Technische Universiteit Delft

Bachelor End Project

Real-time anomaly detection in critical Rabobank processes

A. R. Breurkes
R. Jongerius
M. M. H. A. Kerkhof
T. D. Westerborg

TUDelft

Bachelor End Project

Real-time anomaly detection in critical Rabobank processes

by

A. R. Breurkes
R. Jongerius
M. M. H. A. Kerkhof
T. D. Westerborg

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

Project duration:	April 23, 2019 – July 5, 2019	
Supervisors:	Mevr. M. Brooijmans,	Rabobank
	Dhr. E. Troostheiden,	Rabobank
	Dr. ir. S. E. Verwer,	TU Delft

This report is confidential and cannot be made public until July 5, 2019.

Abstract

Real-time processing of log data can give valuable insights in the behaviour of systems and processes. The Rabobank is a large bank and has several of these systems and processes, one of such is the QR device registration process. In order to monitor this process, an application was built which detects abnormal behaviour. The application reads in log data, parses it and then analyses it. Analysing is done by fitting distributions of the duration of each step in the process, and checking for anomalies in new incoming data. Finally, a dashboard was made in which the analysis and the distributions are visualised. More information is also available in the dashboard, like a Markov chain of the process and several key performance indicators.

Preface

This report describes the process and results of our Bachelor End Project, the final project for the Bachelor of Computer Science and Engineering at the Technical University of Delft. Over the course of ten weeks, we designed, implemented and delivered an application for the Online Department of the Rabobank. The main goal of this application is to detect and visualise anomalies in the different steps of Rabobank processes and explore the possibilities that data analysis and learning systems could offer.

We would like to thank all the Rabobank employees that helped us in gathering the right data, guided us trough internal processes and introduced us to the right people for their time and efforts. Our particular appreciation goes to Erik Troostheiden, Mireille Brooijmans and Danielle de Graaf, our clients and coaches from the Rabobank.

Additionally, we would like to express our special gratitude towards our supervisor, dr.ir. Sicco Verwer, for his valuable feedback and guidance during the project.

A. R. Breurkes
R. Jongerius
M. M. H. A. Kerkhof
T. D. Westerborg
Delft, July 2019

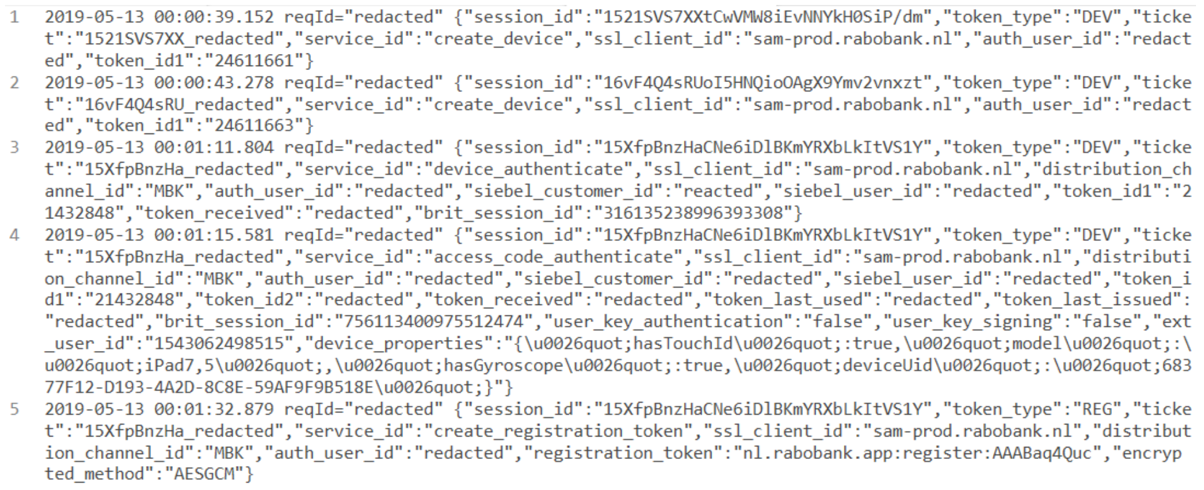
Contents

1 Introduction

The Rabobank is one of the largest banks in the Netherlands with a market share of approximately 35% [12]. Every day, thousands and thousands of its customers use their applications to keep track of their finances or manage their mortgages and investments. All of them rely on the applications to be available, making their continued operation of the utmost importance for the Rabobank.

Keeping all these systems and applications up and running is not an easy feat. Some processes require multiple applications to be online and sometimes there is a dependency from one application or system to another. If one of these applications fails or malfunctions, or there is an outage caused by a third-party, quickly getting an insight as to what happened is essential. Ideally, engineers are alerted the moment there is an indication that something is going wrong in order to deal with the problem swiftly.

To keep track of what all these systems are doing at all times, applications generate logs. Log files are records of what exactly happened at any moment and can store, for example, information on the state of the program, actions the user or system performed, and which errors and exceptions are thrown. Keeping detailed logs for every application means one should be able to tell what happened at all times. An example of a part of a log file can be seen in 1.1. However, the amount of collected log data that is generated is enormous and most of the log data is from applications functioning nominally, which is not the priority when looking for anomalies.



```
2019-05-13 00:00:39.152 reqId="redacted" {"session_id":"1521SVS7XXtCwVMW8iEvNNYkH0SiP/dm","token_type":"DEV","ticke
t":"1521SVS7XX_redacted","service_id":"create_device","ssl_client_id":"sam-prod.rabobank.nl","auth_user_id":"redact
ed","token_id1":"24611661"}
2019-05-13 00:00:43.278 reqId="redacted" {"session_id":"16vF4Q4sRUoI5HNQioOAgX9Ymv2vnxzt","token_type":"DEV","ticke
t":"16vF4Q4sRU_redacted","service_id":"create_device","ssl_client_id":"sam-prod.rabobank.nl","auth_user_id":"redact
ed","token_id1":"24611663"}
2019-05-13 00:01:11.804 reqId="redacted" {"session_id":"15XfpBnzHaCNe6iDlBKmYRXbLkItVS1Y","token_type":"DEV","ticke
t":"15XfpBnzHa_redacted","service_id":"device_authenticate","ssl_client_id":"sam-prod.rabobank.nl","distribution_ch
annel_id":"MBK","auth_user_id":"redacted","siebel_customer_id":"reacted","siebel_user_id":"redacted","token_id1":"2
1432848","token_received":"redacted","brit_session_id":"316135238996393308"}
2019-05-13 00:01:15.581 reqId="redacted" {"session_id":"15XfpBnzHaCNe6iDlBKmYRXbLkItVS1Y","token_type":"DEV","ticke
t":"15XfpBnzHa_redacted","service_id":"access_code_authenticate","ssl_client_id":"sam-prod.rabobank.nl","distributi
on_channel_id":"MBK","auth_user_id":"redacted","siebel_customer_id":"redacted","siebel_user_id":"redacted","token_i
d1":"21432848","token_id2":"redacted","token_received":"redacted","token_last_used":"redacted","token_last_issued":
"redacted","brit_session_id":"756113400975512474","user_key_authentication":"false","user_key_signing":"false","ext
_user_id":"1543062498515","device_properties":"{\u0026quot;hasTouchId\u0026quot;:true,\u0026quot;model\u0026quot;:\
u0026quot;iPad7,5\u0026quot;,\u0026quot;hasGyroscope\u0026quot;:true,\u0026quot;deviceUid\u0026quot;:\u0026quot;683
77F12-D193-4A2D-8C8E-59AF9F9B518E\u0026quot;}"}
2019-05-13 00:01:32.879 reqId="redacted" {"session_id":"15XfpBnzHaCNe6iDlBKmYRXbLkItVS1Y","token_type":"REG","ticke
t":"15XfpBnzHa_redacted","service_id":"create_registration_token","ssl_client_id":"sam-prod.rabobank.nl","distribut
ion_channel_id":"MBK","auth_user_id":"redacted","registration_token":"nl.rabobank.app:register:AAABaq4Quc","encryp
ted_method":"AESGCM"}
```

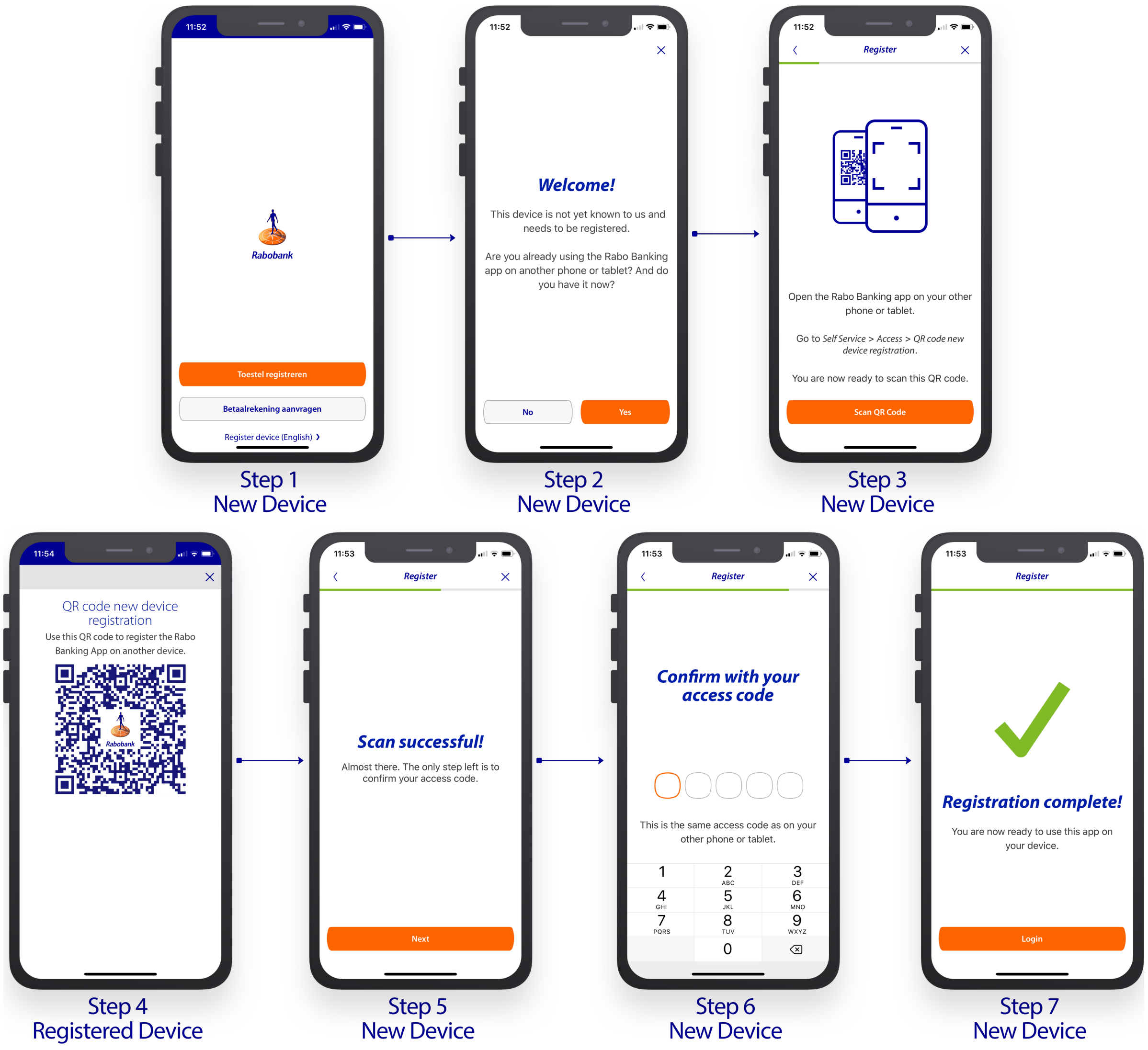
Figure 1.1: An example of a log file from the QR device registration process. Each new line represents the occurrence of a single event. Some fields are redacted due to their sensitivity.

Sifting through all this data to find the events that can signal an anomaly is unfeasible. Storing it would require huge amounts of storage and would make searching through it all slow. Ideally, the analysis of this data happens automatically in real-time, so that situations that could indicate a disturbance in the functioning of applications are spotted as early as possible. This will help to minimise unavailability of systems and along with that, minimise the impact on customers.

1.1. QR Device Registration

One of many processes in which users engage with the Rabobank online environment is the QR-device registration process. In this process, a user is able to register a new device using an already registered device. Registered devices are devices, like phones or tablets, known by the Rabobank which are authorised to perform transactions without the need for the banks dedicated authorisation device, the "Rabo Scanner." Previously, the Rabo Scanner was always necessary to register a device for the first time. As it is one of the main goals of the Rabobank to minimise the dependence on the Rabo Reader, the QR registration process was created.

A visual representation of the process can be seen in 1.2. In short, the QR device registration process is as follows. On a new device, the Rabobank app is installed. On opening it for the first time, the user is asked whether or not the user currently has a device registered for Rabobank online banking (Step 1). If the user chooses yes, the process is started (Step 2 & 3). On the already registered the device, the user chooses the option to generate a QR code (Step 4). Finally, the QR code is scanned with the new device (Step 5) and an access code for the new device is chosen (Step 6), after which the QR registration process is completed successfully (Step 7).


This device is not yet known to us and needs to be registered.

Are you already using the Rabo Banking app on another phone or tablet? And do you have it now?

Open the Rabo Banking app on your other phone or tablet.

Go to *Self Service > Access > QR code new device registration*.

You are now ready to scan this QR code.

Use this QR code to register the Rabo Banking App on another device.

Almost there. The only step left is to confirm your access code.

This is the same access code as on your other phone or tablet.

You are now ready to use this app on your device.

Figure 1.2: Each step of the QR device registration process and the device on which it needs to be performed.

As this process is an important part of Rabobank's effort to decrease dependency on the Rabo Scanner, its availability and performance are of high importance. Therefore, the goal of this project is to develop a tool which analyses this process and its performance. At the moment, the process is initiated approximately 2500 times a day. For each of these processes, the time between each sequential step of the process, i.e. the du-

Process ID	Duration between step 1 & step 2 (ms)
1	430
2	410
3	450
4	500
5	435
6	510
7	350
8	375
9	360
10	480

Table 1.1: Durations between step 1 and step 2 for ten processes in milliseconds.

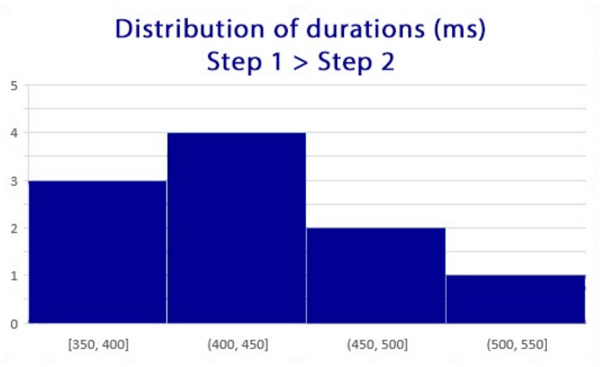


Figure 1.3: Distribution of the durations between step 1 and step 2 from table 1.1.

ration of the step, might differ. For every pair of steps of the process, a distribution can be made of these durations. An example of this can be seen in 1.1 and 1.3.

Over time, new sessions of the process are started and finished, causing these distributions to change as new data is added. When a distribution for one step suddenly changes too much, this could indicate that there is something wrong with the systems responsible for that step, i.e. there is an anomaly. The main task of the application is to detect these anomalies in the QR registration process and show users a warning when one is detected.

1.2. Problem Analysis

The problem is defined as follows: *Given real-time streams of log data, detect anomalies in the data signalling outages or erroneous behaviour.*

To get a better understanding of what this means, there should be a formal definition of the word "anomaly" and a more clear view of what erroneous behaviour they cause. An *anomaly* is a situation that deviates from the expected or normal situation. Where the latter mentioned is crucial for this project, because deviations from what is "normal" will trigger warnings. More on that subject will follow in chapter 3 and 4.

Because anomalies differ from the expected situation, they can cause and indicate all kinds of problems. Take, for example, a situation where the (QR) device registration process is interrupted at the same step many more times than usual in a small amount of time. This could indicate that a service the registration process depends on has stalled. If the development team has a clear indication of this anomaly, they're more likely to be able to identify the cause and solve the problem more quickly.

Another example of an anomaly is a situation where a certain step in the process repeats itself too often. In this case, the Rabobank application will make many more requests than usual to the same end point on the server, which could have serious consequences for the availability of the Rabobank services due to server overloading. However, if the development team knows where this anomaly has occurred, they can narrow their scope in search for the cause of the anomaly. Resulting in quicker repair time for the error.

As a final example, an anomaly could take the form of very long durations between requests. Say the time between step four and five in figure 1.2 normally takes little time, but suddenly it takes twice as much time as usual per request. The newly formed distribution of duration between requests, such as one depicted in figure 1.3, differs from the normally observed distribution. This could indicate stalling services or (newly) introduced bugs in the code. Again, identifying the anomaly can help the development team find the cause of the anomaly more quickly.

In order to detect the anomalies, the software has to adhere to a few requirements. The developed software should be able to visualise distributions of the duration between events as well as other key performance indicators (KPI) generated by the Rabobank QR device registration process. Examples of KPIs are, given a certain time frame, the percentage of successful requests, the total amount of requests, the total amount of occurred errors, etc... The program should learn which distributions and KPI values are normal for specific events, and, using these distributions and KPIs, detect anomalies. The (graphical) user interface should throw a warning indicating anomalies have been detected and of which kind and severity. The focus will initially lay on the QR device registration process, but expandability and scalability of the software to other processes has to be kept in mind. This means that it should also not matter whether data of the QR registration process or any other process is fed to the software, as long as the format of the logging remains the same.

