such an issue is labeled to make clear to which part of the product it belongs (for example `MachineLearningModel` or `Monitoring`). An issue is also linked to a milestone, which is a point in the project which we are working towards, in our case a stage. After creation, the issue is put in the 'To Do' product backlog.

Every morning, we started by choosing which issues we wanted to pick up that day, each issue for that day was assigned to a team member. When a team member started to work on an issue, he started by creating a git branch on which he could work. When the issue was fully implemented it needed to be tested as well. We used the `phpdbg` framework to generate test coverage reports in which we could see if all the created code was properly tested.

When an issue was completed, the assigned team member created a merge request for merging his issue branch into the main develop branch. The issue was then labeled with the 'Review' label. Now another team member looked at the merge request. He was responsible for checking the code and test quality, when it was needed he would place comments at certain lines in the code. When the reviewer was satisfied with the newly created code, he merged it into the main develop branch. Now a new issue could be picked up. In figure 3.1 an example screenshot of our issue board is presented.

After a stage was finished, the develop branch was merged into the master branch and a tag with the name of that stage was created. This way we we always had access to a fully working and tested version of our product.

Figure 3.1: Example issue board screenshot

3.3. Experiences and contact with the client and coach

This project is not the first time we work by the agile methodology or with a Git repository manager. This is however the biggest and most extensive software project we faced during our studies, which is why the advantages of the used software development methodology were very clear to us. The agile approach forced us to split up the project in smaller stages, which made it easier to understand all the parts. It also helped a lot during the planning. Besides this, we liked the fact that we could build working prototypes at every stage. This made the product more tangible and enabled us to receive concrete feedback.

Furthermore, GitLab satisfied our needs very well. It enabled us to work together on a complex software system without a lot of overhead. GitLab made it easy for us to review each others code and to divide the issues over the team members.

Over the course of this project, we stayed in good contact with our client, as well as with our TU coach. During this whole project, we have been working at the bunq office in Amsterdam. Being there every day made it possible for us to quickly get answers to all our questions. We also gained very useful insights about working at a young software company.

Every week, we planned a meeting with our TU coach. This enabled us to keep him up to date and we received very valuable tips and feedback during these meetings.

4

Stage one: Minimal viable product

This is the first stage of the design process. During this stage, we will design and implement the minimal viable product (MVP). The MVP is the most basic form of our application. This section will describe the software design of the MVP, which will serve as a base which we will expand during the coming stages.

4.1. Initial software design

This section describes the initial software design which was made before we started implementing the MVP. During the implementation, some changes had to be made. These changes can be found in the next section.

4.1.1. MVP overview

[redacted]

Figure 4.1: Schematic overview of the minimal viable product, ORM and machine learning model

[redacted]

4.1.2. Job controller

[redacted]

4.1.3. Machine learning worker

[redacted]

[= *
]

[redacted]

[= *
]

[redacted]

[= *
]

[redacted]

[= *
]

[redacted]

4.2. Problems during implementation

During the implementation of the initial design which is described above, we encountered several problems which led to redesigns of our code structures. These redesigns will be described in this section.

Figure 4.2: Schematic overview of the redesigned minimal viable product

[= *
]

4.3. Used design patterns

5

Stage two: Filter rules, monitoring and training

At the start of this stage, the MVP is fully designed, implemented, tested and working. It is now time to implement three new parts of the system, which are filtering, monitoring and training. These parts will be explained in further detail in this section.

Figure 5.1 shows a schematic overview of the design of the application, including the initial design of the new parts that will be added during this stage. This figure is used to illustrate the processes that will be described during this section.

Figure 5.1: Schematic overview of the initial stage 2 application design

5.1. Filter rules

[redacted]

5.1.1. Initial software design

[redacted]

5.1.2. Problems during implementation

This section will the describe the changes that we made to the initial software design during it's implementation.

Transaction types [redacted]

[redacted]

[redacted]

[redacted]

Figure 5.2: Schematic overview of the redesigned job controller

Storage

Figure 5.3: Schematic overview of the redesigned stage 2 application

[redacted]

[= *
]

[redacted]

5.1.3. Used design patterns

[redacted]

5.2. Monitoring

When an application is added to a system as big as the bunq back-end, it is important to track it's performance. The goal for the transaction monitoring application is to reduce the number of false positive hits. To help bunq in getting insights about this goal, monitoring is used.

Monitoring covers two areas: application processes and analyzing. The running application processes need to be monitored to ensure that the application is functioning as it should. The data gathered can be used for analyzing different aspects of transaction monitoring: performance of machine-learning models, performance of rules, etc.

5.2.1. Initial Design

This section will describe the initial design of the monitoring application.

Software choices [redacted]

Metrics [redacted]

System design [redacted]

5.2.2. Problems during implementation

This section will describes the problems which we encountered while implementing the above described initial design.

Graphite's limitations Not knowing a lot about monitoring and time serie databases (TSDB), we assumed that all the metrics could simply be put into Graphite. For DevOps related metrics that are time based this was no problem, but for more analytic ones Graphite wasn't suited. Graphite is solely a TSDB. This means that it expects data to come in on a time interval. Because not all our data is structured this way and because we found out that Graphite is not very easy adaptable, we started analyzing other data sources. Depending on the metric certain data sources will be better then others. Grafana can use different data sources next to each other, but it would be best to use as little data sources as possible to keep it understandable and maintainable. The databases that are currently supported by Grafana are:

- Graphite [11]
- Prometheus [22]
- Elasticsearch [9]
- InfluxDB [14]
- OpenTSDB [20]
- AWS Cloudwatch [5]

MySQL is not yet supported, but support for MySQL is in development. It is already integrated in a pre-release of Grafana. While bunq uses a MySQL database, it could be very efficient to use MySQL as data source.

In choosing the data source, we look for efficiency and simpleness. AWS Cloudwatch and OpenTSDB are disregarded almost directly. ASW cloudwatch is coupled to AWS Cloud Resources and applications that run on AWS. This is not used at bunq so not applicable. OpenTSDB runs via HBase. A distributed environment that can give a lot of overhead when self hosted, so this is also not applicable. The next data source to take of the list is InfluxDB. Although the systems was easy to install and use, it came with a restricted set of metrics that where mostly related to server monitoring. Next up are Prometheus and Elasticsearch. Prometheus and Elasticsearch are open source and newer databases. Prometheus is,

just like Graphite, solely build as a TSDB. Elasticsearch proved to be rather difficult to set up for our use case.