Process details

A successful QR Device Registration Process consists of multiple consecutive events that happen in the background, other than the steps shown in figure 1.2 which shows the process on the graphical user interface side. Ideally, such a process is composed of 6 distinct events, shown in Table 1.2.

New Device	Old Device
Create_device	Device_authenticate
	User_key_authenticate OR Access_code _authenticate
	create_registration_token
validate_registration_token	
apply_registration_token	

Table 1.2: All required events for the QR registration process. After the new device has triggered the `create_device` event, it requires the QR code from the old device to continue. The old device has to trigger the events shown in the right column in order to get to the QR code. After scanning the QR code on the new device, the new device can continue with the last events in the left column.

Table 1.3 shows an ideal registrations process of a new device with the data used for linking. The events of two devices can be linked in step 5 or 6, using the `orig_device_id`. Since every step in the described process has a timestamp, the duration of consecutive events can be calculated.

Step	Event	Session_id	token_id1	orig_device_id
1	Create_device	X	A	-
2	Device_authenticate	Y	B	-
3	User_key_authenticate	Y	B	-
4	create_registration_token	Y	-	-
5	validate_registration_token	X	A	B
6	apply_registration_token	X	A	B

Table 1.3: The events and IDs of a single faultless registration process in order of their occurrence. X and Y are the different unique IDs of the new and old device sessions respectively. A and B represent the unique IDs of the tokens corresponding to the requests of the new and the old device respectively.

1.3. Data Format

The log files provided by the Rabobank have a semi-structured format; A single event typically has a timestamp and a field `reqId`, followed by a JSON object. The timestamp on each log lines can be used to calculate the time between every subsequent event. This information gives the software the possibility of determining the normal situation of durations for every request.

The JSON object contains multiple fields that are of interest. For one, it contains the `session_id` field, used to link the events of one device to one process. This is of particular importance, because it is required to determine the sequence of requests made by a device. It is not always the case that the order of events is the same as depicted in the ideal situation in table 1.3. Furthermore, it is also possible —and inevitable— that other devices start the same process simultaneously. Therefore, the software cannot depend solely on timestamps to determine the sequence of events in the process, and uses the `session_id` to match events in a process.

Second, it contains the fields `token_id1` and —in the case of certain events— `orig_device_id`. These field are used to link events of two seperate devices together. For the QR device registration process, this is of great importance, because the user relies on two separate devices at all times. One of the devices on which they had already registered their account, and another on which they would now like to register their account. In order to extract the same information from subsequent events on two different devices, as for one device, the events must be linked together using the two mentioned IDs. Briefly put, they make information extraction such as described in the last paragraph possible for subsequent events that happened on different

devices.
Listing 1.1 shows an example of a log line with most data redacted.

```
2019-05-13 04:15:37.999 reqId="5djucFavAtXXXXXXX"
{
    "session_id":"redacted",
    "token_type":"DEV",
    "ticket":"redacted",
    "service_id":"access_code_authenticate",
    "ssl_client_id":"sam-prod.rabobank.nl",
    "distribution_channel_id":"MBK",
    "auth_user_id":"redacted",
    "siebel_customer_id":"redacted",
    "siebel_user_id":"redacted",
    "token_id1":"redacted",
    "token_id2":"redacted",
    "token_received":"redacted",
    "token_last_used":"redacted",
    "token_last_issued":"redacted",
    "brit_session_id":"redacted",
    "user_key_authentication":"false",
    "user_key_signing":"false",
    "ext_user_id":"redacted"
}
```

Listing 1.1: An example of a log line of the QR-Registration process, expanded over multiple lines to improve readability. As can be seen, most of the fields of the log lines were redacted for this project. This was done because they contained sensitive information that cannot be made public, on which ground Rabobank ordered the redaction.

In the JSON object, fields like `error_code` and `device_properties` may exist. In case of a present `error_code`, it is interesting to provide the end user of the software, the software developer, with as much additional information as possible. For example with `device_properties`, the developer could find out that a certain error only happens on one type of device or one operating system. This information helps the developer in aiming more efficiently at the cause of the problem.

The `service_id` field can be used by the software to identify each different event in the sequence. If an anomaly is found in the event with a specific `service_id`, rather than in a sequence, this could, for example, indicate that this specific event is causing erroneous behaviour. Another example of its use is determining whether a specific event causes users to end the QR registration process prematurely more often than others, in which case the registration step corresponding to this event might not be user friendly enough.

2 Use Cases & Requirements

To build an application that complies with the customer's wishes, it is important to list what the end product will be used for. This can be done by using so-called 'use cases' which can be further divided into product requirements.

2.1. Use Cases

The use cases listed here give a general overview of what the application will be useful for. Its usefulness is described by scenarios with a description of the solution. Note that, in this report, the application will focus on detecting anomalies in the QR device registration process. However, the application can also be applied to log data of other processes. For the use cases below, the application will be applied to several processes, other than and including the QR device registration process. Also note that below the 'client' is the user of the Rabobank mobile banking app, and the 'developer' is the user of this project's application.

1. **Scenario:** The Rabobank developers have just updated the QR device registration process. They discover that less clients register a new device on a daily basis than before. The developers thought they had eased the process, and thus want to find out why less clients register their new devices.
   Solution: The Rabobank developers can use this project's application to figure out whether there is a smaller number of clients initiating the process or a smaller number of clients finishing the process. For the former case, it could be argued that there is less demand. While for the latter case, the developers can use this project's application to figure out exactly where clients decide to quit registering their device. This information could be used to check whether there is a bug in their app or the process became less user-friendly.

2. **Scenario:** An outage in the Rabobank mobile banking app was reported to the developers. It is known that the app can still communicate with the server. However, after entering the login code, the application shows an empty screen and finally returns a time-out message.
   Solution: The developers use this project's application to quickly identify which HTTP request to the server are taking much longer than expected. They can do so by looking at the distributions in the application. If one distribution starts to deviate from what is normal, the application will display a warning message, pinpointing the step in the process which takes longer. The developers can then fix the issue without having to go through all the log data themselves. Instead, the application has done it for them.

3. **Scenario:** It is noon and everything seems to be operating normally. However, for the last hour, calls have been coming in that Rabobank's clients are unable to use the mobile banking app. Log data shows no errors.
   Solution: As soon as the developers look at the project's application, they see a number of warnings, indicating that the total amount of calls to the server had dropped to zero. This indicates that the mobile banking app has been unable to contact the server. The developers now have an indication on where to start repairing the server.

4. **Scenario:** The developers wonder if the QR device registration process is working smoothly for all users, and use the application to look for strange or abnormal behaviour. Specifically, they want to see how

users are progressing through the steps in the process.
Solution: The developers use the Markov chain visualised in the application to see which transitions happen extremely infrequent. This information gives them a starting point to determine how and why these transition could have taken place.

2.2. MoSCoW

The requirements of the application can be described using the MoSCoW(**M**ust haves, **S**hould haves, **C**ould haves and **W**on't haves) method. More specific use cases are listed per requirement. After analysing the problem, the following MoSCoW list was made, with the client's desires kept in mind. In this list, the following definitions are used:

Event: An individual request/call which expects a single response. Consists of a single log line. Identified using a service ID which links it to a certain step in the process.
Process: A sequence of events with a clear goal, e.g. registering a device using QR-registration.
Session: A sequence of events belonging to a single user with two session IDs, one for each device. I.e. an instance of a process.
User: An engineer at the Rabobank who will be using the application.

Must Haves

1. **Learn individual distributions**
   User Story: As a user, I want to be able to view a distribution of durations for each distinct event in the process.
   Analysis: To meet this requirement, for each event in a process, a distribution of the event's duration has to be fitted.

2. **Anomaly detection in event distribution**
   User Story: As a user, I want to be able to know whether there are anomalies in the distributions.
   Analysis: To meet this requirement, anomalies that diverge from the learned distribution of normal behaviour have to be detected, and shown in some form.

3. **Warnings**
   User Story: As a user, I want to be warned in some way if an anomaly has been detected.
   Analysis: To meet this requirement, warnings have to be implemented, which are subsequently shown to the user when anomalies are detected.

4. **Static GUI**
   User Story: As a user, I want to be able to see a visual representation of the data and information.
   Analysis: To meet this requirement, a static GUI which visualises the distributions of durations of events has to be implemented.

Should haves

1. **Customisable warnings**
   User Story: As a user, I want to be able to customise the threshold for divergence from the norm before I am notified.
   Analysis: To meet this requirement, there has to be a place where the user can modify the threshold, and connect this to the back end.

2. **Interactive GUI**
   User Story: As a user, I want to be able to interact with the GUI.
   Analysis: To meet this requirement, the content displayed has to be dynamic, and based on user input. This could be done by using filters, for example.

3. **Interactive graphs**
   User Story: As a user, I want to be able to interact with the graphs.
   Analysis: To meet this requirement, interactions with the graphs which make interpreting the data easier have to be possible. E.g. zooming in and selecting certain areas of the graph are features which have to be implemented.

4. **Combining data**
   User Story: As a user, I want to be able to compare two distributions.
   Analysis: To meet this requirement, the option to compare two distributions in order to visualise the difference between them has to be implemented.

5. **Visualise session**
   User Story: As a user, I want to track a specific end-user through the process.
   Analysis: To meet this requirement, the option to visualise a single user's steps (through a single session) has to be implemented.

6. **Process agnosticism**
   User Story: As a user, I want to use this application for different processes in the Rabobank.
   Analysis: To meet this requirement, the tool has to be developed with scalability in mind.

7. **Real-time processing of logs**
   User Story: As a user, I want to be able to process data in real-time
   Analysis: To meet this requirement, the application has to be able to handle a real-time stream of data.

Could haves

1. **Data agnosticism**
   User Story: As a user, I want to detect anomalies in other characteristics than durations.
   Analysis: To meet this requirement, the whole program has to be made data agnostic.

2. **Detection of session patterns User Story:** As a user, I want to be able to detect patterns in session timelines.
   Analysis: To meet this requirement, the application has to learn normal behaviour of individual sessions, and recognise any abnormalities in these sessions.

3. **Export to Excel**
   User Story: As a user, I want to be able to save what I am seeing for later.
   Analysis: To meet this requirement, the ability to export the (currently viewed) data to Excel has to be added.

4. **Common factor session**
   User Story: As a user, I want to find a link between abnormalities.
   Analysis: To meet this requirement, the common factor between two different sessions with abnormal durations has to be detected and shown.

Won't haves

1. **Prediction of future anomalies.**
   User Story: As a user, I want to be able to predict when future possible anomalies will take place.
   Analysis: To meet this requirement, the application has to use machine learning to predict the next period of time.

3 Background

The previous chapters introduced the problem and clarified the wishes of the client. To find out how to implement the desired requirements for the product, a dive into literature was required. This chapter only describes the highlights of the research that has been applied in the final product. Some preliminary research was also conducted, where many of the topics were found to be inapplicable. This chapter also serves as background information necessary to understand how the application works.

3.1. Detecting anomalies

Several methods exist to detect anomalies in data. The focus of the project will lie on two main methods of anomaly detection; through the comparison of distributions, and with the help of Markov models. Additionally, some other key interesting factors can be compared, like the total number of errors for each day. Comparing distributions enables the detection of abnormal behaviour in specific transitions from one step in the QR device registration process to another. Whereas the Markov model sheds some light on abnormalities in the entire process, and transitions between states which should not really happen in the first place.

3.1.1. Distributions

This project mainly focuses on the durations of events in a certain process. Incoming data can be used to construct a distribution over a certain quantity of interest, in this case the aforementioned durations. As more and more data comes in, it is possible that the distribution changes. Therefore it is necessary to fit an initial distribution, and to have a model that is capable of adjusting the distribution if need be.

There are several methods to fit a distribution, but it is especially important that the distribution can be calculated in real time for scalability purposes. The question for this chapter is "What method is most feasible to learn the distribution over a quantity of interest given a stream of log data?", while taking into account that the distribution might change over time.

Learning and adapting distributions using windows

Approaches to dealing with change can be divided in two categories[13]. One of which does not take into account whether changes in concept have really occurred, and simply adjusts the distribution at regular intervals. The other of which aims to first detect the changes in concept, and then adjusts the distribution.

Approaches in the first category take weighted examples and use time windows of a predefined size. In this category it is especially important to specify proper window sizes. Small windows could yield good results when concept changes have occurred, while bigger windows are more suitable in stable situations in order to not influence performance significantly. In approaches belonging to the second category, the window sizes are adaptive. Some properties of the data are monitored in order to detect concept change. When a concept change is detected, the window's size decreases in order to adapt better. Otherwise, the window's size is increased in order to improve stability of performance. As this project deals with a relatively simple process, where a concept change is unlikely to happen, the first approach works best and will therefore be utilised. Even if a concept change were to occur, new distributions of the duration of a transition from one step to another would be created. Meaning instead of the existing ones having to adapt to change, brand new distributions would be created.

Comparing the distributions

In order to calculate the difference between the distributions, a way of testing the distance of two distributions is necessary. There are multiple different methods for measuring distance between distributions. The tests which are relevant for this project are described below.

Bhattacharyya distance

One way to compare two distributions is using the Bhattacharyya distance. The Bhattacharyya distance uses the similarly named Bhattacharyya coefficient. This is a measure of the overlap of two statistical distributions [2]. The overlap can indicate the relative closeness of the distributions. To calculate the Bhattacharyya coefficient, integration of the overlap of the two distribution is applied. The Bhattacharyya distance is defined as follows:

$$D_B(p,q) = -\ln(BC(p,q)) \tag{3.1}$$

where

- $BC(p,q) = \sum_{x\in X}\sqrt{p(x)q(x)}$ for discrete probability functions, and
- $BC(p,q) = \int\sqrt{p(x)q(x)}dx$ for continuous probability functions.

In either case, $0 \le BC \le 1$ and $0 \le D_B \le \infty$. What this means, is that every point in the domain is compared by taking the root squared error, sum these, and finally take the negative of the natural log of this sum. An example of two identical distributions is given in Figure 3.1, and two completely different distributions in Figure 3.2. The result of Figure 3.1 is 0, which makes sense as the distance between two identical distributions doesn't exist. Meanwhile, the result of Figure 3.2 is undefined, but tends towards infinity because:

$$\lim_{x\to 0} -ln(x) = \infty$$

This also makes sense, as the distance between any two distributions which have nothing in common is infinite; they are as dissimilar as possible.

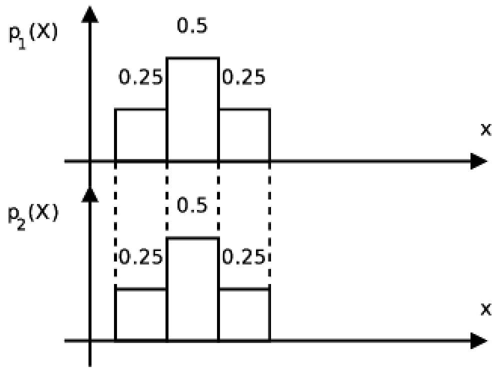


Figure 3.1: Bhattacharyya distance of two similar distributions, calculated as follows: $D_B(p_1,p_2) = -ln(0.25+0.5+0.25) = -ln(1) = 0$.

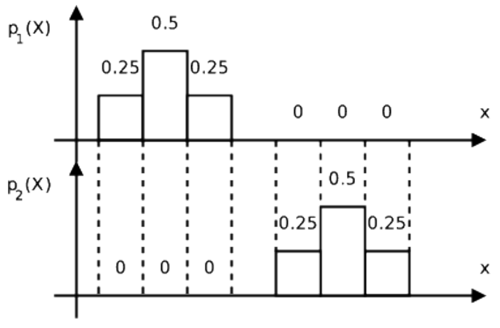


Figure 3.2: Bhattacharyya distance of two different distributions, calculated as follows: $D_B(p_1,p_2) = -ln(0) = undefined$.

As mentioned before, $0 \le D_B \le \infty$. As $0 \le BC \le 1$, the domain of D_B is $(0,1)$. Even though the distance can go up to ∞, even low distance measures can be quite significant, as illustrated in Figure 3.3. This is the result of the logarithmic function in the distance measure. Whether a distribution is actually significantly different depends on the specific situation; when this measure is applied, one still has to decide on their own thresholds from what distance the distributions are significantly different. However as can be seen in Figure 3.3, a distance of 2 already indicated a very different distribution.

The Hellinger distance is closely related to the Bhattacharyya distance, and can also be used to quantify the similarity in two distributions. Furthermore, it can also be derived from the Bhattacharyyan coefficient by using the following formula:

$$H(p,q) = \sqrt{1 - BC(p,q)} \tag{3.2}$$

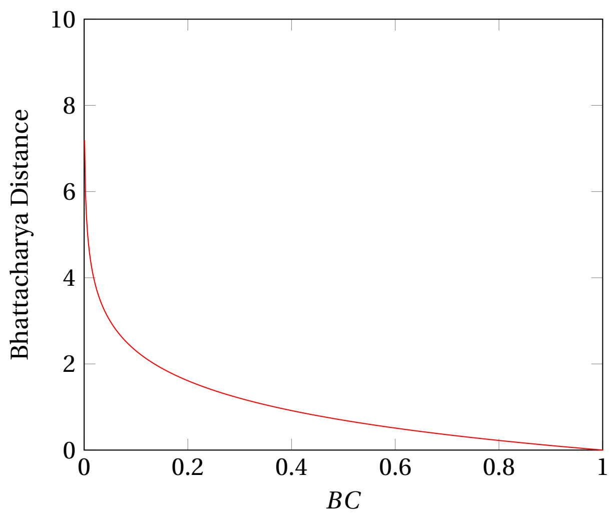


Figure 3.3: The Bhattacharyya distance from extremely dissimilar (left, $BC = 0$) to exactly the same (right, $BC = 1$).

It gives similar results to the Bhattacharyya distance, so implementing only one of these distance measures is enough. The Hellinger distance is formally defined in terms of the Hellinger integral, which in turn is a special case of the Kolmogorov integral. The name Kolmogorov is famous in statistics, as there is another widely used test to test the equality of two probability distributions: the Kolmogorov-Smirnov test.

Kolmogorov–Smirnov test

A third way of checking the distance between two sets of data and the corresponding distributions is the Kolmogorov-Smirnov test (KS-test). The KS-test uses the empirical distribution function of two samples to test their equality. The KS-statistic is given by the following equation:

$$D_{n_1,n_2} = \sup_x \left|F_{1,n_1}(x) - F_{2,n_2}(x)\right| \tag{3.3}$$

where $F_{i,n_i}(x)$ are the empirical distribution functions of the tested sets[17]. The resulting statistic D_{n_1,n_2} is then used to calculate

$$\frac{D_{n_1,n_2}}{s(n)} \tag{3.4}$$

where $s(n) = \sqrt{(n_1 + n_2)/n_1 n_2}$. The resulting value is then checked against the following table to check if the difference found is significant.

α	$\frac{D_{crit}}{s(n)}$
0.10	1.22
0.05	1.36
0.01	1.63
0.005	1.73
0.001	1.95

If for example $D_{n_1,n_2} = 0.2384$ is found and $n_1 = n_2 = 5000$ and we check for a confidence level of 99.9%, we see that $D_{crit} = s(n) * 1.95 = 0.04$. Because $D_{n_1,n_2} > D_{crit}$ with $\alpha = 0.001$, we can reject the hypotheses that the data sets are the same at the 99.9% confidence level [17].

In our application, the KS-test can be used to compare two distributions from the same event but a different time of day. For example, the durations of the last 1000 occurrences from a specific step of the process can be used to create a distribution, and another distribution could be made in the same way one hour later. Then using the KS-test, the difference between these two samples can be calculated, D_{n_1,n_2} in the above example. A certain confidence level can be chosen to calculate a D_{crit} value. If this critical value is smaller than D_{n_1,n_2}, so $D_{n_1,n_2} > D_{crit}$, we can say with the chosen confidence level that the two samples are likely drawn from different underlying distributions. By taking a higher confidence level, D_{crit} will become larger so $D_{n_1,n_2} > D_{crit}$ will only be true if the distance is larger. This allows the distribution to vary over time, as it is

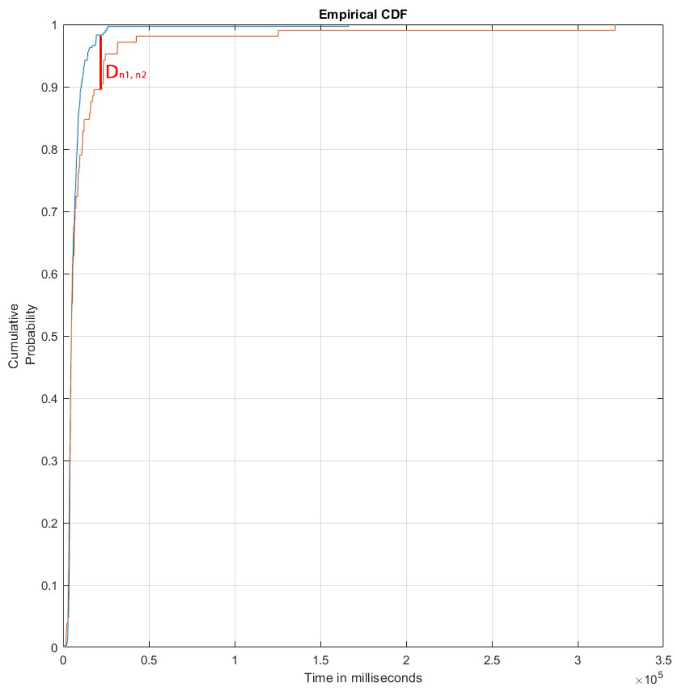


Figure 3.4: A visual example of the KS distance between two empirical cumulative distribution functions. The underlying distributions are from the same step in the QR device registration process, but from different moments. The KS distance between the two functions is equal to D_{n_1,n_2}.

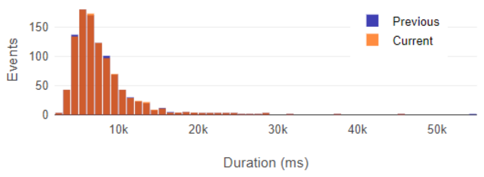


Figure 3.5: Two distributions that are quite similar, there are only small changes visible which is expected. The D_{n_1,n_2} value resulting from the KS-test will be low.

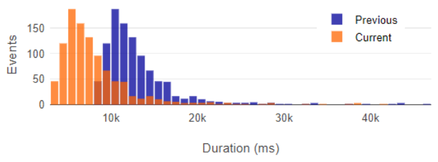


Figure 3.6: Two distributions that are significantly different. The D_{n_1,n_2} value resulting from the KS-test will be higher than the D_{crit}, indicating a potential anomaly.

expected to do, but larger, more significant changes in the distribution will be detected. These sudden, significant changes signal an increase or decrease in the amount of events with a certain duration. As we compare two distributions from one specific step of the process, such a sudden change can be a sign of an anomaly that has something to do with that step, e.g. a failing system or application that that step is depending on.

Justification of Choice

While there are other ways of comparing or measuring a distance between two distributions, we decided to go with the Bhattacharyya distance and Kolmogorov-Smirnov test for several reasons.

First of all, both statistics do not need assumptions about the kind of the distributions that are tested, e.g. if they are normal or gamma distributed. We cannot assume that every distribution we are going to encounter has a specific form, so having tests that are suitable in every situation is necessary.

Secondly, the results of both tests are relatively straightforward. The Bhattacharyya distance is a single number for which a threshold can be set. If the distance is larger than the chosen threshold, we find the distance significant and should show a warning. Multiple significance levels can be used to indicate the severity of the dissimilarity. The same holds for the KS-test. The result of the test can be yes or no for certain confidence levels, e.g. these distributions are similar with 99% confidence.

While implementing more statistical tests is possible, we think that having two different tests to compare distributions is sufficient for our application. There are other ways of finding anomalies in data however, and one of them is through the usage of Markov models.

3.1.2. Markov models

A Markov model is a model used to describe randomly changing systems. The key assumption of these models is that future states solely depend on the current state, and not on previously occurred events. It doesn't have a "memory" of how it was before. One can think of a Markov chain as evolving through discrete "steps" in time.

For example, if a Markov chain model was constructed of a child's behaviour, one might include "playing," "eating," and "sleeping" as states, which together with other behaviours could form a 'state space.' This state space is a list of all possible states the child can reside in. Additionally, a Markov chain also defines the probability of transitioning from one state to any other state, e.g. the chance that a child currently playing will fall asleep in the next five minutes without crying first.

Generally, a Markov chain is used to find the equilibrium in states after a high number of iterations. Using this, one can find the chance of eventually ending up in each of the Markov chain's states. For this project, this application does not make sense as it deals with a process which has a predefined order of steps. Other applications of Markov Models are clustering of data [11] [6], or finding anomalies in a data set [8]. Clustering utilising Markov Models is also not relevant for this project, however finding anomalies using Markov models definitely is.

Finding anomalies using Markov models is achieved by first learning a model of normal behaviour. Once a probabilistic model of said normal behaviour has been learned, it can subsequently be compared to newly observed data in order to detect abnormal behaviour. In essence, a learned model enables the estimation of the probability of each observed event, given the previously observed sequence of events. So, when an event with a low estimated probability does occur, an anomaly warning should be triggered. Likewise, if a never seen before transition happens, this should also trigger a warning. Furthermore, simply showing all transitions which happen with a probability lower than a certain threshold is already valuable information. This project will be dealing with a certain process that has a logical path through which end-user should travel, meaning there are transitions between states which should not happen at all.

Comparing two Markov chains

Just like with distributions, the question of how to detect the differences between two Markov chains arises. As Markov chains are essentially nothing more than axis-labelled matrices, a method to compare the distance between two matrices should suffice to compare two Markov chains.

There are several methods to compare matrices [14]. The first group to be mentioned are the classical cell-by-cell distance measures. This is probably the most intuitive way to compare two matrices, and can be measured in both the absolute and the squared distance. These equations are defined as:

$$D_{AD}(P,Q) = \sum_{i=1}^{n}\sum_{j=1}^{n} \left|p_{ij} - q_{ij}\right|, \tag{3.5}$$

and

$$D_{SD}(P,Q) = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n} \left(p_{ij} - q_{ij}\right)^2} \tag{3.6}$$

For example, take two transition matrices M_1 and M_2:

$$\mathbf{M_1} = \begin{array}{c|ccc|} & A & B & C \\ A & 0.7 & 0.2 & 0.1 \\ B & 0.3 & 0.5 & 0.2 \\ C & 0 & 0 & 0 \end{array} \qquad \mathbf{M_2} = \begin{array}{c|ccc|} & A & B & C \\ A & 0.65 & 0.25 & 0.1 \\ B & 0.3 & 0.55 & 0.25 \\ C & 0 & 0 & 0 \end{array}$$

Simply looking at these transition matrices, one can see they are quite similar. This is confirmed by the two distance tests, as $D_{AD}(M_1, M_2) = 0.2$, meaning there is only a 0.2 absolute difference in probability in the whole system. Also $D_{SD}(M_1, M_2) = 0.1$, meaning once you square all the individual cell distances and take the root the distance is only 0.1. Now, take transition matrix M_3:

$$\mathbf{M_3} = \begin{array}{c|ccc|} & A & B & C \\ A & 0.1 & 0.2 & 0.7 \\ B & 1 & 0 & 0 \\ C & 0 & 0.6 & 0.4 \end{array}$$

After looking at M_3, it is obvious it is a completely different transition matrix to M_1, almost none of the cells are the same. And indeed, the tests confirm this. $D_{AD}(M_1, M_2) = 3.6$, and $D_{SD}(M_1, M_2) = 1.643$. These are much higher values, indicating a much more significant difference between the two.

As shown, these equations work fairly well in measuring the distance between two matrices. However, they do not measure differences in cells relatively. They see 0.95 and 0.93 equally different as 0.04 and 0.02, even though the relative difference is much bigger in the second example. To circumvent this issue, normalised distance measure can be used. The normalised versions of Equation 3.5 and Equation 3.6 are defined as follows:

$$D_{NAD}(P,Q) = \sum_{i=1}^{n}\sum_{j=1}^{n} \frac{|p_{ij} - q_{ij}|}{p_{ij}}; \tag{3.7}$$

and

$$D_{NSD}(P,Q) = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n} \frac{(p_{ij} - q_{ij})^2}{p_{ij}}} \tag{3.8}$$

where the elements p_{ij} cannot be zero. However, this makes a straightforward application to transition matrices rather difficult, since it is quite likely that some transition probabilities can be zero. Also, since $D_{NAD}(P,Q) \neq D_{NAD}(Q,P)$, the difference is no longer symmetric which might lead to inconsistent results, and make it more difficult to set proper thresholds. Luckily, and easy fix for the symmetry is using a combination of both directions of the comparison: $D_{NAD^{symm}} = 0.5 \cdot (D_{NAD}(P,Q) + D_{NAD}(Q,P))$. Still, if the chance for the transition is extremely small, the distance will tend towards infinity. However this might not be a problem, as we can use this property to detect paths which should never happen in the first place. This is possible because we are not dealing with a random process, but with a process with a known sequence of steps when correctly ran through.

In conclusion, two main ways to detect anomalies were explored. Firstly, one can compare two distributions of the durations of events using different statistical tests or measures. Examples of such measures are the Kolmogorov–Smirnov test and the Bhattacharyya distance. Secondly, a Markov model of the process can be constructed, which can subsequently be compared with Markov models generated with newer data.

4 Software Design

This chapter goes into more detail about the design choices made during the project. First the design of the back-end will be discussed, how the actual logic of the application works. After that, the choices made in the front-end will be explained.

4.1. Back end

The back end of the application needs to perform a series of tasks. First of all, data has to be read from a set of text files and parsed to meaningful objects for further processing. After this initial step, analysis has to be performed on the data to determine if there are indicators of an anomaly present. Finally, the data has to be send to the front end and, if an anomaly was found, the results of the analysis should be wrapped in a warning and send to the front end as well.

When designing the back end of an application that will be written in an object oriented language like Java, it is important to keep certain design principles in mind. The best known software design principles are represented by the acronym *SOLID*: the **S**ingle responsibility principle, the **O**pen-closed principle, the **L**iskov-substitution principle, the **I**nterface segregation principle and the **D**ependency inversion principle. These principles are fairly straightforward and were followed where possible during the process of designing the application.

4.1.1. Parser

As stated in section 1.3, the data to be parsed will be semi-structured and partially in JSON format. New data will be handled line by line and split into a timestamp, reqId and a JSON-structured part. After the JSON-structured part is parsed using a JSON-parser, the timestamp will be added. To ensure a scalable program, the Apache Flink framework will be used.

4.1.2. Analysis

Markov chains

As explained in chapter 3, we can use Markov chains to detect anomalies. In essence, a Markov chain is nothing more than a transition matrix. For example, a Markov chain M for a system with 3 states, A, B and C, could look as follows:

$$\mathbf{M} = \begin{array}{c|ccc|} & A & B & C \\ A & 0.7 & 0.2 & 0.1 \\ B & 0.3 & 0.5 & 0.2 \\ C & 0 & 0 & 0 \end{array}$$

One major characteristic from conventional Markov chains is the fact that for each state, all outgoing transitions have to add up to a total chance of 1, i.e. each row should total 1. As can be seen in the example above, this does not hold up for a typical Markov chains used in this project. This decision was made because traditionally Markov chains are used to find an equilibrium of a system, giving a chance for each state to 'finish' there after a large number of steps. However, in this project they are used differently. They are applied to a process with a set correct path through the transitions, including one final state where the user ends. This state does not have any outgoing transitions, as users in this state have succussfully registrated their device. Instead, this project uses Markov chains to detect anomalies, i.e. checking whether two Markov chains from different time periods differ a lot, and contain transitions which should not happen. To achieve this, the rule that each row should total 1 does not need to hold up, they can also be full of zeros.

As Markov chains are essentially transition matrices, they will be stored as a two-dimensional array of doubles. In order to keep track of states, they will be tracked separately in a simple `ArrayList`. Linking the correct state to its corresponding row or column in the transition matrix can be done using the index of the state in the ArrayList.

Statistical distance measures
The in subsection 3.1.1 described tests and distance measures like the Kolmogorov-Smirnov test and Bhattacharyya distance will be implemented based on their mathematical definition. Because they have to be performed with a high frequency, it is important that their implementation is done efficiently.

4.2. Front end

The front-end will be a web-based dashboard. The choice for a web application is largely based on the enormous amount of JavaScript visualisation libraries available. Native user interface design is quite time-consuming, especially in Java. Web development makes things much easier, where the addition of new components or tabs is trivial. Given the limited time frame of the project, a web-based front end is the ideal option. It also means that any device with a somewhat modern web browser will be capable of displaying the application.

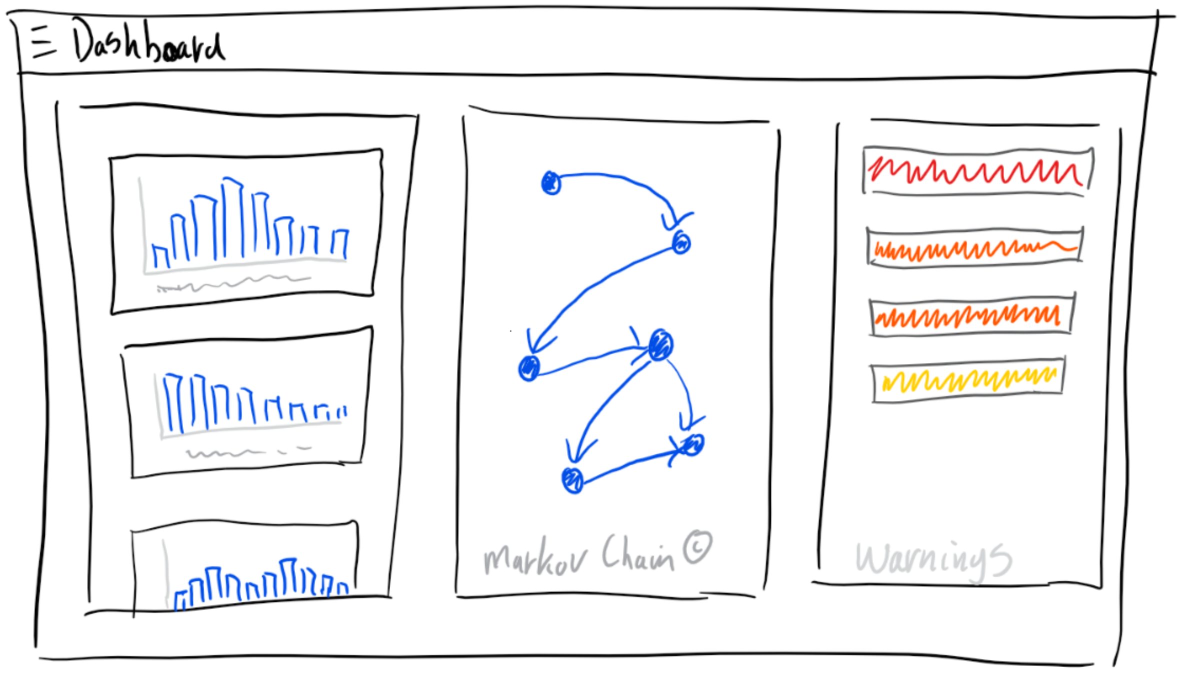


Figure 4.1: First sketch of the dashboard

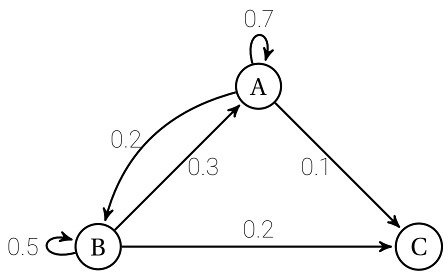
4.2.1. Data Visualisation

There are three main categories of data which will be visualised in the dashboard. These are described below.