After researching the data sources, we wanted to try if the MySQL data source in the pre-release of Grafana could be used. We were quickly able to connect the MySQL database to Grafana to show plots. The calls to MySQL are SQL-queries, so there are no systems needed between Grafana and the bunq database. This offers a lot more possibilities and simplicity when creating a dashboard for the monitoring, which is why we decided to use MySQL.

Segregated databases Some metrics need multiple database tables for calculation. As stated, bunq structured their tables as separate databases, for scalability reasons. Grafana can only use one data source per query, so when the data will be split over several databases in the future, this could cause a problem. However, it is very well possible that Grafana will support multiple data sources per query in the future.

Dashboard script A lot of metrics need to be shown in the same ways: per 30 days, per model, per type, per panel etc. This results in 8 different queries per metric, where only the `SELECT` part is a bit different. When a database name changes or when a query is slightly altered, you would have to change around 30 to 40 queries in Grafana. To improve the ease of use, we created a dashboard generation script. Based on metric settings files and the basic queries, this script generates a custom dashboard and uploads it to Grafana.

During the process of implementing, a few more metrics were added to the list: [redacted].

Figure 5.4 shows an example dashboard with the current metrics.

Figure 5.4: Example of transaction monitoring dashboard

5.3. Training

In order to improve the machine learning model over time, training is crucial. During training, the features that were calculated for each transaction are needed again. It is undesirable to recalculate all those features, we already calculated them during the monitoring process.

5.3.1. Initial software design

[redacted]

5.3.2. Problems during implementation

6

Stage three: Fast access to data

During this stage we will look at the different techniques which we can use to get fast access to our data. As we mentioned in section 2.5, simple features like the amount of the transaction can be easily fetched from the database. However, more complex features like the average transaction amount of a certain user over the past 30 days require much more computation time. This is why there is a need for a smarter, faster way to access the complex data. Furthermore, we will look at the data that we send to and receive from the machine learning model.

6.1. Data from the database

During the initial design phase, we looked at different alternatives for fast data access. We will describe the currently applicable alternatives and derive at a conclusion on which technique we will use. Then, we will describe the software design of this technique within our application.

6.1.1. Data gathering alternatives

In the research part of this report, we discussed data storage and ETL. A small recap, a data store is a database which contains analytical data which can be quickly accessed by for example our transaction monitoring application. ETL is the process of filling the data store with data and updating this data. The data store is the place where we store data that is to complex too calculate from the current database at each transaction.

ETL framework There are several open source ETL frameworks available. However when we started to try them out, we found out that the segregated setup of the bunq database prevents us from properly using these frameworks. For example, a lot of frameworks use `JOIN` operators on several tables, while the bunq tables are all stored in different databases, this is not possible. Also most frameworks were focused on retrieving information from different data sources (csv files, databases, etc.), this is not what we need for our application. Furthermore, almost all of the ETL frameworks only let us use a trial version. While we do not have a budget available for this project, we can not use the paid version.

Custom ETL While the current ETL frameworks do not fully cater our needs, we can look at creating our own custom ETL process. Although this would require more work than using an existing framework, it has some clear advantages. We would be able to use the ORM that is currently in place to communicate with the database. Furthermore it enables us to store the data in the exact format in which we need it. The downside is that we need to find a good solution for keeping the data in the data store up to date and remove data that is not longer needed.

OLAP cube An OLAP (online analytical processing) cube is a way to store multi-dimensional data. OLAP cubes are very suitable to store analytical data which needs to be accessed in more then one way. We could for example look at the number of transactions per postal code or per gender.

Several open source OLAP cube frameworks exist. While we work with the constraints that the OLAP cube should be compatible with the existing MySQL database and we can not host our data on

non-bunq servers, a lot of those frameworks are not suitable. The advantage of the OLAP cubes which we tried out is that it is very easy to fetch data. The problem however is that this technique works rather slow. Also it is very difficult to configure a cube, which makes it hard to maintain.

Conclusion After considering the options mentioned above, we conclude that the best approach is to the usage of a data store. The data store principle seems to be a very good fit for our project due to it's simplicity and customizability. To fill the data store, we expect that we have to create our own custom ETL. This way we can use the very solid ORM which is already in place and we can store the data in the exact form that we need. Also, we can implement our ETL system in PHP. This is an advantage because it contributes to the consistency of our application and PHP is already widely used within bunq.

6.1.2. Initial software design

This section describes the initial software design of the mechanism responsible for providing us with fast access to data which is too complex to calculate for every transaction.

Data store While the data store is also a database, we have various different database options to choose from. We started by looking at a SQL (MySQL) database, while they are already widely used by bunq and by our application. However, we quickly found out that due to the dynamic nature of the complex feature storage, SQL was probably not the best choice. Complex features should be easy to add and remove. Futhermore, some of them should be accessible over multiple time dimensions (e.g. transactions per day, week, month, etc.). While the database structure of SQL is very static, we choose to look at NoSQL database alternatives.

After comparing the specifics of the most used NoSQL databases [17], we conclude that MongoDB is the best fit for our data store. MongoDB is very suitable for dynamic data, while data is stored in JSON format. This means that there is no need to predefine the columns, the data structure can be changed over time. Furthermore, queries, indexing and real-time aggregation provide us with fast access to the needed data. Finally, MongoDB is free and open-source.

ETL process Our idea was to design a custom ETL process. This process would be build up out of several cron jobs which ran on set intervals. The ORM that is currently in place at bunq would be used to fill the MongoDB data store. However, during the design phase of the ETL process, our focus pivoted in a different direction. This was caused by a meeting with the bunq CEO, which will be described in the next section.

6.2. From H2O's Steam to our own implementation

This section describes our reasoning behind the decision to replace H2O's Steam with our own implementation.

6.2.1. The old situation

Until now we have been using the Steam service from H2O [13]. This is a runnable jar file, which boots up a simple webserver where it is possible to upload a POJO generated by the H2O framework to compile a war file. This war file can then be used to serve a REST API on which we can make calls to calculate the fraud probability of a given transaction. If a new machine learning model is created that should replace the currently running one, the running REST API server should be killed, a new war file should be created, and this new war file should be used to boot up a new server. Also if we would like to serve multiple models at the same time, we would have to run multiple Steam instances next to each other, running on different ports or servers. These characteristics are not desirable within our application. This is why we decided to move to our own implementation.

6.2.2. Used language and framework

We decided to create our own implementation to host a REST API. Since we receive the machine learning models in the form of a POJO and a jar library, we have to create the API in a JVM language. A natural choice would probably be Java, but since some of us also have prior experience with Kotlin

[15], we have chosen for this language. Kotlin is a statically typed language and, according to our opinion, more concise. Also it has full interoperability with Java. This means that Java code can directly be called from Kotlin code and vice versa. Because of this, it does not matter that the POJO is written in Java.

Next, we chose a framework for creating the API. Since we also already had some experience with Spring Boot [27] we decided to use this. It is also really easy to configure and write API endpoints with Spring Boot with the neat annotations it supports. Controllers are annotated with @Controller and these contain methods which are annotated with @GetMapping(”<enpoint>”) for GET requests and with @PostMapping(”<enpoint>”) for POST requests and so on.

6.2.3. The new situation

Figure 6.1: Global overview of the machine learning model server

7

Stage four: Making the system production ready

The goal of the bachelor project is to do a full software engineering project in a real-world environment, at a real company. During this stage, we found out that our system will actually be integrated in the bunq back-end. This means our system will be used in production. This integration did cause a pivot in our focus, which resulted in a change of the global design of our system. This will be described in this section. This stage will be the last stage of the project.

7.1. Pivoting our focus

On June 1^{th}, we had a meeting with the CEO of bunq, Ali Niknam. Being a TU Delft Computer Science alumnus himself, he wanted to review our code and talk about the next steps of our project. Until then, our project has been made separately from the bunq back-end core. As you can read in the sections above, we were mainly focused on the efficiency side of the project.

During the meeting, Ali made clear that it was very important that our system would be ready to merge into the currently existing bunq back-end. This means, among other things, that we need to use the bunq workflows and code styles. A lot of those code styles were unknown to us until this meeting. The compatibility of our application with the bunq back-end and systems which are already in place became the top priority over efficiency and optimization.

What this means is that the optimization techniques which we used, like the `Beanstalk` queue or the data store, will be taken out of the project for now. We will no longer focus on these future-oriented optimizations, but rather on making the code more bunq compatible and ready to integrate in the bunq back-end.

7.2. Updated software design

This section contains all the changes that we implemented in our software design. These changes only influence the general structure of our application, so most of the code that we have designed and implemented can be reused within the new design. For example, the actual implementations of the features or data builders has remained the same.

7.2.1. Direct database querying instead of data store

In section 2.5 and 6, we described different techniques which can be used to fetch and compute the needed (complex) features. We derived at the conclusion of using a data store. However, our pivoted focus resulted in us not using the MongoDB or any other data store anymore. Instead, we returned to the solution which was also proposed in section 2.5, direct querying of the bunq database. This means that the information needed for every feature is fetched from the database on the go, this also holds for the computation of the features.

7.2.2. Workflows

The bunq application is build up out of a lot of workflows. A workflow is a process which is run within an application. It has a start point after which it executes several tasks or other workflows. The general design of our system is changed into workflows. For example, a `Worker` within our application has become a workflow. Within this workflow, a transaction is classified. As was stated, these changes only influence the general set up, the implementation of the specific components remain the same. Here we will describe the workflows which are present within our transaction monitoring application. In figure 7.1, a schematic representation of our workflows is given.

Figure 7.1: Schematic overview of the system workflow

[redacted]

7.2.3. Storing features for training

In section 5.3, we described the way in which our application saves the calculated features in the database, so that they can later be accessed while training the machine learning model. Within the new design, this method of saving the features in the database is removed. This was done because of the way we were storing these features. Because of the very dynamic nature of the features, it is very hard to store them in a relational database. In our initial implementation, we encoded the features to a JSON string. We removed this implementation because this is not compliant with the bunq coding style.

The solution which we have implemented to replace the previous implementation is a csv log file. Every time a new machine learning model is loaded into our system, a new log file is created. In the log file, the system stores the calculated features for each transaction it processes. Another advantage of using csv files is that they are very easy to use during the training of the machine learning model.

Our first intuition was that writing the feature values to the csv log file would take a lot of time. However, during the testing of the execution times we found out that it only takes our system around 0.0001 seconds. So this barely has any influence of the efficiency of our system.

7.2.4. Mapping in code instead of database

Until now, our application stores mappings (for example `FeatureMapping`) in the database. We have chosen to do this because it allows users to dynamically add and remove features. However, we found out that there is no need for this at the moment. At bunq, such mappings are stored in associative arrays in the code. We will also move our mappings from the database to these associative arrays.

7.2.5. Naming conventions

[redacted]

7.2.6. Shell commands instead of queue

Our initial application used a queue to split the classification jobs over several workers. While optimization is no longer our focus, the queue makes our application more complex then it needs to be. Therefore, the queue is removed from the project. Instead, we will use a daemon to segregate the classification jobs, this way our application is no longer vulnerable for queue failures.

The daemon consists out of two parts, the daemon master and the daemon worker. The daemon master is responsible for creating the jobs and launching a daemon worker to execute these jobs. First the daemon master will look if there are any new mutations. [redacted]. After that it launches a worker via a

shell command to execute this job and classify all the mutations within the batch.

Currently the daemon master waits for the worker to complete it's task, because using one worker is enough to classify each mutation without falling behind. However, when the amount of mutations increases to a level in which the worker does fall behind, the daemon master can launch several other workers. The daemon master launches a worker via a shell command so that it is possible to run several workers at once in the future, without the use of multi-threading.

7.3. Optimizing the execution time

After the new design was implemented, we started testing our application on a test database. Although the test database is smaller than the actual production database, it gave us valuable insights in the efficiency and speed of our application. From these tests, we saw that our system classifies a transaction in around [redacted] seconds. Because this was a long way of from the [redacted] second threshold, we looked at optimizations that could be made without changing the whole set up (e.g. without using a data store).