Distributions
First of all, the main focus of the application will be the distributions of durations. This will be the main page

which users are initially presented with. Here, all distributions of durations, one for each pair of steps, will be listed. A JavaScript library will be used to generate the actual visualisation of the distributions, most likely in the form of histograms.

Markov Chains
While a Markov chain in mathematics is usually depicted as a set of states and their transition matrix, this is not something which makes sense to the average laymen, especially when they don't even know exactly what a Markov chain is or represents. Therefore, a much more useful approach would be to represent it as a network graph. Such a visual representation of the Markov chain described in subsection 4.1.2 would look as follows:



Implementing such visualisation is quite a task in and of itself, which is why a library will be used to aid with the visualisation. As graphing frameworks based on JavaScript are plentiful, finding one pleasant to work with did not result in any problem. vis.js [15] was chosen .

Key performance indicators
In addition to the distributions of durations and the Markov chain of the process, other useful information can be extracted from the data. An obvious example might be the number of processes which do not result in a successful device registration, i.e. unfinished processes. In order to identify more key performance indicators, extensive discussion with Rabobank will have to be arranged, but the following key performance indicators will be implemented as a start.

- Percentages like the successfully finished processes
- Pie charts that display ratios like the ratio in which error codes occur
- Graphs that display information like the number of errors that occurred per time unit.

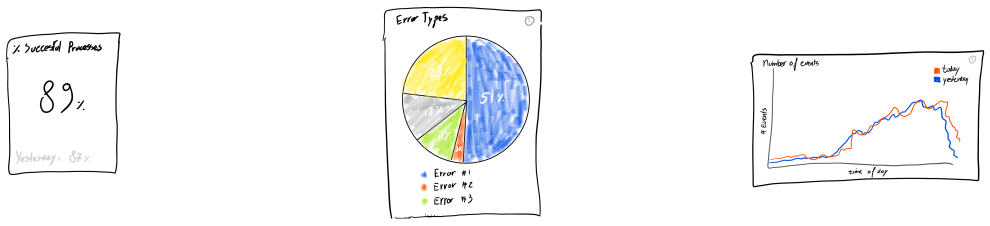


Figure 4.2: First sketches of the different kinds of key performance indicators

Figure 4.2 shows an initial sketch of these KPIs, but future discussion with the Rabobank may lead to adjustments. More on the actually implemented KPIs in subsection 5.1.2.

4.2.2. Warnings

In addition to the visualisation of the data, the user has to be notified if an anomaly is detected. This will be done through warnings. These warnings have to be clear, concise, and be visible at any point in the application. Not all warnings are equal, some might be more severe than others. For this reason, different colours will be used to indicate the significance of a warning. Furthermore, the latest warning has to appear at the top of the warning list, because more recent warnings are more interesting. Lastly, warnings should be dismissible once they have been dealt with, or the Rabobank engineers have been notified.

5 Implementation

After doing research and making an initial design, the actual implementation of the application has to be done. The application consists largely of two parts, the back end and front end. The details on how these parts are implemented will be discussed in section 5.1 and section 5.2.

5.1. Back end

The back-end of the application is built in Java and consists of several key components for it to work properly. In order to make the back-end as future proof and maintainable as possible, the code is well documented and implemented according to renowned techniques such as design patterns. In the following subsections, the implementation of the back-end has been dissected into its key components with explanations on the choices made for the implementation.

5.1.1. StateController

To control and update the known distributions, KPIs and other data, a class named `StateController` is in place. This controller calls the parser and is responsible for 'feeding' new data to and updating the `State`. The described process is of high importance, and thus carries a responsibility with it. Because the coordination of these tasks bears a lot of responsibility and should only be done by one entity, the `StateController` is implemented using a Singleton Pattern; there can only be one `StateController`. This ensures that the coordination done by the `StateController` across the system cannot be interfered by another entity.

State

The `State` class contains all relevant data of the current state of the application for a process. Since only one process is tracked at this time—namely, the QR device registration—, only one instance of `State` is instantiated. The QR registration process involves, as described in section 1.2, multiple consecutive events of which the duration can be calculated. For each subsequent pair of steps, a distribution of durations is stored in a `Distribution`. Besides keeping track of the `Distributions`, the `State` updates and keeps track of the Key Performance Indicators (KPIs) and Markov chains. More about distributions will be discussed in subsection 5.1.2.

Parser

The `Parser` is responsible for parsing text files to a stream of JSON Objects and is implemented using the Apache Flink framework and the Java JSONObject JSON parser. Since data is delivered in batches of a fixed time-window, a `FileFinder` is implemented. The `FileFinder` finds all files in a pre-defined folder and returns files in chronological order. `Parser` also uses a Singleton design pattern, as only one parser should be active at all times.

StreamHandler

To extract information from the parsed data, a class `StreamHandler` is used. As described in section 1.2, a process consists of multiple events on two devices. Since both devices use a different session ID, the duration of successive events cannot be calculated without extra information. Instead, the `StreamHandler` keeps track of events per session ID, and combines events of the two devices when possible. Moreover, the `StreamHandler` creates new data for the KPIs to update from. More information on the data processed by `StreamHandler` can be found in section 1.3.

If the Rabobank decides to replace the current system of reading files of log data with a stream of log data, `StreamHandler` will not require any modification as long as the data stream comes in the same format as deliverred by the `Parser` class.

5.1.2. Analysis

Anomalies are detected in multiple ways, through differences in distributions, differences in Markov chains and analysis of the KPIs. An anomaly is detected when a result of one of several tests is deemed significant. All of these factors are elaborated on below. When an anomaly is detected, a warning will be presented to the user via the front-end.

Distributions

The `Distribution` class represents the distribution of duration between events. A distribution contains the following fields:

- Name : The name of this transition
- Data : The most recent data
- old Distribution : A Java Emperical Distribution representing the data from a previous time window
- new Distribution : A Java Emperical Distribution representing the most recent data

Besides the previously defined fields, all distance measures and other tests to detect anomalies in the distribution are implemented within this class. These include the Kolmogorov-Smirnov test and Bhattacharyya distance as described in subsection 3.1.1 The class also contains update methods that, given new data, updates the most recent data and Emperical Distributions accordingly.

Test Results

For the best result, it could be argued that all tests performed on distributions should be combined to form a concluding report on whether or not a anomaly has been detected, different tests on the same distribution yield different, incomparable results. Therefore, results of the performed tests are handled separately, but in a standarized way. An interface `Result` has been implemented for setting the standard for classes that will handle the test results. This interface implements 3 main methods:

- getSignificance: returns a by the test determined significance of the result
- getDescription: returns an explanation of the result
- getName: returns the name of the test that resulted in the result

To determine a significant result, a class `ResultHandler` is implemented. Based on the weighted average of the significance, a warning is created.

Markov Chains

The Markov chains are built using a `MarkovChain` class. This class is essentially a data wrapper. It contains the transition matrix, based on a 2D array of doubles; a List of the states in order to give the nodes in the visual representation their name, and a size to know the dimensions. It has two constructors, one with the aforementioned fields, and also one with a `transitionCount` matrix which is a 2D array of integers. This is used initially because the data doesn't inherently know the probabilities for all transitions, only the number of times they happen. Using these counts a transition matrix is easily constructed using by dividing the number of transitions from the current state to each other state divided by the total outgoing transitions for the current state. Additionally, some methods which can be used to compare two `MarkovChain` objects have been implemented, returning the distance (absolute or squared) between them. Finally, 2 methods have been implemented which are used to export the `MarkovChains` in a JSON format; one for the states and one for the transitions.

Key performance indicators

Key performance indicators can be created based on the information in the provided log files. Different KPIs require different data to be tracked, but all KPIs have some similarity. Hence an abstract class KPI was created with the following fields:

- Name: The name of this KPI
- Type: The type of this KPI
- Date: The date of which the Data of this KPI is
- Info: Contains some information on the KPI if available

In order to be able to handle all KPIs equally, KPIs should implement a predefined set of methods. These methods are defined as follows:

- updateData: A method that, given a list of new data, updates this KPI
- checkAnomalies: A method that executes all tests implemented within the specific type of KPI and returns the results of those tests

There are four types of KPIs implemented: Pie Charts, Stacked Bar Chart, Percentages and simple Graphs. Examples of KPIs are, given a certain time frame, the percentage of successful requests, the total amount of requests and the total amount of occurred errors.

The `GraphKPI` has its own methods to determine whether an anomaly has occurred. The process of checking for anomalies goes by the following concept. Days are cut into 96 time spots of fifteen minutes. For each time spot, the mean value of the KPI is tracked. After each time spot has passed on a day, the methods check how much the observed value deviates from the mean. If the difference surpasses a certain threshold, the back-end will signal the front-end to display a warning. The threshold is surpassed if, for a particular time spot i, the observed value is:

1. Greater than $\mu_i + 0.75 \cdot \mu_{\{0 \le j \le 4, 28 < j \le 95\}}$, and
2. Greater than $1 + \mu_i^{-0.125}$, with a minimum of 1.33 maximum of 2.

Furthermore, if the observed total amount of requests in a time spot is less than $0.75 \cdot \mu_{\{0 \le j \le 4, 28 < j \le 95\}}$ a warning is also fired.

Where μ is the daily mean, μ_i is the mean value for a time spot i, and $0 \le i \le 95$ (a time spot for every quarter of an hour in a day). Note that the for threshold condition 1, and for the threshold condition on the total amount of requests, the daily mean is calculated excluding observed values between 1:00AM and 7:00AM. The reason for this is that on this time interval the network traffic is the lowest observed daily. Including this interval would have drastic influence on the value of the daily mean and, with that, on the threshold.

This threshold was established after trying out multiple functions with boundaries. From the beginning it was clear that a function was necessary that converges to ∞ for very small numbers and converges to 0.5 for very large numbers. It establishes a steady threshold for events with a lot of occurrences, and a large enough threshold for events with (very) little occurrences. If, for instance, a fixed threshold would have been used, events with little occurrences would fire warnings after measured differences as little as 1. While events with many occurrences would possibly fire warnings after very high measured differences, which is too late or not at all.

5.1.3. API

In order for the front-end to acquire the data necessary for its visualisation, the back-end has a RESTful application program interface (API) in place. RESTful APIs use HTTP requests for data handling (GET, POST, PUT, DELETE, etc...[1]). It connects the information the back end processed from the initial data to certain specified end points. The end points are simply uniform resource identifiers (URI) that process HTTP requests and return proper specified responses.

[1]For documentation on HTTP, see https://httpwg.org/specs/.

APIs can easily be implemented in Java using the Spring framework[2]. It allows for easy implementation of a stand alone application with specified API end points. It can also be built on top of an already existing Java application.

This was particularly useful in the beginning of the project, when there was only a simple Java application that read data, calculated processing times and then wrote this to a text file. The application terminated after processing all available data and the front-end would simply read these files. In this stage of the project — before the application might be too complex to handle this easily — it was important to immediately start working on data agnosticism of the app. In other words, neither frond or back-end should require any knowledge of the data prior to processing it.

The back end required minor modifications; Processed data is now kept in lists instead of written to files, and the application is now run through the Spring framework. Keeping the data in lists allowed for data access with little to no overhead, and the API was able to access it at all times.

Some API specific implementation choices were made:

- Because the API was in place only to serve data to the front end, only GET requests were allowed on the specified URIs.
- Before the front end can acquire data, it should first request all unique keys belonging to the data. This can be done in one GET request.
- Because the keys are unique, every set of data has its own URI on one end point. This is done using the base of the URI for getting data, extended by a variable (`/getDurations/{key}`). Requesting data from an unknown key returns the proper HTTP response.

5.2. Front end

This section shows the progress throughout the project and discusses the visual aspect of the application. Firstly, the choices made in design language and principles are explained. After that, a timeline of the visual evolution the application is shown, elaborating on the progression and changes throughout the different versions.

5.2.1. Design Language

The choice has been made to follow the material design principles[3] for the implementation of the front-end. Material Design is a visual language that incorporates the classic principles of good design with the innovation of technology and science. As one of its core designer said: "Unlike real paper, our digital material can expand and reform intelligently. Material has physical surfaces and edges. Seams and shadows provide meaning about what you can touch." [7]. Since its inception by Google in 2014, material design has quickly become one of the most prevalent design languages used, especially in android and web development [5]. Material design inherits its intuitiveness from the fact that it is based on real paper and ink, which can be layered like real physical material, while maintaining all the added benefits of being a digital design language. For these reasons, and its relative ease of implementation, the choice for material design was an obvious one.

5.2.2. Front-end Timeline

The application has undergone various changes which have been labelled as different version. Here, each version will chronologically be explained shortly with screenshots of the relevant components.

Version 0.1

The first version of the front end can be seen in Figure 5.1. A simple menu has been implemented, but has no functionality as of now. Figure 5.2 is the first visual representation of a distribution. Every graph has a title that states the transition the distribution is about. The x-axis is the duration of the transition from one event to another, the y-axis represent the number of occurrences.

[2]For documentation on the Spring framework, see https://spring.io/.
[3]For documentation and demos, see http://material.io

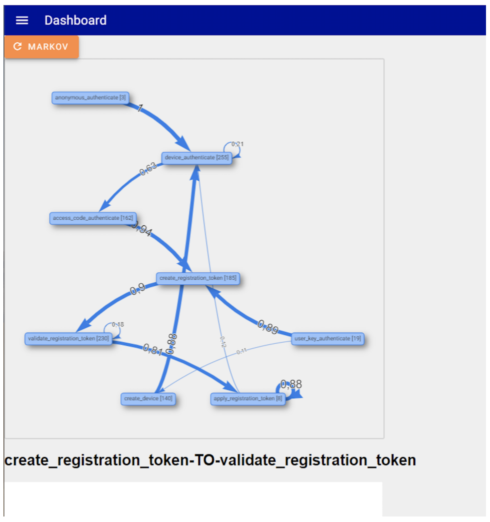


Figure 5.1: Dashboard version 0.1: All the elements like distributions and the Markov chain are in a single column.

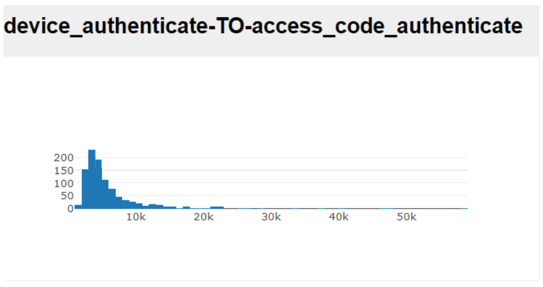


Figure 5.2: Dashboard Version 0.1: Distribution

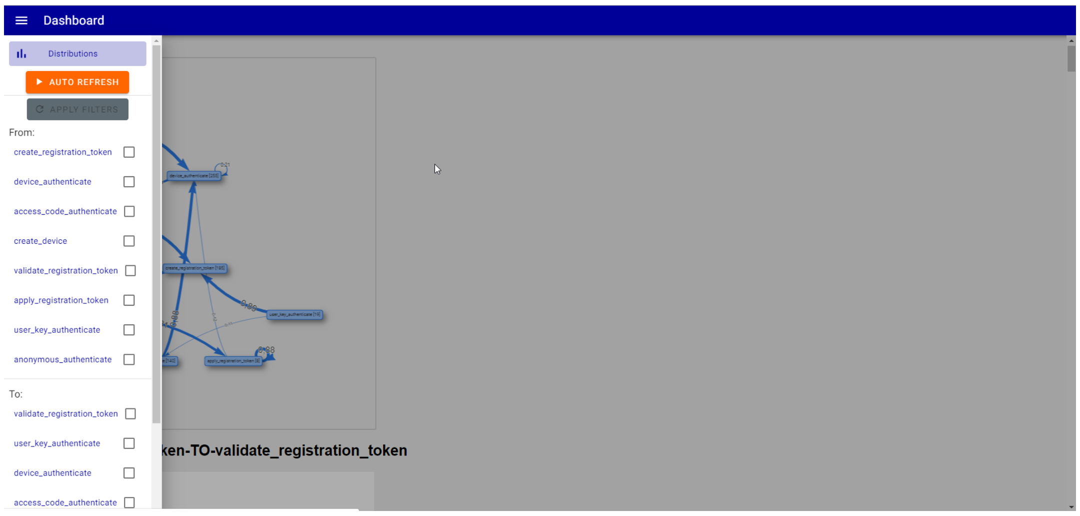


Figure 5.3: The sidebare menu with placeholders for the filters.

Version 0.2

In version 0.2, multiple changes to the front end have been made. As can be seen in figure 5.4, the distributions have a fixed location on the left, while the Markov chain, Warnings and other graphs like 'Unfinished Processes' are placed on the right.

The filters seen in Figure 5.1 are now working and the distributions and KPI-graphs now automatically retrieve new data from the back end at a fixed interval. Version 0.2 also is the first version with Warnings implemented in the front end. Figure 5.5a(a) shows an example of a warning. Warnings have a 'Dismiss' button, which removes the warning from the page.

Also new in Version 0.2 are the 'key performance indicators' (KPIs). Figure 5.5b(b) shows a plot with the number of unfinished processes per time unit which is an important KPI for this process. Distributions now also display the previous data as well as the current data, as cans be seen in Figure 5.4 so that a visual comparison can be easily made.