After a more detailed analysis of the execution times, we found out that around 90% of the time is spent on fetching information from the database. Within this system, [redacted] queries were made to the database. From this data, it became clear that we needed to reduce the amount of queries made to lower the execution time. As we stated before, the biggest subset of features are the aggregated features. These features describe a certain value over different timeframes. [redacted] of the [redacted] queries are made to compute these aggregated features.

To lower this large number, we started by grouping the aggregated features by data type, this if for example the number of a certain type of payment over different timeframes. Instead of running a separate query for all of these features, we then made one query per data type. The computations needed to get the value for a certain timeframe are now executed on the data that we get from the data type query. After we implemented this method at the aggregated features, we also looked at where we could implement it at the normal features, which resulted in another optimization.

The result of these optimization is that our system now makes [redacted] database calls instead of [redacted]. This is a huge improvement, which is also clear when we look at the execution times. Our system now classifies a transaction in [redacted] to [redacted] seconds instead of [redacted]. Although this is still to slow for the initial [redacted] second threshold, it is fast enough to use the system in the production environment. In chapter 9, recommendations are made to further optimize the execution time.

8

Conclusions

Over the course of ten weeks, we were able to successfully complete the transaction monitoring project. The project resulted in a fully working and tested software solution to the given problem. The system will be taken into production at bunq, which is most likely the most progressive bank of the Netherlands.

The used bunq-made machine learning model has shown to be trustworthy and gives a substantial lower amount of false positives then the current system. The system created by us during this project enables bunq to combine this machine learning model and a set of pre-defined rules to classify a transaction as being possibly fraudulent or not, in a production environment. Furthermore, the system gathers and calculates all the needed data in an efficient manner. Next to this, a Grafana dashboard has been implemented which allows bunq to monitor the performance of our system, as well as the performance of the machine learning model and the pre-defined rules.

Of course, there are still enhancements which can be implemented. [redacted]. We have described our recommendations for future work in chapter 9.

A lot of effort was put into ensuring the maintainability of the system. This does not only mean that the code is very well tested, reviewed and documented, but also that it complies with the coding conventions that are currently in place at bunq. This was done to fulfill the clients wish of a system that would fit into the currently existing back-end.

Everything mentioned above was achieved by strictly following the proposed software development methodology. The relevance for bunq and it's users, the use of a lot of real-world data and the integration in a very large and complex existing back-end have made this project a unique and very exiting challenge.

9

Recommendations and ethics

Although we delivered a fully working product at the end of this project, there are of course always enhancements which can be made. We will describe our recommendations for future work in this chapter. Furthermore, we will also dilate upon the ethical side of our project.

9.1. Recommendations

In this section, we will describe our recommendations for future work. These recommendations relate to the improvement of our system given the expanding of bunq and some unimplemented components which were not vital for our application. These components were not implemented due to the short time span of this project.

9.1.1. Specific machine learning models

[redacted]

9.1.2. Data storing

As we described in this report, we researched the possibility to use data stores to store the complex input features for the machine learning model and the pre-defined rules. We already designed a possible set-up for the data store in section 6.1.2. However, we chose to focus on making our system production ready instead of implementing this design. Although we already greatly optimized the time it takes to calculate the features, we still think a data store would result in an even bigger optimization.

9.1.3. Load tests

During the project, we extensively unit and integration tested our system. The next testing step is performing load tests, to see how many transactions our application can handle in a certain time frame. Sadly, we did not have the time to perform these tests ourselves. For systems that are built to consistently run all the time, it is important to find out what the boundaries are.

9.1.4. Use the generated data

Our system is generating and storing a lot of data about the classification of a transaction. It is also showing statistics about these classifications in the Grafana dashboard. Future work could focus on

using this data to gather new insights about fraud. From this, new machine learning features and rules could be derived.

9.2. Ethics

The system that we actually implemented does not have a big ethical aspect. However, the input for the machine learning model and the pre-defined rules which is gathered and calculated within our application does. These ethical issues will be described in this section.

9.2.1. User data

The transaction monitoring system is a transaction based classifier. This means that the system uses information about a transaction to flag a transaction. Because we are flagging transactions and not people, it would seem that there is no ethical side to this system. However, the system is using information about the owner of the transaction during classifying. This does cause an ethical question. Is it permissible that the transaction monitoring system uses all the information available about a user? Should it for example take into account that a certain user has committed fraud three years ago, even when that user was already punished for that crime? Although we are not directly involved in these decisions, because we do not design the input features, it is still wise to think about the implications of some of these decisions. Our system does not distinguish different input features, so in theory it could be used to run a very biased machine learning model. This is why we strongly recommend to think about the ethical implications before using new input features.

9.2.2. EU General Data Protection Regulation

In 2016, the General Data Protection Regulation (GDPR) [10] was signed into action. A part of this regulation is the so called ‘right to explanation’. This describes the right of a consumer to receive an explanation about why a certain decision was made by for instance a company. In this project’s environment, this means that a user has the right to know why his transaction was flagged or not. With the current system, this is not a problem. We can show why a transaction was flagged. However, as we already described in section 2.3.1, it is possible to include deep learning in the transaction monitoring process [redacted]. For these deep learning techniques, it is often not possible to say why it flagged a certain transaction. Only relying on such a system would make it very difficult to uphold the right to explanation. This is why we would advise to be very cautious in working with these techniques.

A

Project plan

This project plan gives an oversight of the project assignment and setup. Within this plan, we will describe the environment in which this project takes place, as well as the goals and requirements for the final product. Furthermore, we will describe which methods, tools and techniques will be used during this project and which quality assurance measures need to be taken into account.

A.1. Project assignment

This section will describe the assignment given to us by bunq. The first section describes the environment in which the project is done. The second section describes the goals of the project, after which the third section defines the assignment itself. Finally, the product that we will create and it's requirements are described in the fourth section.

A.1.1. Project environment

bunq is a software development company with a banking license. The company is focused on making banking easy and social again, it does this from an IT perspective and using the latest technology.