Besides the addition of new content, improvements have been made to the layout and styling of the dashboard. The layout is now based on cards, giving users a more clear overview of all the content. Furthermore, colours and fonts have been changed to be more in line with the Rabobank's own design language.

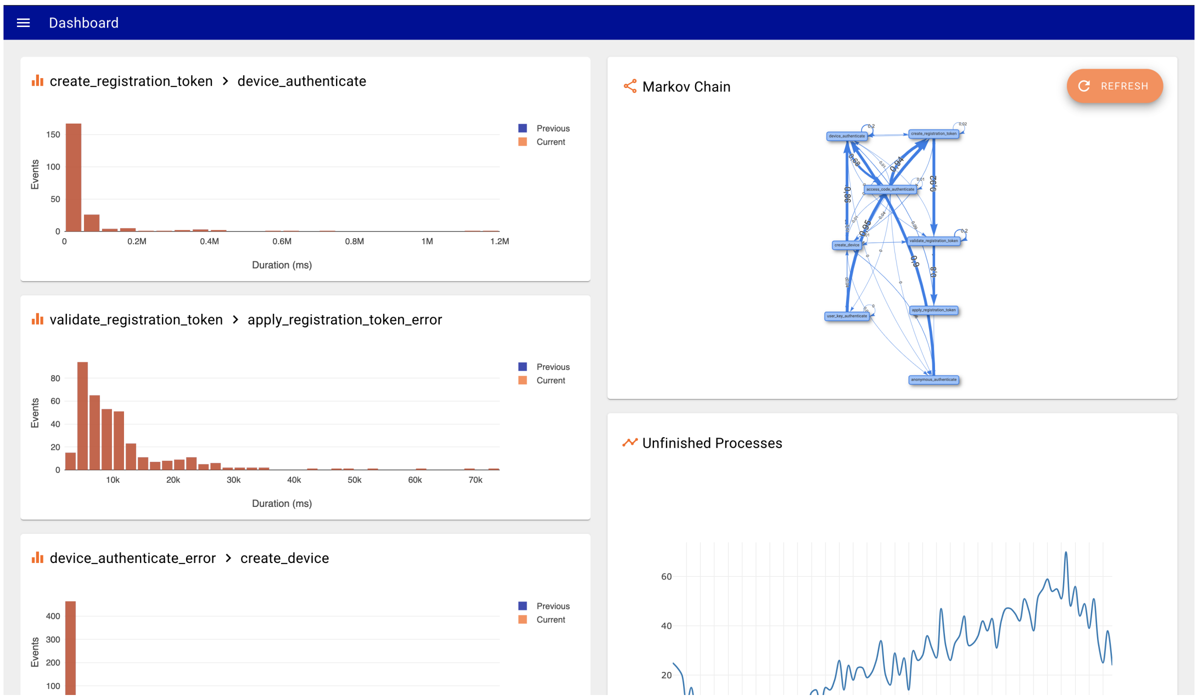
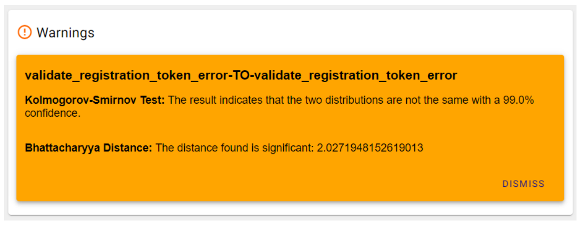

Dashboard
create_registration_token > device_authenticate
Previous
Current
Events
Duration (ms)
validate_registration_token > apply_registration_token_error
device_authenticate_error > create_device
Markov Chain
REFRESH
Unfinished Processes

Figure 5.4: Dashboard Version 0.2

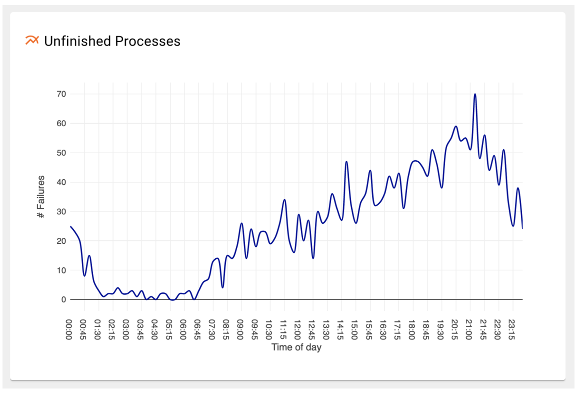

Warnings

validate_registration_token_error-TO-validate_registration_token_error

Kolmogorov-Smirnov Test: The result indicates that the two distributions are not the same with a 99.0% confidence.

Bhattacharyya Distance: The distance found is significant: 2.0271948152619013

(a) Dashboard Version 0.2 : A warning with the results of the tests that were performed

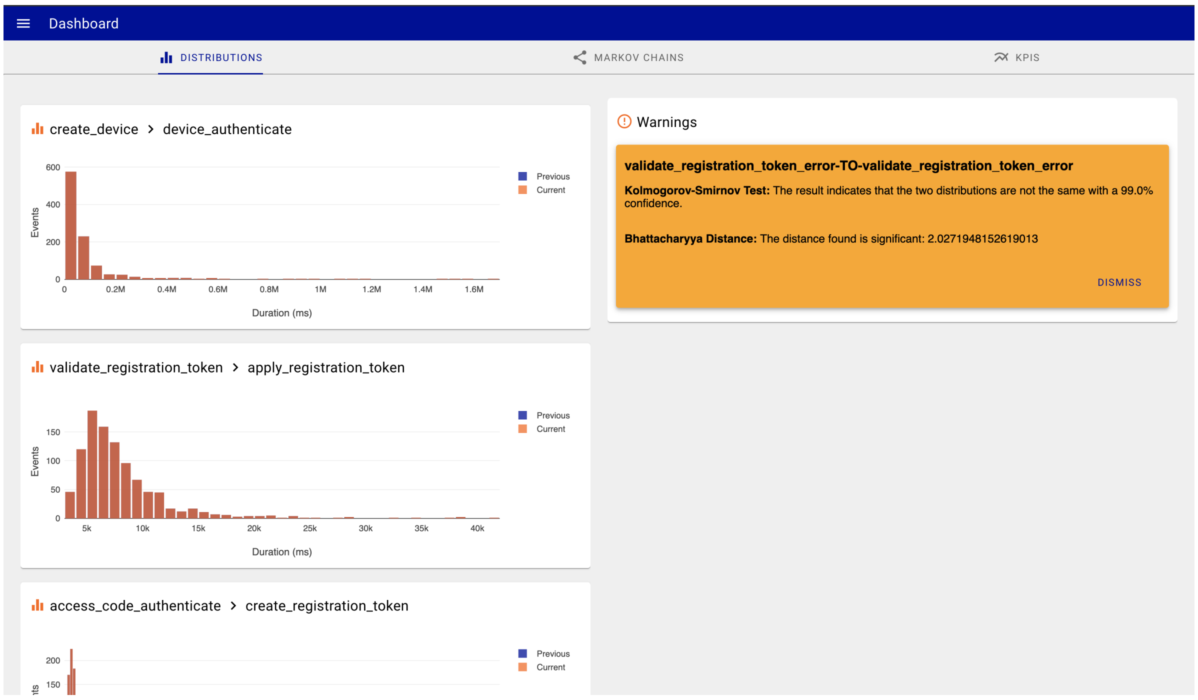


(b) Dashboard Version 0.2 : A KPI plot of the numbers of failures per 15 minutes.

Figure 5.5: A warning and a KPI graph in version 0.2 of the application.

Version 0.3

In the third version of the front end, the layout has been changed to use tabs. For now, it has three tabs: 'Distributions' (Figure 5.6), 'Markov Chain' (Figure 5.7) and 'KPIS' (Figure 5.8). Every page also has the warnings listed on the right side of the page. Warnings can be dismissed using the dismiss button on the bottom right.

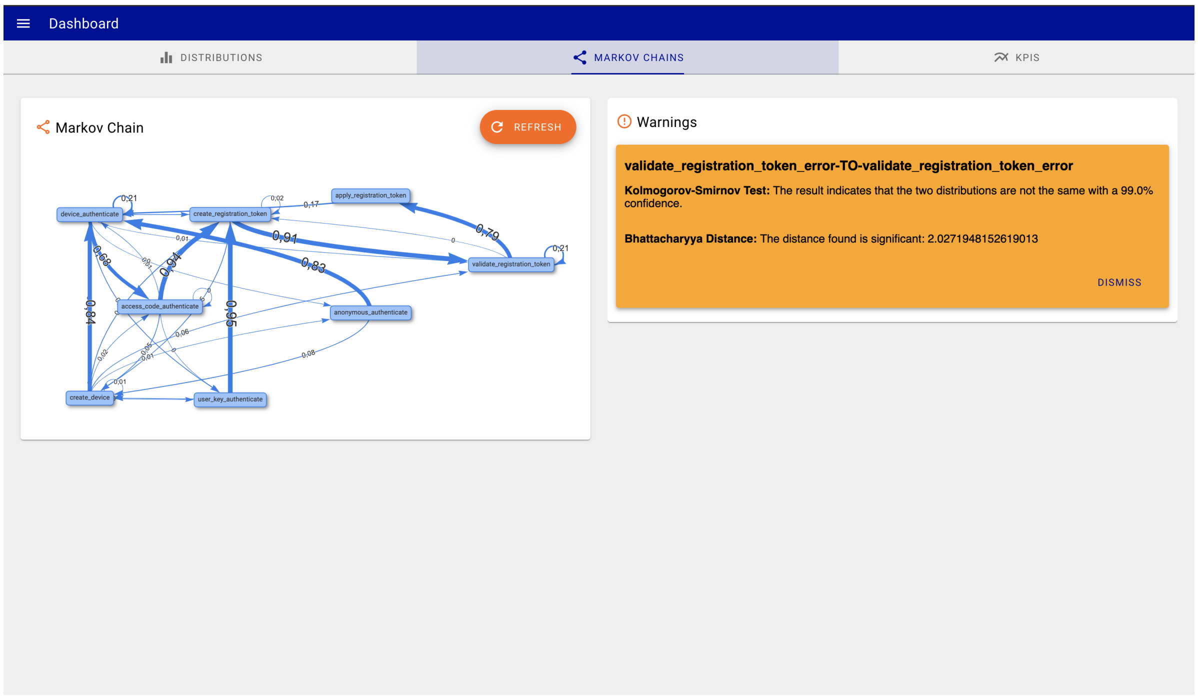

Dashboard
DISTRIBUTIONS
MARKOV CHAINS
KPIS
create_device > device_authenticate
Events
Duration (ms)
Previous
Current
validate_registration_token > apply_registration_token
access_code_authenticate > create_registration_token
Warnings
DISMISS

validate_registration_token_error-TO-validate_registration_token_error

Kolmogorov-Smirnov Test: The result indicates that the two distributions are not the same with a 99.0% confidence.

Bhattacharyya Distance: The distance found is significant: 2.0271948152619013

Figure 5.6: Dashboard Version 0.3: Distribution Tab

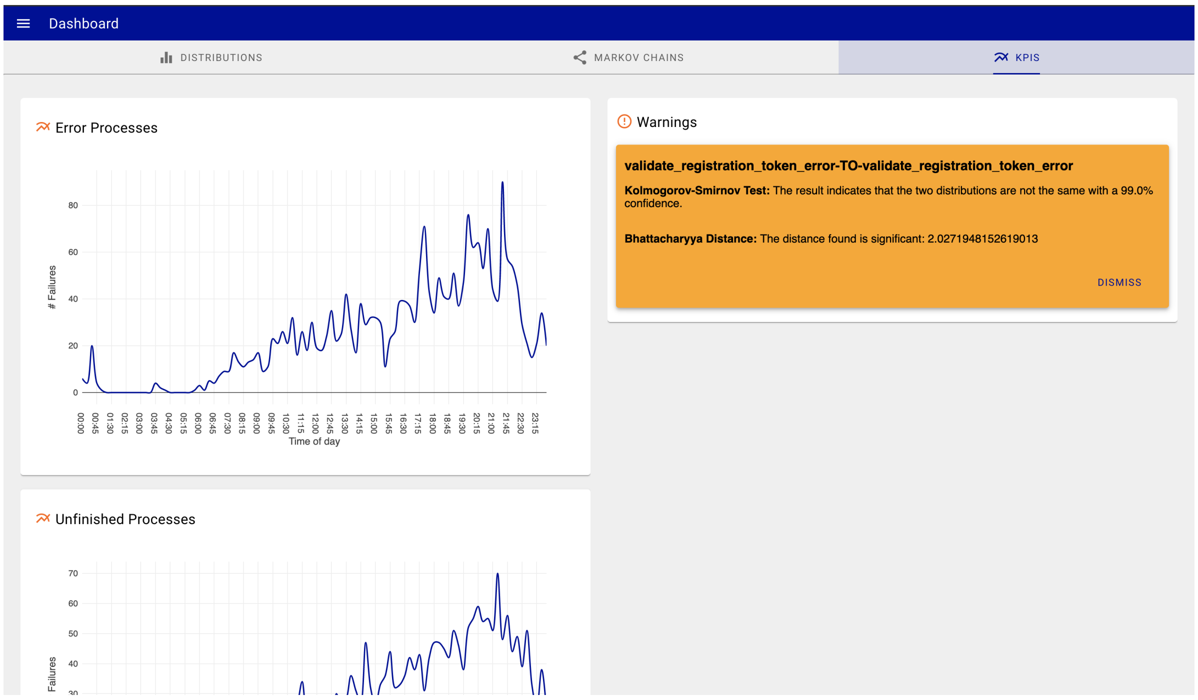


validate_registration_token_error-TO-validate_registration_token_error

Kolmogorov-Smirnov Test: The result indicates that the two distributions are not the same with a 99.0% confidence.

Bhattacharyya Distance: The distance found is significant: 2.0271948152619013

Figure 5.7: Dashboard Version 0.3 : Markov chain Tab



validate_registration_token_error-TO-validate_registration_token_error

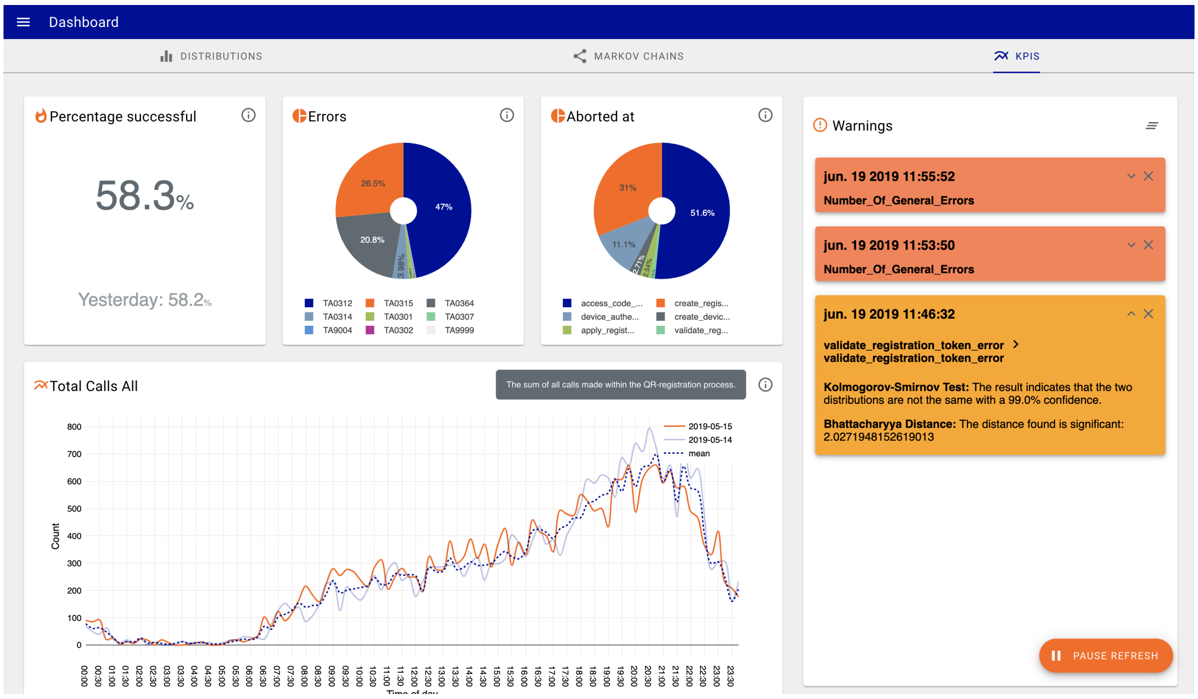
Kolmogorov-Smirnov Test: The result indicates that the two distributions are not the same with a 99.0% confidence.

Bhattacharyya Distance: The distance found is significant: 2.0271948152619013

Figure 5.8: Dashboard Version 0.3 : KPI Tab

Version 1.0

The final version of the application, version 1.0, features some significant changes to the front end. Most of the changes have been to the KPIs tab. This tab now contains pie charts, stacked bar charts and other useful information like the percentage of successful finished processes, as can be seen in Figure 5.9 and Figure 5.10. Each KPI has an infobox with some explanation about the specific KPI which pops up when a user hovers over the information icon.



The sum of all calls made within the QR-registration process.

jun. 19 2019 11:55:52
Number_Of_General_Errors

jun. 19 2019 11:53:50
Number_Of_General_Errors

jun. 19 2019 11:46:32

validate_registration_token_error >
validate_registration_token_error

Kolmogorov-Smirnov Test: The result indicates that the two distributions are not the same with a 99.0% confidence.