As a bank you are responsible for the security of your costumer's money. Therefore, bunq is legally required to monitor the transactions and scan these for possible fraudulent transactions. A transaction is the transfer of money from one bank account to another. These transactions can be fraudulent or non-fraudulent. [redacted].

Following the responsibility for the security of the customer's money, a bank can not afford to have false negatives. This is why the current system is very careful in the labeling of transactions. This results in a lot of false positives, which all need to be checked by the transaction monitoring department. This is a costly and time consuming task.

Being a software development company, bunq wants to improve their transaction monitoring. The goal of the improvement is to create a new system that classifies transactions into fraudulent and non-fraudulent using machine learning. This should reduce the amount of false positives and thus make the process of transaction monitoring more efficient. This should be done while keeping the false negatives as low, and thereby as trustworthy, as possible. The transactions should be classified as fast as possible. If possible, the transactions should be classified in real-time.

At this moment a PhD-student, Ali el Hassouni, is working on such an improvement in the form of a machine learning model which classifies transactions.

A.1.2. Project goal

Although a machine learning model is being build, there is not yet a system in place to take the model into production. There is a need for such a system which allows a machine learning model to work with the bunq back-end. Furthermore, this system should be able to accept updated models, show statistics about the used model and compare the performance of these models. As stated above, efficiency is a very important factor for the system.

A.1.3. Assignment specification

We will create a software product which fulfills the above described goal. To do this, we will start with researching the problem and the methods needed to connect the model to the existing back-end of bunq.

It is known that the machine learning model which is currently in place is too slow, mainly because it uses a lot of input variables. This is why we will start by using a very simple model. However, in a later phase, part of the assignment is to find a better model or alter the existing one to make it faster. This process will be guided by Ali el Hassouni.

A.1.4. Final product requirements

The final product will be an application that meets all the described requirements and demands as specified below. These requirement and demands are specified using the MoSCoW model [7].

Must have

- The system must create a connection between a machine learning model and the bunq back-end, to make the transaction monitoring production ready.
- The system must allow models to be updated.
- The system must show statistics about the performance of the used model.
- The system must be able to compare the performance of different models.
- The system must work efficiently, the threshold given by bunq is to keep the classification of a transaction under 0.1 second. This ensures that the system can be called real-time.
- The system's performance must be monitored.
- The system must work on BSD systems.
- The system must meet the quality assurance measures which are stated later in this report.

Should have

- [redacted].
- The current machine learning model should be altered to/replaced by a more efficient model.
- The used machine learning model should reduce the amount of false positives, without increasing the amount of false negatives.

Could have

- There could be a function to train the model with new data, without replacing the model.
- There could be an overview of all the used models and their performances.
- There could be a feature to run several models at the same time.

Won't have

- We will not make our own machine learning model, but use an existing framework.

A.2. Project Setup

To run a project there a multiple aspects to take into consideration. This section describes which methods, tools and techniques are used, it gives a global planning and states information about the contracts with bunq.

A.2.1. Methods

During the project, we will make use of SCRUM [29] to regulate the development. While we do not have a lot of time available to deliver a working product, we will work in weekly sprints. During these weekly sprints we split and prioritize the product backlog as much as possible, which will enable us to combine these small parts in a 'better' product, every week. Daily stand-ups (in the morning) will help us to check if we are on schedule. At the end of each day we will send a daily status update to our coach from bunq. This status update generally describes what we did that specific day. It helps us to reflect on our own work and it forms a base for the daily standup. Furthermore it gives bunq an insight in what we are doing and if we are still working on schedule. Moreover, we will discuss our progress and results with the client two times a week. This way we can get fast and direct feedback and the client stays up-to-date. We will also have a weekly meeting with our TU coach to make sure that what we are doing is still in line with the course guides.

A.2.2. Tools

bunq provided us all with a MacBook to work on during the project. We were given our own office space to work during the whole project, located at bunq in Amsterdam. We will use notebooks or our MacBook to record meetings, so that we do not miss or forget valuable information or discussions. Furthermore, bunq provided us with all the necessary development software and tools.

A.2.3. Techniques

The product we are going to build will be an application which uses a machine learning model to classify possible fraudulent transactions and show statistics about this model. This results in three parts:

- *Connection application:* an application that works with the bunq back-end and uses a machine learning model to create transaction monitoring classifications.
- *Dashboard:* a web-application which shows statistics and comparisons about the performances of the used machine learning models.
- *Machine learning model:* the altered/replaced machine learning model which is used by the connection application.

In the research report we will describe the different options and substantiate our choices. To guarantee quality of our code, SIG will look at our code and give us feedback, which we will then use to improve our code.

A.2.4. Planning

We set the project up in different phases: getting started, research, development and finishing. Getting started includes setting up the development environment, getting to know the company and creating the project plan. This phase ends at the start of the second week. During the research phase, we will create the research report. The research report will show how we are going to build the product and why we will make certain choices. This phase should be finished after the second week. After the research phase, the development phase starts. Development includes everything related to the creation of the product: software design, writing code, testing, SIG, etc. Every week we want to deliver a new version of the product. In the middle of the project there will be a mid-term project meeting, together with the client and the TU coach. It is important that everything we do during the development phase is well documented, this information will be used to create the final report. The last phase, finishing, consist of transferring the knowledge of the system and how to use it to bunq, completing the final report and making and giving the final presentation.

The table below shows the planning for the project, the dates shown are indications of the deadline dates.

Date	Description
01-05-2017	Project plan
03-05-2017	First draft research report
08-05-2017	Final research report
15-05-2017	Finish software design
16-05-2017	Start coding
29-05-2017	Mid-term project meeting
01-06-2017	SIG first submission
26-06-2017	SIG second submission
26-06-2017	Submission Final Report
3/4-07-2017	Final Presentation

A.2.5. Contract

bunq is a bank, therefore it interacts with a lot of personal and privacy sensitive data. Because of this, everything we do falls under an NDA we signed. This means that all code and the information discussed in this report stays property of bunq and may not be made publicly available until bunq, or the NDA, allows it. Besides the NDA we have signed a contract and swore the 'Banker's oath'. The contract includes basic internship details. The oath has been made a legal requirement for employees of banks that work/interact with client data, after the 2008 financial crisis.

A.3. Quality assurance

This section describes four topics which need to be taken into account during this project: functionality, maintainability, efficiency and security and privacy. These topics relate to assuring the quality of the final product, as well as the quality of the project itself.

A.3.1. Functionality

As stated in this project plan, the goal of transaction monitoring is to find fraudulent transactions in a large set of transactions. Within our project environment, it is very important that our product functions correctly. This importance is best explained by looking at the consequences of a false positive and a false negative, which determine the correctness of the product (more false positives/negatives cause a lower correctness).

False positive Here a non-fraudulent transaction is labeled as a fraudulent transaction. False positives lead to a large amount of work, because flagged transactions need to be checked manually. Although false positives lead to a higher workload, they do not lead to a product which detects less fraudulent transactions. A product which gives false positives, but no false negatives, would flag every fraudulent transaction.

False negative Here a fraudulent transaction is labeled as a non-fraudulent transaction. A false negative is more harmful then a false positive. This is due to the fact that, while a false positive only leads to a higher workload, a false negative causes a product which detects less fraudulent transactions. While the main goal of the product is to detect fraudulent transactions, false negatives are very harmful to the correctness of the product.

From these descriptions, we conclude that the final product should minimize the amount of false positives and false negatives, where the false negatives are more harmful then the false positives. This is related to the goal to create a product which correctly flags fraudulent transactions in an efficient manner.

A.3.2. Maintainability

The goal of our project is to create a product which will be used by bunq, after our project is finished. As a result, the code that we create should be highly maintainable so it can be used and altered by other employees of bunq. bunq requires that all the code is written compliant to the PSR-2 coding style guide [12]. Furthermore, the code has to be extensively documented and thoroughly tested.

A.3.3. Efficiency

While our product has to deal with very large sets of transactional data, efficiency is a key aspect in creating a useful classification of the data. The project goal is to create a real-time application. If our product would not be efficient enough, it would take to much time to classify the data and the product would not work real-time. Because we are working with transaction data, the classification loses it's usefulness if it takes too much time.

A.3.4. Security and privacy

bunq is a bank, therefore security and privacy are key aspects of everything they do. During this project we will be working with transaction data of real users, and so will the final product. Of course, the financial data of a user should remain private. To ensure this, our final product should meet bunq's high security standards. We will also take the security and privacy into account while working on this project. For example, this means that we will only work on the MacBook provided to us by bunq and we will only store the data on machines owned by bunq. Furthermore we will only use anonymized data to test our product.

B

Software Improvement Group

Over the course of this project, we had to send our code to the Software Improvement Group (SIG) two times. After the first submission, we received feedback which we then integrated before our final code submission. This chapter contains SIG's feedback, as well as our response to this feedback.

B.1. First feedback

One June 12^{th}, we received the following feedback from SIG.

> De code van het systeem scoort 3,5 ster op ons onderhoudbaarheidsmodel, wat betekent dat de code gemiddeld onderhoudbaar is. De hoogste score is niet behaald door een lagere scores voor Duplication.
>
> Voor Duplication wordt er gekeken naar het percentage van de code welke redundant is, oftewel de code die meerdere keren in het systeem voorkomt en in principe verwijderd zou kunnen worden. Vanuit het oogpunt van onderhoudbaarheid is het wenselijk om een laag percentage redundantie te hebben omdat aanpassingen aan deze stukken code doorgaans op meerdere plaatsen moet gebeuren.
>
> In jullie geval zit er bijvoorbeeld duplicatie tussen [redacted].
> Deze classes implementeren allebei de interface genaamd `Feature`, dus dit duplicaat wijst op een architectuurprobleem. Het is hier beter om een nieuwe abstract class te introduceren voor het gedeelde gedrag, om te voorkomen dat je later meerdere kopieën van de code parallel moet onderhouden. De keerzijde hiervan is dat je een vrij diepe inheritance-keten krijgt (eerst de interface `Feature`, dan de nieuwe abstract class, dan [redacted], en dan pas de daadwerkelijke implementatie). In dit geval is deze oplossing desondanks beter. De hoeveelheid code in jullie systeem is nu nog vrij klein, en als er nu al duplicatie tussen die twee classes zit is het zeer aannemelijk dat de hoeveelheid duplicatie later in de levensloop nog verder gaat groeien.
>
> De aanwezigheid van test-code is in ieder geval veelbelovend, hopelijk zal het volume van de test-code ook groeien op het moment dat er nieuwe functionaliteit toegevoegd wordt.
>
> Over het algemeen scoort de code bovengemiddeld, hopelijk lukt het om dit niveau te behouden tijdens de rest van de ontwikkelfase.

This is our translation of the feedback to English.

> The code of the system scores 3.5 stars on our maintenance model, which means that the code is average maintainable. The highest score was not achieved because of a low score on Duplication.
>
> Duplication looks at the percentage of code which is redundant, in other words code which is found multiple times in the system and which could be deleted. From a maintenance

> point of view, it is desirable to have a low redundancy percentage because changes to these pieces of code usually need to happen on multiple places.
>
> In your case, there is duplication between [redacted]. These classes both implement the interface named `Feature`, so this duplicate points out a architectural problem. It is better to introduce a new abstract class for the shared behaviour, to prevent several copies of the code which need to be maintained parallel in the future. The downside of this is there you create a pretty deep inheritance chain (first the interface `Feature`, then the new abstract class, then [redacted] and then the actual implementation). However, this solution is better in this case. The amount of code in your system is still pretty comprehensible, if there already exists duplication between those two classes it will be very likely that the amount of duplication will grow in the future.
>
> The presence of test code is promising, the volume of test code will hopefully grow when new functionality is introduced.
>
> In general, the code scores above average, hopefully this level will be retained during the rest of the development stage.

B.2. Our response

First of all, we were very pleased with the feedback we received from SIG. Their report is mostly very positive. The only negative thing that they point out is the existence of code duplication. From their feedback and our own code scans, we saw that code duplication exists within the feature classes. We fixed this by implementing their proposed solution, we created new abstract classes which included functions which were duplicated within features. The features now extend these abstract classes.

As you can read in this report, our application went through a redesign stage. Sadly, this was done right after we send our initial submission to SIG. This is why the code base changed a lot in between the two SIG submissions, which is why we think it will be difficult to compare the two submissions. However, the above mentioned features are still present in the code base. Because this was the only feedback, we were still able to process all of SIG's feedback in the refactored application.