Bhattacharyya Distance: The distance found is significant: 2.0271948152619013

Figure 5.9: The KPI tab of version 1.0. New types of charts are added along with extra information on each chart. Warning cards are now expandable by clicking on them.

More warnings have also been implemented based on the newly added KPIs and the width of the warning column has been reduced to allow for more content. By default, warning cards only show the time and subject of the warning. They can be expanded by clicking on the arrow in the top-right corner of the card.

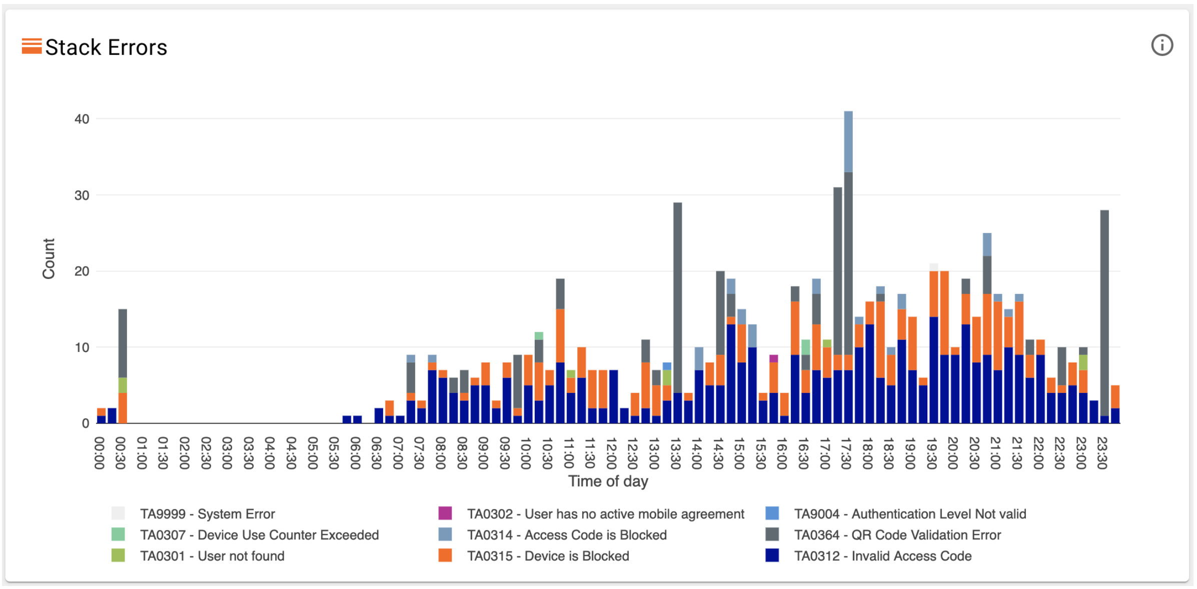


Figure 5.10: Dashboard Version 0.4: KPI Stacked bar chart

Some small changes have also been made to the distribution tab as can be seen in Figure 5.11. The reduced width of the warning column allows for two distributions to be shown alongside each other. An information icon with hover text has also been added to explain what the graph is about.

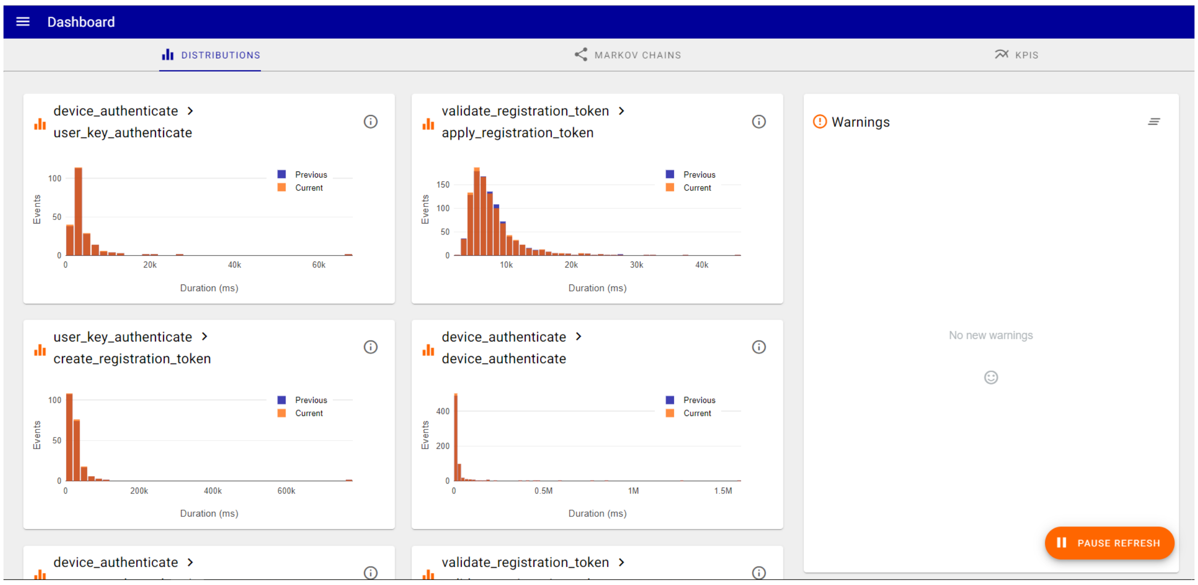

Dashboard
DISTRIBUTIONS
MARKOV CHAINS
KPIS
device_authenticate > user_key_authenticate
validate_registration_token > apply_registration_token
user_key_authenticate > create_registration_token
device_authenticate > device_authenticate
device_authenticate >
validate_registration_token >
Previous
Current
Events
Duration (ms)
Warnings
No new warnings
PAUSE REFRESH

Figure 5.11: Dashboard Version 0.4: The distribution tab

Finally, the Markov chain tab has also been updated. The cards of the Markov chains can now be collapsed to give users a more clear overview of which Markov chains are available. Selecting an element in the Markov chain also highlights all the paths to and from it, as shown in Figure 5.12.

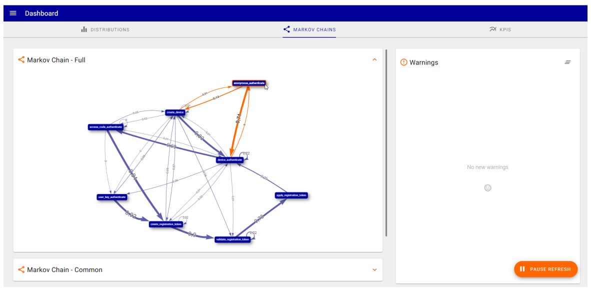


Figure 5.12: A Markov chain with one state selected.

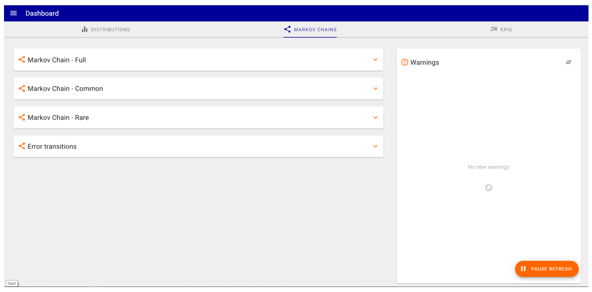
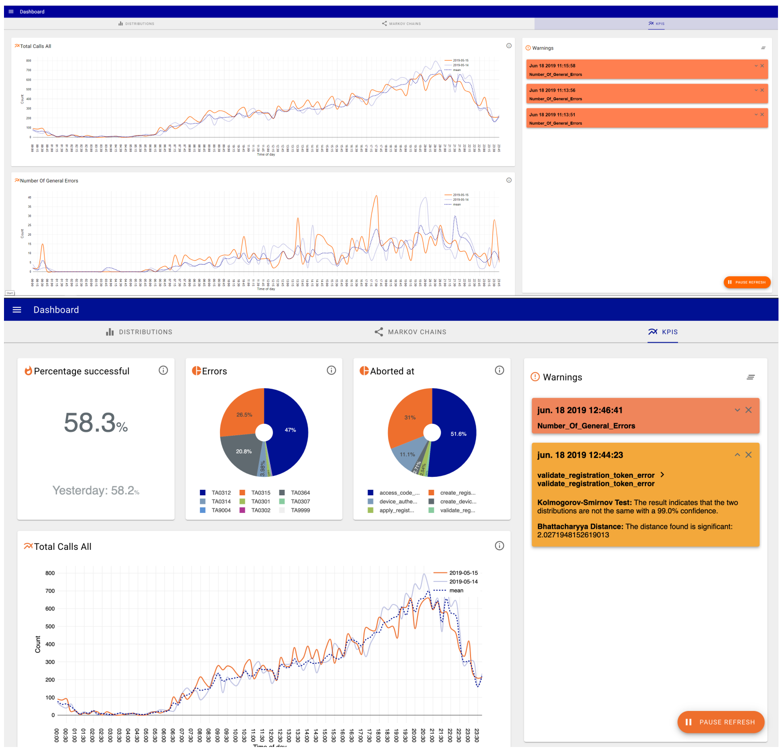


Figure 5.13: The Markov chain tab with all elements collapsed.

Responsiveness

During the development of the application, responsiveness in relation to different screen sizes and ratios has been kept in mind. This allows for users to also view and use the dashboard on different devices and screens, like smartphones, tablets and screens with a 21:9 ratio. Some examples of this can be seen in



Warnings

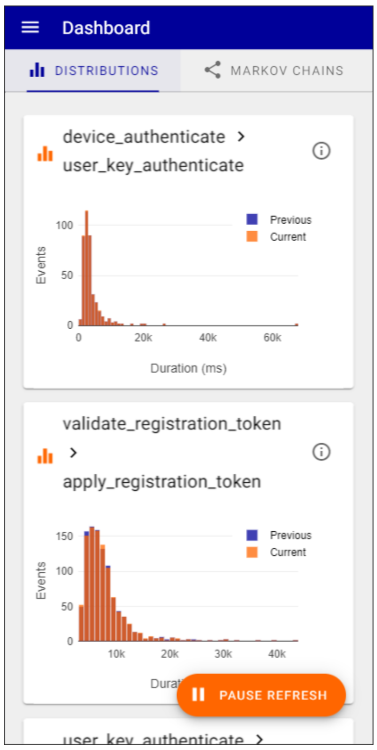
jun. 18 2019 12:46:41
Number_Of_General_Errors

jun. 18 2019 12:44:23
validate_registration_token_error > validate_registration_token_error

Kolmogorov-Smirnov Test: The result indicates that the two distributions are not the same with a 99.0% confidence.

Bhattacharyya Distance: The distance found is significant: 2.0271948152619013

(a) Dashboard on screens with a ratio of 21:9 (top) and 16:9 (bottom).



(b) Dashboard on a smartphone.

Figure 5.14: The dashboard shown on screens with different ratios. All the elements scale accordingly to the screen's size.

6 Testing

Testing software is important in order to deliver a viable product and consists of two phases. The first phase is usually executed by developers and happens on the software environment's level. In the second phase, (simulated) users use the software in order to test the user readiness of the software. Some key aspects of the testing phase as a whole are:

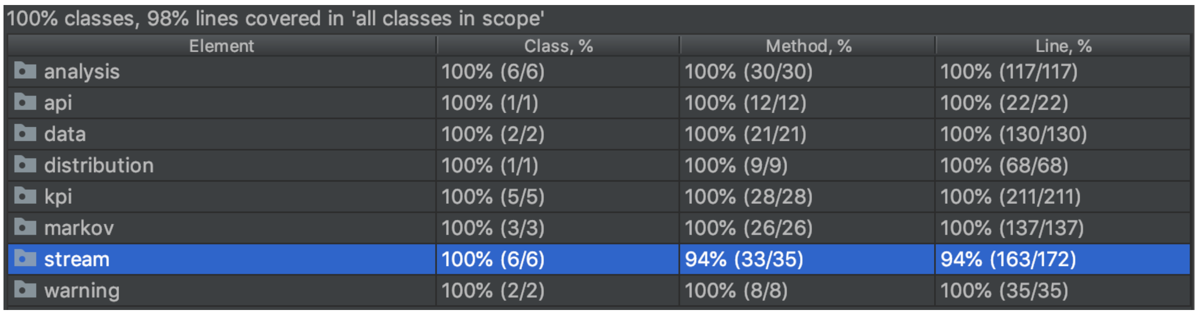
- Is the written code of high quality?
- How does the software perform?
- Is the software user friendly?
- Does the user run into any glitches using the software?

6.1. Technical Testing

In this phase, components of the code are tested individually, ignoring the rest of the software's components, in a process called *Unit Testing*[16]. For this project, the goal is to attain at least 80% code coverage with these tests, which means that at least 80% of all lines of code have to be executed while running tests. Only public, protected and package private methods, excluding constructor methods, are tested directly. Private and constructor methods are indirectly covered as part of other components in the software. The `APIApplication` class seems inappropriate for testing for the same reasons stated in [16]. Therefore, it is not tested.

Another special case is the `StateController` class, which has a `synchronised` method `run`. The test coverage for this class is low (64%) because there was no feasible way to unit test the `run` method.

JUnit will be used as a framework to write and execute unit tests. In order to ignore or manipulate behaviour of other components, so-called *Mocks* are made of dependencies using Mockito.


100% classes, 98% lines covered in 'all classes in scope'

Element	Class, %	Method, %	Line, %
analysis	100% (6/6)	100% (30/30)	100% (117/117)
api	100% (1/1)	100% (12/12)	100% (22/22)
data	100% (2/2)	100% (21/21)	100% (130/130)
distribution	100% (1/1)	100% (9/9)	100% (68/68)
kpi	100% (5/5)	100% (28/28)	100% (211/211)
markov	100% (3/3)	100% (26/26)	100% (137/137)
stream	100% (6/6)	94% (33/35)	94% (163/172)
warning	100% (2/2)	100% (8/8)	100% (35/35)

Figure 6.1: The measured test coverage report, as performed by Intellij IDEA. The highlighted package is the only package in the project with less than 100% line coverage.

6.1.1. JUnit

A common framework to write and execute unit tests for Java is JUnit. It stimulates continuous implementation of tests during software development[3]. This does, however, not mean that writing unit tests suddenly requires little effort. Especially with complex software components, writing unit tests still requires a lot of effort. Even after writing the initial test suite, test components need to be adapted to code change, which requires synchronous implementation of regular code and tests. The overhead of writing tests increases quickly when code changes happen on a regular basis. Therefore, during development of software, testing is done less extensively than arguably optimal[3]. The test coverage on the anomaly detection software is 98%, and has been 98% throughout the entire project, which is well above the 80% goal set for this project.

6.1.2. Mockito

While writing unit tests, it is possible to run into dependencies that have an influence on the behaviour of the component you are testing. In order to ignore or manipulate the behaviour of a dependency, Mockito, a testing framework for Java, can make a so-called *mock* of the dependency. Mockito helps to make sure you are testing an individual component and that component only. It enables you to, for example, determine what certain method calls on a mocked object return while keeping the tested component fully functional.

Mocked objects also enable you to test whether these objects have been processed by a specified method or how many times a specified method has been called on an object. This can be particularly useful when, for example, randomness is involved in the output of a software component.

6.2. User Testing

The user base of this software are programmers themselves. Therefore, we can argue that these users are more knowledgeable than those of a piece of software designed for all possible users. Nevertheless, the software should be easy to use and may cause no confusion among the user base. The users must be able to easily pinpoint anomalies after all.

The first phase of user testing, in this case, was showing demonstrations of the software to the project's supervisors at Rabobank. Demonstrations were given to our supervisors on a weekly basis, which resulted in various amounts of feedback. This feedback was particularly useful, because it gave insight on whether the displayed information was easily accessible and complied with the users' wishes. These sessions were, of course, nothing more than the weekly scrum meetings with the project customer, but were very useful in user testing.

The second phase of user testing consists of letting a group of end users ask all their questions on the application after seeing it. This phase was less useful for the project, because of the simplicity of the software in terms of features, and because of the amount of data made available by the bank. The small amount of data available made it difficult to realistically simulate a stream. In the later weeks of the project, however, the project supervisors realised the importance of data availability and strove to provide as much data in an as real time manner as possible.

The graphical user interface (GUI) was made in compliance with the Material Design[1] guidelines. These guidelines make it easy to create a user friendly GUI. The findings of the second phase of user testing were, as expected, that the GUI looked good and everything was clear. However, the project customer would like to see some changes in layout so that they would have a better overview of everything going on. Again, the GUI is as simple as it is, because the hard work is done in the back-end, but it does exactly what it should do according to the feedback given by users after user testing.

[1]As can be found at https://material.io.

7 Process

This chapter reflects on the development process of this project. The used software development paradigm will be discussed in 7.1 as well as the tools and resources used during development in 7.3.

7.1. Development Paradigm

There are different ways of developing software, each with its own specific up and down sides. During the Bachelor Computer Science, multiple project development paradigms like the waterfall method and the agile way of working are discussed. Based on experiences during other courses, the limited time available and the small size of the group, a partially adapted form of Scrum was employed.

In this adapted version, it was decided to disregard the roles that normally are a part of Scrum, like scrum master and product owner. Instead, each team member had an equal responsibility to keep the backlog up-to-date and make sure their task was done appropriately. Sprint cycles consisted of one week, with a new working version at the end of each week of programming. To reduce the amount of overhead, there were no daily stand-up meetings. The group worked at the same location every day, which meant communication was not a problem because updating the others on progress or asking questions could be done straightaway.

To maintain a clear overview of each team member's assigned tasks and progress during the project, Trello was used, as explained in 7.3.