B.3. Second feedback

On June 30^{th}, we received the following final feedback from SIG.

> In de tweede upload zien we dat zowel de omvang van het systeem als de score voor onderhoudbaarheid is gestegen. Deze stijging is deels veroorzaakt door een verbetering op het gebied van Duplication, dat in de feedback op de eerste upload als verbeterpunt werd aangemerkt. Daarnaast hebben jullie los van onze feedback nog een aantal verbeteringen in het gebruik van lange parameter-lijsten doorgevoerd.
>
> Ook is het goed om te zien dat jullie naast nieuwe productiecode ook aandacht hebben besteed aan het schrijven van nieuwe testcode.
>
> Uit deze observaties kunnen we concluderen dat de aanbevelingen van de vorige evaluatie zijn meegenomen in het ontwikkeltraject.

This is our translation of the feedback to English.

> Within the second upload, we see that the size as well as the maintainability score of this system has increased. This increase is partly caused by an improvement on Duplication, which was part of the suggested improvements in the first feedback. Moreover, you have made a number of improvements in the usage of large parameter lists separately from the feedback we have given.
>
> Furthermore, it is good to see that you have focused on writing new test code next to your new production code.
>
> From these observations, we can conclude that the proposed improvements of the first feedback are implemented during the development process.

C

Original project description

The following description was given by bunq on BEPSys.

C.1. Project description

As a young and innovative mobile bank, we made it easy for customers to set up a bank account and make payments. An inescapable part of being a bank is the fact that people try to use their accounts to commit fraud. With our fraud detection and prevention systems we are able to catch these fraudsters. However, these people will keep trying out new strategies to stay under the radar. To always be one step ahead in this cat-and-mouse game we are tweaking and improving our fraud detection and prevention systems by testing on historical data. This process proves to be time consuming and labor intensive.

That's where you come in. You will build an interface that will make it possible to test the performance of custom monitoring rules and machine learning models on the fly using historical data. The interface should communicate with our backend systems and provide statistics and visualizations about the performance of newly defined or tweaked monitoring rules or algorithms.

Possible research questions:

1. How do we ensure this interface is scalable, to accommodate a growing number of historical transactions and to comply with the complexity of custom rules and inputs needed for the algorithms?
2. What are the requirements for such an interface given our backend systems?

C.2. Company description

bunq is not an ordinary bank. Instead of making more money, we want to reinvent money itself with mobile technology. That's why we built our own banking system from scratch, including an app that fits your entire bank in your pocket. And that's just the beginning! We're working non-stop on futuristic payment methods and other innovations to make money and banking as easy, transparent, and fun as possible.

Transaction Monitoring

Presentation date: 03-07-2017

bunq

ENVIRONMENT

Our project took place at a company named bunq, a self-proclaimed 'IT company with a banking license'. bunq is a bank that overthrows the traditional way of banking and focuses on making banking more personal and social. Instead of an unwieldy big company (like most other banks), they are much more agile.

CHALLENGE

Being a bank, bunq deals with transaction fraud on a regular basis. All transactions that are handled by bunq are monitored for cases of fraud by a transaction monitoring system. When this system flags a transaction as being possibly fraudulent, a bunq employee has to manually check this transaction. The problem with the current system is that it proves to be time consuming and labor intensive. This is caused by the fact that there are a lot of transactions which are falsely flagged as possibly fraudulent, these transactions are called false positives. This resulted in a demand for a system which reduced the number of false positives and thereby the time needed to manually check the flagged transactions.

To fulfill this demand, bunq has been working on creating a machine learning model which classifies transactions as fraudulent or not. This machine learning model showed promising results during test runs on historical data. However, it was not yet production ready, because it was very slow and there existed no connection with the existing bunq back-end. Our challenge was to design a system which enables the in production usage of a machine learning model for transaction monitoring.

PRODUCT

During the project, a new transaction monitoring system was designed and implemented. The new system uses a combination of a bunq-made machine learning model and a set of pre-defined rules to flag a transaction as possibly fraudulent or not. The final system implementation consists out of five different components: (1) an incoming transaction system, responsible for noticing new transactions and segregating those over different workers so that they can be classified in parallel, (2) an information gathering system, which efficiently gathers large sets of needed information for the classification, (3) a machine learning model server, which enables fast communication with altering machine learning models, (4) a set of pre-defined rules, which check transactions for indicators of fraud and (5) a dashboard which monitors the performance and statistics of the machine learning model, the pre-defined rules and our system.

The system is fully tested by unit and integration tests. Furthermore, new machine learning models and pre-defined rules can be easily adopted.

RESEARCH

During the research phase, we focussed on how our application would fit into the currently existing bunq systems. Furthermore, we looked into fast ways of accessing data and the monitoring of our application.

PROCESS

Over the course of this project, we have been working at the bunq office in Amsterdam. Being there every day made it possible for us to quickly get answers to all our questions. During the development phase, we followed the agile methodology.

During the project, we found out that our application would be integrated in the bunq back-end. This caused a pivot in our focus to make our application production ready.

OUTLOOK

The final implementation of the system is focused on integrating the system in the currently existing bunq back-end, because the system will actually be used in production.

Recommendations for future work were made to further improve the efficiency and accuracy of the sytem. This incluces data storing, using specific machine learning models and performing load tests.

The relevance for bunq and it's users, the use of a lot of real-world data and the integration in a very large and complex existing back-end have made this project a unique and very exiting challenge.

Contact information

Client: bunq
Ir. Wessel Van
Developer at bunq

TU Coach
Prof. dr. C. Lofi
Web information systems
Software technology

Contact persons
Tom Harting
Sven Popping
Mathieu Post
Daniël Swaab

Project team

Tom Harting

Interests

Data science, machine learning

Contributions

Database research
Feature implementation
General report chapters

Sven Popping

Interests

Machine learning, software engineering

Contributions

Back-end research
Rule implementation
Deamon and qommander

Mathieu Post

Interests

Software development, artificial intelligence

Contributions

Machine learning model:
- research
- server
- communication

Daniël Swaab

Interests

Artificial intelligence, human computer interaction, entrepreneurship

Contributions

Monitoring and statistics:
- research
- design
- implementation

Contributions by all members

Specific report chapters
Creating the final presentation
General code design
General code implementation

The final report for this project can be found at: http://repository.tudelft.nl

Bibliography

[1] Apache. Apache spark mllib, 2017. URL http://spark.apache.org/mllib/. Accessed on 05-05-2017.