7.2. Planning

The Bachelor End Project is a short project with a duration of only ten weeks, two of which are meant for research. Due to this limited working time, a well defined planning was essential. As part of the Bachelor End Project, a project plan was made, which outlines our approach to the project and our global planning. This project plan can be found in [Appendix Project Plan

However, due to the nature of banks, adhering to this planning was sometimes difficult. Flexibility in planning was necessary as the group relied on different Rabobank employees, each of them with their own busy schedule. Finding the right person to answer our question and arranging a meeting sometimes was a challenge, as is the case with every large company. However, all of the Rabobank employees were very helpful and tried to make processes as smooth as possible for us, which certainly helped with keeping to our overall planning.

7.3. Project Tools

As is common in software development, we used a set of different tools to help us with the process. The main goals of these tools were version control and project management.

7.3.1. Version Control

In every large software development project that is done by a team of programmers, keeping track of all the changes to your code base and who is doing what is essential. Version control software can help with keeping everything structured and makes cooperation between software developers easier.

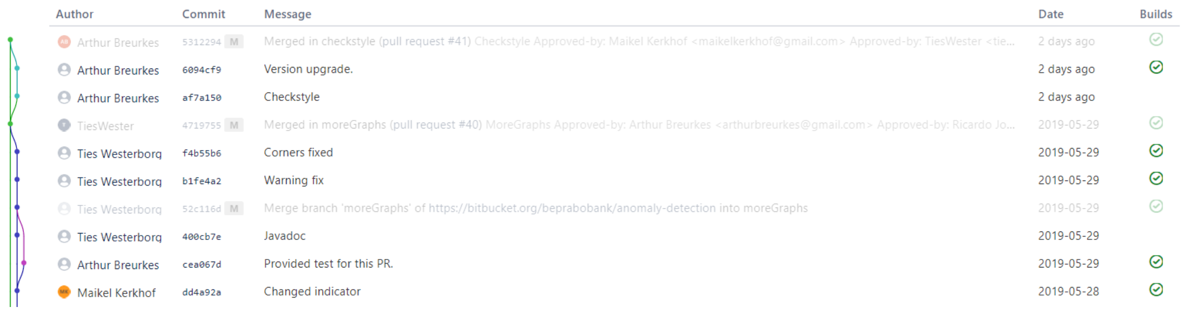
Git

All of the team members had experience with using Git as version control software. Because of the fact that all of the team members would be programming, the familiarity with Git and the ease of use of Git, it was decided to also use it for this project.

The Rabobank already had version control software in place, but different departments used different web-based solutions like GitHub and Bitbucket. Bitbucket however, seemed to be preferred so we decided to also use Bitbucket. It was decided to setup the Bitbucket accounts independant of the Rabobank infrastructure. The main reason for this was the time it would take to get access to the environment, and the fact that the internal environment could only be accessed through a VDI, which would significantly reduce the workflow efficiency.

To take full advantage of Git and to avoid overhead, some rules were set up beforehand. First of all: all development should be done in separate branches. This is a common practice when working with Git and ensures that the master branch always contains a stable, working version. Besides that, after work on a feature is done, all conflicts with the master should be resolved before a pull request is opened. Pull requests should then be reviewed by at least two other team members before they can be merged with the master, after which the branch is archived.

By following these rules, it was ensured that the code base stayed clean and that all team members could work in parallel on their own features without introducing overhead. An example of the Bitbucket usage can be seen in 7.1.



Author	Commit	Message	Date	Builds
Arthur Breurkes	5312294 M	Merged in checkstyle (pull request #41) Checkstyle Approved-by: Maikel Kerkhof <maikelkerkhof@gmail.com> Approved-by: TiesWester <tie...	2 days ago	✓
Arthur Breurkes	6094cf9	Version upgrade.	2 days ago	✓
Arthur Breurkes	af7a150	Checkstyle	2 days ago	
TiesWester	4719755 M	Merged in moreGraphs (pull request #40) MoreGraphs Approved-by: Arthur Breurkes <arthurbreurkes@gmail.com> Approved-by: Ricardo Jo...	2019-05-29	✓
Ties Westerborg	f4b55b6	Corners fixed	2019-05-29	✓
Ties Westerborg	b1fe4a2	Warning fix	2019-05-29	✓
Ties Westerborg	52c116d M	Merge branch 'moreGraphs' of https://bitbucket.org/beprabobank/anomaly-detection into moreGraphs	2019-05-29	✓
Ties Westerborg	400cb7e	Javadoc	2019-05-29	
Arthur Breurkes	cea067d	Provided test for this PR.	2019-05-29	✓
Maikel Kerkhof	dd4a92a	Changed indicator	2019-05-28	✓

Figure 7.1: Overview of commits

Continuous Integration

Another helpful tool was Pipelines, an add-on for Bitbucket. Pipelines are a continuous integration tool which integrates with the git repository. The idea behind continuous integration is that when new code is added to the code base, this code is verified by compiling the code and running tests. This ensures that integration of the new code in the code base is done correctly or shows a warning when there is something wrong.

In the project, Pipelines was used to run a Maven script on every commit done to an open pull request, or to the master, i.e. a merge from a feature branch to the master. This decision was made because the resources on the Pipelines platform were limited by the free plan which was used. By always verifying the feature branch before, and the master after a merge, a stable master branch is guaranteed. An overview of the Pipelines usage can be seen in 7.2.

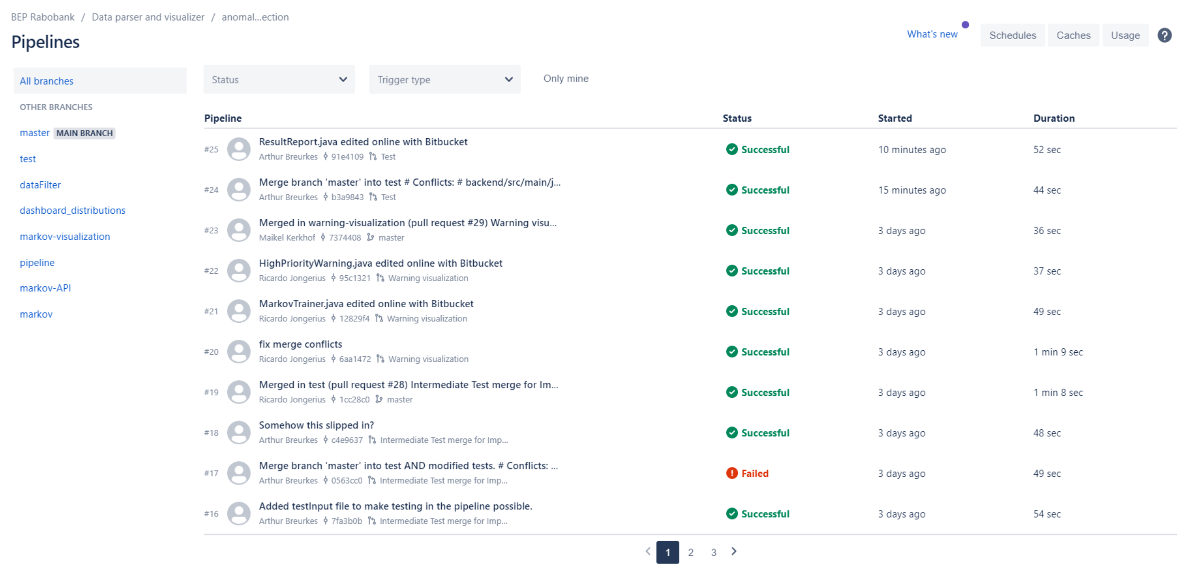
7.3.2. Project Management

When doing a project with a team, an overview of tasks and progress is important. To make sure such an overview was available at all times, different tools were used.

Trello

With larger software projects, keeping track of the backlog and the division of tasks can get complicated. To keep the project manageable and structured, Trello was used. Trello is a simple online tool, in which cards with tasks or issues are divided in different columns representing the state of the task.

An overview of how Trello was used can be seen in 7.3. A card was created for every issue or feature that was to be mplemented. Each of these cards was tagged with a colour code representing the section of the



BEP Rabobank / Data parser and visualizer / anomal...ection

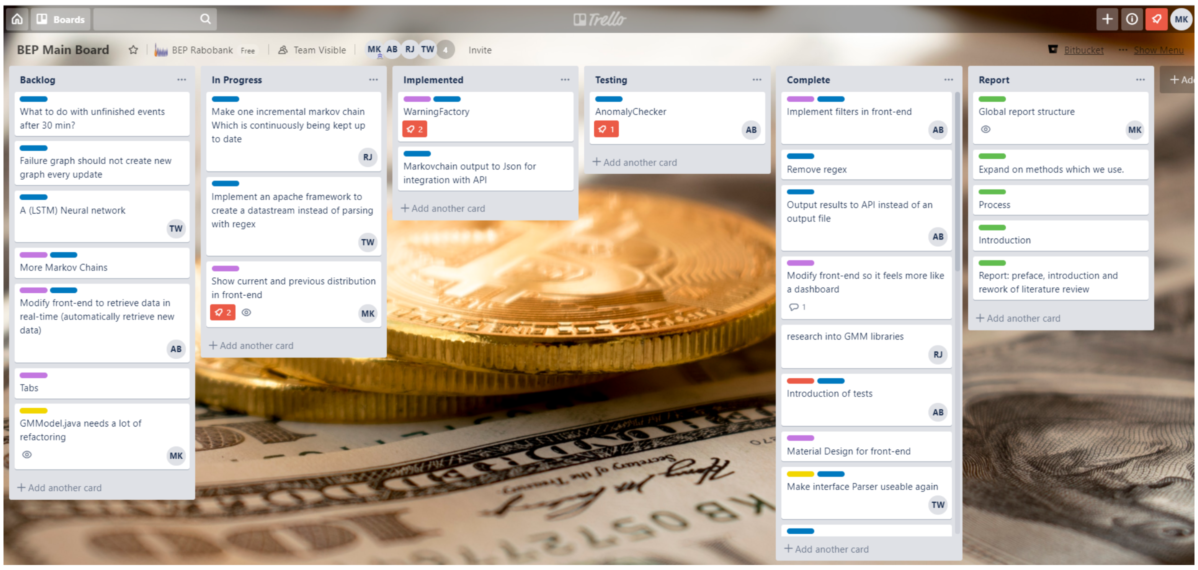
Pipelines

	Pipeline	Status	Started	Duration
#25	ResultReport.java edited online with Bitbucket Arthur Breurkes 91e4109 Test	Successful	10 minutes ago	52 sec
#24	Merge branch 'master' into test # Conflicts: # backend/src/main/j... Arthur Breurkes b3a9843 Test	Successful	15 minutes ago	44 sec
#23	Merged in warning-visualization (pull request #29) Warning visu... Maikel Kerkhof 7374408 master	Successful	3 days ago	36 sec
#22	HighPriorityWarning.java edited online with Bitbucket Ricardo Jongerius 95c1321 Warning visualization	Successful	3 days ago	37 sec
#21	MarkovTrainer.java edited online with Bitbucket Ricardo Jongerius 12829f4 Warning visualization	Successful	3 days ago	49 sec
#20	fix merge conflicts Ricardo Jongerius 6aa1472 Warning visualization	Successful	3 days ago	1 min 9 sec
#19	Merged in test (pull request #28) Intermediate Test merge for Im... Ricardo Jongerius 1cc28c0 master	Successful	3 days ago	1 min 8 sec
#18	Somehow this slipped in? Arthur Breurkes c4e9637 Intermediate Test merge for Imp...	Successful	3 days ago	48 sec
#17	Merge branch 'master' into test AND modified tests. # Conflicts: ... Arthur Breurkes 0563cc0 Intermediate Test merge for Imp...	Failed	3 days ago	49 sec
#16	Added testInput file to make testing in the pipeline possible. Arthur Breurkes 7fa3b0b Intermediate Test merge for Imp...	Successful	3 days ago	54 sec

Figure 7.2: Continuous integration with Bitbucket Pipelines

project it belongs to, e.g. back-end, user-interface or report; and the team member that is currently working on it. Each of these cards started in backlog and was moved to the next column according to the development phase that it was in.

Using Trello this way ensured that it was clear what was done and still had to be done, and who was working on what task at each moment.



Backlog
- What to do with unfinished events after 30 min?
- Failure graph should not create new graph every update
- A (LSTM) Neural network — TW
- More Markov Chains
- Modify front-end to retrieve data in real-time (automatically retrieve new data) — AB
- Tabs
- GMModel.java needs a lot of refactoring — MK
- + Add another card

In Progress
- Make one incremental markov chain Which is continuously being kept up to date — RJ
- Implement an apache framework to create a datastream instead of parsing with regex — TW
- Show current and previous distribution in front-end — 2 — MK
- + Add another card

Implemented
- WarningFactory — 2
- Markovchain output to Json for integration with API
- + Add another card

Testing
- AnomalyChecker — 1 — AB
- + Add another card

Complete
- Implement filters in front-end — AB
- Remove regex
- Output results to API instead of an output file — AB
- Modify front-end so it feels more like a dashboard — 1
- research into GMM libraries — RJ
- Introduction of tests — AB
- Material Design for front-end
- Make interface Parser useable again — TW
- + Add another card

Report
- Global report structure — MK
- Expand on methods which we use.
- Process
- Introduction
- Report: preface, introduction and rework of literature review
- + Add another card

Figure 7.3: Project management with Trello

Google Drive

Besides lots of code, other resources and files had to be shared among team members. Therefore Google Drive was used to efficiently share media, meeting notes and reference material with each other.

8 Evaluation

In the following section the project will be evaluated. Firstly the feedback from SIG will be discussed. After that, a short general evaluation of the project will be given, followed by an evaluation by the Rabobank of the project and product.

8.1. SIG Code Evaluation

There were two moments the code quality was checked by SIG, at the end of week 6, and at the end of week 9. The original feedback, received in Dutch, can be found in Appendix C. Below, the results of these two reports are summarised.

8.1.1. First Review

Our first submission received a score of 3.9 on a scale of 1 to 5, meaning that the maintainability of the code is on par with industry standards. Reasons for not scoring the full 5 out of 5 score are the presence of code duplication and unit size of some methods.

Code duplication complicates the maintenance of the code: when updating a section of code, all of its copies have to be updated as well, resulting in more work. It also increases the risk of forgetting one of those copies.
Code duplication was found in the following classes:

- backend/src/main/java/Main.java en backend/src/main/java/api/APIApplication.java
- backend/src/main/java/warning/MediumPriorityWarning.java en
  backend/src/main/java/warning/HighPriorityWarning.java en
  backend/src/main/java/warning/LowPriorityWarning.java
- backend/src/main/java/analysis/BhattacharyyaResult.java en
  backend/src/main/java/analysis/MarkovResult.java

The unit size score represents the percentage of code that has an above average length. Reducing the methods to have a single responsibility removes the need to have comments in between the code and implicitly documents the functionality through the method's name.
The following methods with a too large unit size were found:

- `StreamHandler.handleFinalEvent(JSONObject,Hashtable)`
- `MarkovChain.combineWith(MarkovChain,int)`

SIG found the presence and amount of tests promising, stating that they hoped that new tests will be added along new production code. Their overall conclusion is that there is still room for improvement, with the hope that this improvement can be realised during the rest of the project.

8.1.2. Second Review

On our second review, no actual score was given. However, the feedback indicates that while the amount of code has grown, the maintainability still improved.
It is noted that the feedback from the first evaluation has been used to improve the code, which resulted in higher scores, especially in the code duplication category. Besides that, SIG indicates that it is good to see that the amount of test code has grown along with the amount of production code.
SIG concludes that the feedback from the first evaluation has been taken into account in the development process.

8.2. Performance Evaluation

The QR device registration process is relatively small scaled. Every Rabobank costumer only has to go through the process once for each device they want to register, but modern consumers having more devices, people resetting their device or people accidentally starting the process results in around 2500 processes started every day. This in turn, results in around 27000 log lines for each day. While this is still a relatively low amount, the developed tool is process-agnostic and should also be able to analyse a much higher throughput of log lines. In order to test what the tool is capable of, 30 days of historical data (827447 lines) was parsed, processed and analysed as fast as possible. The program ran on a laptop with a Intel Core i7 processor and 16GB of RAM, the results are in the table below.

	Run time, average of 5 runs	Events per second
Back end	179.4 seconds	≈4612
Back end + front end	197 seconds	≈4200

Table 8.1: Results of performance tests with the back end and front end of the application.

As can be seen, the application is able to parse, process and run analysis on 30 days of data in just under three minutes, with the front end only having a small impact on the performance when ran on the same device. The average performance requirement for the QR device registration process is about 0.3 events per second (eps), calculated by dividing the average amount of events per day by the amount of seconds per day. Looking at the performance test results, the application can handle around 4200 eps (total events divided by the run time). This shows that the application, in its current form, is more than capable of handling the data for the QR device registration process in real-time and could even be used for the analysis of processes with a higher throughput.

8.3. Project Evaluation

We started discussing our project at Rabobank a few weeks before starting the actual project. Erik Troostheijden and Mireille Brooijmans welcomed us in the headquarters of the Rabobank in Utrecht, where we discussed the terms and scope of the project. The first topic we agreed on was 'Using AI in creating self healing IT systems'. All of us were very excited to work on such a challenging subject.

While discussing this topic with our soon to be supervisor from Delft University of Technology, Sicco Verwer, we quickly came to the conclusion that this subject's scope was too broad and complex for us to achieve a good result in the limited time we were given. After some discussion, we came to the conclusion that this project would focus on detecting anomalies in QR device registration process of the Rabobank mobile application.

On the first day of our project, we were invited to a seminar on development in the cloud, hosted by Rabobank. This was a big event where two speakers came on stage to give informational and motivational lectures on cloud development. The event made us even more enthusiastic for the project.