[2] B. Baesens, V. Van Vlasselaer, and W. Verbeke. *Fraud Analytics Using Descriptive, Predictive, and Social Network Techniques: A Guide to Data Science for Fraud Detection*. John Wiley & Sons, 2015.

[3] De Nederlandsche Bank. Concept good practices transactiemonitoring bij trustkantoren, October 2016. URL http://www.toezicht.dnb.nl/binaries/50-235823.pdf. Accessed on 02-05-2017.

[4] Rich Caruana and Alexandru Niculescu-Mizil. An empirical comparison of supervised learning algorithms. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, pages 161–168, New York, NY, USA, 2006. ACM. ISBN 1-59593-383-2. doi: 10.1145/1143844.1143865. URL http://doi.acm.org/10.1145/1143844.1143865.

[5] AWS Cloudwatch. Aws cloudwatch. URL https://aws.amazon.com/cloudwatch/. Accessed on 10-05-2017.

[6] David Cournapeau. scikit-learn: machine learning in python, 2017. URL http://scikit-learn.org/. Accessed on 05-05-2017.

[7] Andrew Craddock, Barry Fazackerley, Steve Messenger, Barbara Roberts, and Jennifer Stapleton. *DSDM Atern Handbook*. DSDM Consortium, July 2008.

[8] Ali el Hassouni. Fraud detection using machine learning methods. Master's thesis, Vrije Universiteit Amsterdam, 2017.

[9] Elasticsearch. Elasticsearch. URL https://www.elastic.co. Accessed on 10-05-2017.

[10] EU. Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation). *Official Journal of the European Union*, L119:1–88, May 2016. URL http://eur-lex.europa.eu/legal-content/EN/TXT/?uri=OJ:L:2016:119:TOC.

[11] Graphite. Graphite makes it easy to store and graph metrics. URL https://graphiteapp.org/. Accessed on 04-05-2017.

[12] The PHP Framework Interop Group. Psr-2: Coding style guide. URL http://www.php-fig.org/psr/psr-2/. Accessed on 01-05-2017.

[13] H2O. H2O.ai: predicts fraud and stops it in it's tracks, 2017. URL https://www.h2o.ai/. Accessed on 03-05-2017.

[14] InfluxDB. Influxdb. URL https://www.influxdata.com/. Accessed on 10-05-2017.

[15] JetBrains. Kotlin programming language, 2017. URL https://kotlinlang.org/. Accessed on 03-06-2017.

[16] M. I. Jordan and T. M. Mitchell. Machine learning: Trends, perspectives, and prospects. *Science*, 349(6245):255–260, July 2015.

[17] Kristof Kovacs. Cassandra vs mongodb vs couchdb vs redis vs riak vs hbase vs couchbase vs orientdb vs aerospike vs neo4j vs hypertable vs elasticsearch vs accumulo vs voltdb vs scalaris vs rethinkdb comparison. URL https://kkovacs.eu/cassandra-vs-mongodb-vs-couchdb-vs-redis? Accessed on 24-05-2017.

[18] Tom Kyte. The trouble with triggers. *Oracle magazine*, September 2008. URL http://www.oracle.com/technetwork/issue-archive/2008/08-sep/o58asktom-101055.html.

[19] mlpack. What does mlpack implement?, 2017. URL http://www.mlpack.org/about.html. Accessed on 05-05-2017.

[20] OpenTSDB. Opentsdb. URL http://opentsdb.net/. Accessed on 10-05-2017.

[21] Szilard Pafka. Benchmark for scalability, speed and accuracy of machine learning libraries for classification, 2017. URL https://github.com/szilard/benchm-ml. Accessed on 05-05-2017.

[22] Prometheus. Prometheus db. URL https://prometheus.io/. Accessed on 10-05-2017.

[23] r project. The r project for statistical computing, 2017. URL https://www.r-project.org/. Accessed on 05-05-2017.

[24] G. L. Sanders and Seungkyoon Shin. Denormalization effects on performance of rdbms. In *System Sciences, 2001. Proceedings of the 34th Annual Hawaii International Conference*, pages 9–pp. IEEE, 2001.

[25] Ricardo Jorge Santos and Jorge Bernardino. Real-time data warehouse loading methodology. In *Proceedings of the 2008 International Symposium on Database Engineering and Applications*, IDEAS '08, pages 49–58, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-188-0. doi: 10.1145/1451940.1451949. URL http://doi.acm.org/10.1145/1451940.1451949.

[26] Shivkumar, Hasmukhrai, and Trivedi. Software testing techniques. *International Journal of Advanced Research in Computer Science and Software Engineering*, 2(10), October 2012.

[27] Pivotal Software. Springboot, 2017. URL https://projects.spring.io/spring-boot/. Accessed on 03-06-2017.

[28] Open source sponsored by Stitch. Singer. URL https://www.singer.io. Accessed on 05-05-2017.

[29] Hirotaka Takeuchi and Ikujiro Nonaka. The new new product development game. *Harvard Business Review*, 64:137–146, 1986. URL http://apln-richmond.pbwiki.com/f/New+New+Prod+Devel+Game.pdf.

[30] Christian Thomsen and Torben Bach Pedersen. Pygrametl: A powerful programming framework for extract-transform-load programmers. In *Proceedings of the ACM Twelfth International Workshop on Data Warehousing and OLAP*, DOLAP '09, pages 49–56, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-801-8. doi: 10.1145/1651291.1651301. URL http://doi.acm.org/10.1145/1651291.1651301.

[31] Panos Vassiliadis and Alkis Simitsis. *Extraction, Transformation, and Loading*, pages 1095–1101. Springer US, Boston, MA, 2009. ISBN 978-0-387-39940-9. doi: 10.1007/978-0-387-39940-9_158. URL http://dx.doi.org/10.1007/978-0-387-39940-9_158.