Finding and reviewing literature went well after Sicco pointed us in the right direction, and we quickly had a basic understanding of what had to be done for the project. However, not everything moved as quickly as we did and the continue with our assignment, we needed data to work with. The Rabobank processes a lot of data that should be handled with care. Therefore, it took us some time to acquire data from the Rabobank that was suited to work with on our own laptops. After receiving the first batch of data, we stayed in close contact with Rabobank's software architect who provided us with more data and helpful feedback along the road.

We, the project group, worked together well on achieving a fast and good looking application for the Rabobank employees, pushing out new and improved features every week. Every time we had meetings at Rabobank and with our supervisor at Delft University of Technology, we could show something new and improved. The reactions were always very positive and we received constructive feedback on which we strove to improve our release for the next week.

Looking back at the project, our dedication and collaboration as a group helped us in delivering the final product successfully. There was never any bad blood among us. We can say that we are all proud of the final product and our progress throughout the project.

8.4. Evaluation by the Rabobank

Throughout the entire project we have kept in close contact with our project supervisor and other colleagues at Rabobank. We have asked our supervisor, Erik Troostheiden, to write a short evaluation of the project from his perspective.

> Dear Arthur, Maikel, Ties and Ricardo,
>
> Since I am not a subject matter expert, I will focus on how you handled the assignment.
>
> We started with defining the detailed scope together to find the optimal combination of business value for Rabobank and small enough to finish within the given time frame.
>
> During the execution of the assignment you initiated update meetings with me and asked for contact persons within the Rabobank organisation where you could get information or further assistance.
>
> In this way you worked very independently without losing side of the Rabobank stakeholders. Within no-time you built a network within Rabobank with the people you needed.
>
> I am honestly very impressed on how independently you worked and how you all really took ownership of everything needed to finish the job.
>
> Thanks for the good job and we will certainly use it!
>
> Kind Regard,
>
> Erik Troostheiden
>
> Deliverymanager Online Access
> Rabobank Nederland
> erik.troostheiden@raboank.nl

9 Ethics

Any project, in software development or not, has to keep its ethical impact in mind. However, anticipating these ethical implications is often difficult, especially because human psychology and culture is very complex and nuanced. Still, general trends can be predicted. There are two main ethical aspects which this project touched upon. Firstly, it was commissioned by a bank, which has some inherent ethical implications in and of itself. Secondly, the developed application handles data, part of which is privacy-sensitive data. In this section, both of these aspects will be explored more in-depth.

9.1. Working at a Bank

On a daily basis, employees of financial institutions and particularly banks make decisions that potentially impact the lives of lots of people. They directly work with large sums of money and, as banks are still companies, they are focused on making profit. In this goal of maximising profit, difficult ethical dilemmas often arise.

Since the global financial crisis and the eurozone crisis earlier this decade, the behaviour of banks, their management and employees have been getting more and more attention by the public. Banks and their directors were heavily criticised for their roles in the crisis. They were often blamed for their excessive risk-taking to increase their short-term profits, while their 'too big to fail' status meant they needed to be saved with public money, also sparking anger [4], [9]. This place in the spotlight of the public results in every misstep again provoking a negative reaction by the public and government [10].

In an effort to improve the situation at banks and avert a crisis like this or similar ethical dilemmas, measures were taken by the government and banks themselves. Banks were nationalised, salaries and bonuses were capped and oversight on banks was improved [10]. One of these measures we encountered during our time at the Rabobank was the banker's oath. The banker's oath is a moral/ethical declaration with the goal of improving bank employees' awareness about possible ethical dilemma's. Everyone employed at a bank is obligated to take the oath, which consists of eight statements about how an employee should handle during their employment, followed by the statement 'so help me god' or 'this I declare/pledge and promise' [1].

Although we all took the banker's oath as required by law, it was somewhat less relevant in our situation. As temporary interns, we did not have any direct responsibilities towards customers. Other, more generic statements from the oath did apply, like keeping confidentiality and the abuse of knowledge were somewhat relevant and, as we took the oath, certainly kept them in our minds during the project.

9.2. Handling Data

Another aspect of software development that could lead to ethical dilemmas is the usage of user data. In recent years, the mass collection and analysis of sensitive personal data has often been the topic of discussion, with countless scandals topping the headlines. While most of them concern massive companies storing even more massive amounts of data, keeping user privacy in mind is important in every software project.

Software development at a bank holds some challenges on its own. The data that a bank needs to properly perform its job, like citizen service numbers (BSN) and personal financial information like balance and account numbers, are often deemed very sensitive. Handling this data with care is therefore of great importance, and applications should only be able to access the data they need and make sure that users can only access their own data.

The data we used for our project consisted of log files collected in Splunk. Every line in these files represented an event generated by an application or step in the QR device registration process. Besides essential information like a time stamp, session id and the type of event, other more sensitive information was present. The unredacted version of the data set contained, for example, the account numbers of the user, identifying information like an IP-address, whether or not it was a personal or business user and the model of the used device.

A part of these fields were redacted before we received the data, like account numbers and IP-addresses, as they were in no way necessary or useful for our application. After identifying the useful data for our application later in the development phase, we redacted all the fields that weren't used by our application, before feeding the data to our application. These included the personal or business identifier and some fields from the device properties like MAC-address. By removing those fields and only show results of analysing the data in the application, the risk of personal data leaking out is minimised without the loss of functionality.

10 Conclusion

The goal of the project was to create a tool which could, in real-time, detect anomalies in the log data of the QR device registration at the Rabobank. This section will first reflect on the goals which were set, on the basis of the MoSCoW principle, and after that reflect on the project in general.

10.1. Reflection on Product Requirements

After talking with the client, the Rabobank, a MoSCoW list of product requirements was made. In the table below an overview is given of which points have and have not been implemented in the final product. One of

MoSCoW requirement	Implemented?
M: Learn individual distributions	✓
M: Anomaly detection in event distribution	✓
M: Warnings	✓
M: Static GUI	✓
S: Customisable warnings	✓
S: Interactive GUI	✓
S: Interactive graphs	✓
S: Combining data	✓
S: Visualise session	
S: Process agnosticism	✓
S: Real-time processing of logs	✓
C: Data agnosticism	✓
C: Detection of session patterns	
C: Export to Excel	
C: Common factor session	
W: Prediction of future anomalies.	

Table 10.1: MoSCoW list of requirements, as described in chapter 2

the should haves, Visualising a session, has not been implemented. This was a conscious decision, as the goal of the application was to handle data in real time. When handling data in real-time, data is often discarded as soon as it has been analysed. By doing this, visualising a single session becomes very difficult, which is why the decision was made to discard this point.

The last Should have, Real-time processing of logs, has not been implemented fully. This is because it would take far too long to get the proper permissions inside the Rabobank to be connected to the live stream of log data. However, the application pretends the data from files that are read is a stream, meaning it would take minimal effort to connect it to an actual real-time stream.

Because the client requested a more polished GUI, this is what was focused on in the later weeks. The GUI has subsequently progressed a lot during this period, as can be seen in subsection 5.2.2. Because of the progress made in the GUI, there was less focus on the remaining Could haves. Regardless, most of the goals set have been reached.

10.2. Final Remarks

All in all, the project can be considered a success. The original goal was to create a tool which could, in real-time, detect anomalies in the log data of the QR device registration at the Rabobank, which is exactly what was made. Not all product requirements and features were implemented to the same degree, but overall the product is satisfactory and works as intended. Per chapter 8, all the parties involved are happy with the final product, and the code quality is also up to par.

11 Recommendations

Now that the project has been concluded, it is time to look at the future. In this chapter, some recommendations will be made to the Rabobank on how to further improve the developed application. Each section in this chapter describes a part of the program which can be extended.

Connecting to a live data stream

Due to the nature of banks, and the limited time allocated for the project, it was not possible to connect the application to the actual live feed of log data. Instead, historical data was used to mimic such a live feed. Engineers at the Rabobank will not face these kinds of problems, so a connection to the live feed of data should be relatively easy to implement. In fact, they could also use the Splunk API to routinely create an export of historical data and saving it in a certain folder, which would render the implementation of a live connection trivial.

Further extension on KPI anomaly detection

Methods to detect anomalies on distributions, Markov chains and simple graph KPIs have been implemented. However, as time was limited, anomaly detection on more complex KPI structures like stackedbar charts were not implemented. Since useful information is extracted from the provided log data and this data lives well formatted within the KPI classes described in subsection 5.1.2, the implemented methods to detect anomalies can easily be extended.

Moreover, Neural Networks could be used to detect and predict anomalies on the extracted data. Recall that the data of Graph KPIs, as earlier defined, consist of merely two arrays of equal size, with one axis being time, and is therefor actually time series data. This 2-dimensional data could be used to train a Neural Network. Fabio Lattario Fonseca, an employee at the Rabobank, together with a small group of colleagues, is exploring the possibilities of using a Long Short Term Mermory Neural Network (LSTM NN). We've had the pleasure of attending a meeting where problems like the lack of erroneous data arose. Their idea was to train a LSTM NN on normal behaviour of a system and assume that when the network was unable to recognise the current behaviour, abnormal behaviour i.e. anomalies were detected. During this meeting, a crash course on Neural Networks, LSTM in particular, was given and a start on the actual network was made. A proof of concept was implemented and seemed very promising.

While the meeting focused on training a model on different data, implementing such a network could improve the performance and accuracy of the currently implemented program significantly. Also, Neural Networks aim to predict anomalies and potential outages. We therefor strongly recommend the Rabobank to explore these opportunities.

Bibliography

[1] The banker's oath, Apr 2016. URL `https://www.tuchtrechtbanken.nl/en/the-bankers-oath`.

[2] A. Bhattacharya. On a measure of divergence between two statistical populations defined by their probability distributions. *Bulletin of the Calcutta Mathematical Society.*, 35:99–109, 1943.

[3] Yoonsik Cheon and Gary T Leavens. A simple and practical approach to unit testing: The jml and junit way. In *European Conference on Object-Oriented Programming*, pages 231–255. Springer, 2002.

[4] Rutger Claassen. Financial crisis and the ethics of moral hazard. *Social Theory and Practice*, 41(3):527–551, 2015. ISSN 0037802X, 2154123X. URL `http://www.jstor.org/stable/24575743`.

[5] Ian G Clifton. *Android user interface design: implementing material design for developers.* Addison-Wesley Professional, 2015.

[6] Stijn Dongen. A cluster algorithm for graphs. Technical report, CWI, Amsterdam, The Netherlands, The Netherlands, 2000.

[7] Jenna Erickson. 2015 was the year of material design, Jun 2018. URL `https://usabilitygeek.com/2015-was-the-year-of-material-design/`.

[8] Abida Haque, Alexandra DeLucia, and Elisabeth Baseman. Markov chain modeling for anomaly detection in high performance computing system logs. In *Proceedings of the Fourth International Workshop on HPC User Support Tools*, page 3. ACM, 2017.

[9] Øyvind Kvalnes and Salvör Nordal. Normalization of questionable behavior: An ethical root of the financial crisis in iceland. *Journal of Business Ethics*, Feb 2018. ISSN 1573-0697. doi: 10.1007/s10551-018-3803-8. URL `https://doi.org/10.1007/s10551-018-3803-8`.

[10] Bart Meijer. Dutch government launches new bid to curb bankers' pay, 2018. URL `https://uk.reuters.com/article/netherlands-government-banks/dutch-government-launches-new-bid-to-curb-bankers-pay-idUKL8N1YN2FM`.

[11] Tim Oates, Laura Firoiu, and Paul R Cohen. Clustering time series with hidden markov models and dynamic time warping. In *Proceedings of the IJCAI-99 workshop on neural, symbolic and reinforcement learning methods for sequence learning*, pages 17–21. Citeseer, 1999.

[12] Rabobank. Rabobank group interim report 2015, 2015. URL `https://www.rabobank.com/en/images/rabobank-group-interim-report-2015.pdf`.

[13] Raquel Sebastião, João Gama, Pedro Pereira Rodrigues, and João Bernardes. Monitoring incremental histogram distribution for change detection in data streams. In *International Workshop on Knowledge Discovery from Sensor Data*, pages 25–42. Springer, 2008.

[14] Stefan Trueck and Teodosii Rachev. *Measures for Comparison of Transition Matrices*, pages 129–143. 12 2009. ISBN 9780123736833. doi: 10.1016/B978-0-12-373683-3.00008-7.

[15] vis.js. vis.js - a dynamic, browser based visualization library., 2019. URL `https://visjs.org/`.

[16] James A Whittaker. What is software testing? and why is it so hard? *IEEE software*, 17(1):70–79, 2000.

[17] Ian T. Young. Proof without prejudice: Use of the kolmogorov-smirnov test for the analysis of histograms from flow systems and other sources. *The Journal of Histochemistry and Cytochemistry*, 25:935–941, 01 1977.

A Original Project Description

Using AI in creating self healing IT systems

Organisation background

The Rabobank Online department is responsible for the IT for `www.rabobank.nl`, Online banking (web and app).

Goal

As department we want to improve the availability of our systems by applying Artificial Intelligence. Sometimes also referred to as AIOPS. The aim for the students assignment is to make the first steps here, where the end ambition is that 90% over are outages should be prevented by AI.
If you look at logical stages in applying AI in an IT OPS environment you can think of:

1. Collect data;
2. Use Algorithms to analyse data;
3. Give an automated advice on action to be taken;
4. Executing the action advised.

After the 2 weeks literature study at the start of the assignment, it is to be determined what a realistic end goal could be.

Technical

We are using tools like Splunk and Zabbix which should be usable as a environment in creating these kind of AI solutions. Or at least a view of the necessary steps.

B Infosheet

General Information

Project Title: Real-time Anomaly Detection in Critical Rabobank Processes.

Organization: Rabobank, Online Department

Date of Final Presentation: 05-07-2019, 11:00 AM

Project Description

Real-time processing of log data can give valuable insights in the behaviour of systems. The Rabobank is a large bank with several critical systems. One of such systems is the QR device registration process. In order to monitor this process, an application was built which detect abnormal behaviour. The application reads the log data, parses it to interpret it better, and then analyses it. Analysing is done by fitting distributions of the duration of each step in the process, and checking for anomalies in new incoming data. Finally, a dashboard was made in which the analysis and the distributions are visualised. More information is also available in the dashboard, like a Markov chain of the process and several Key Points of Interest.

Team Members

- Arthur Breurkes - A.R.Breurkes@student.tudelft.nl
- Ricardo Jongerius - R.Jongerius@student.tudelft.nl
- Maikel Kerkhof - M.M.H.A.Kerkhof@student.tudelft.nl
- Ties Westerborg - T.D.Westerborg@student.tudelft.nl

Coach

Dr.ir. Sicco Verwer, TU Delft - S.E.Verwer@tudelft.nl

Clients

Erik Troostheiden, Rabobank - Erik.Troostheiden@rabobank.nl
Mireille Brooijmans, Rabobank - Mireille.Brooijmans@rabobank.nl

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

C SIG Feedback

C.1. First Review

The original feedback received in Dutch from the Software Improvement Group on our code submitted in week 6 of the project.

De code van het systeem scoort 3.9 sterren op ons onderhoudbaarheidsmodel, wat betekent dat de code marktgemiddeld onderhoudbaar is. We zien Duplication en Unit Size vanwege de lagere deelscores als mogelijke verbeterpunten.

Bij Duplication wordt gekeken naar de hoeveelheid gedupliceerde code. We kijken hierbij ook naar de hoeveelheid redundantie, dus een duplicaat met tien kopieën zal voor de score sterker meetellen dan een duplicaat met twee kopieën. Vanuit het oogpunt van onderhoudbaarheid is het wenselijk om de hoeveelheid gedupliceerde code zo laag mogelijk te houden. Na verloop van tijd zal de gedupliceerde code moeten worden aangepast. Dit leidt niet alleen tot extra werk, aangezien op dat moment alle kopieën tegelijk moeten worden veranderd, maar is ook foutgevoelig omdat de kans bestaat dat één van de kopieën per ongeluk wordt vergeten.
Voorbeelden in jullie project:

- backend/src/main/java/Main.java en backend/src/main/java/api/APIApplication.java
- backend/src/main/java/warning/MediumPriorityWarning.java en backend/src/main/java/warning/HighPriorityWarning.java en backend/src/main/java/warning/LowPriorityWarning.java
- backend/src/main/java/analysis/BhattacharyyaResult.java en backend/src/main/java/analysis/MarkovResult.java

Bij Unit Size wordt er gekeken naar het percentage code dat bovengemiddeld lang is. Dit kan verschillende redenen hebben, maar de meest voorkomende is dat een methode te veel functionaliteit bevat. Vaak was de methode oorspronkelijk kleiner, maar is deze in de loop van tijd steeds verder uitgebreid. De aanwezigheid van commentaar die stukken code van elkaar scheiden is meestal een indicator dat de methode meerdere verantwoordelijkheden bevat. Het opsplitsen van dit soort methodes zorgt er voor dat elke methode een duidelijke en specifieke functionele scope heeft. Daarnaast wordt de functionaliteit op deze manier vanzelf gedocumenteerd via methodenamen.
Voorbeelden in jullie project:

- StreamHandler.handleFinalEvent(JSONObject,Hashtable)
- MarkovChain.combineWith(MarkovChain,int)

De aanwezigheid van testcode is in ieder geval veelbelovend. De hoeveelheid testcode ziet er ook goed uit, hopelijk lukt het om naast toevoegen van nieuwe productiecode ook nieuwe tests te blijven schrijven.

Over het algemeen is er dus nog wat verbetering mogelijk, hopelijk lukt het om dit tijdens de rest van de ontwikkelfase te realiseren.

C.2. Second Review

The original feedback received in Dutch from the Software Improvement Group on our code submitted in week 9 of the project.

In de tweede upload zien we dat het codevolume is gegroeid, terwijl de score voor onderhoudbaarheid is gestegen.
We zien dat de verbeterpunten uit de feedback op de eerste upload zijn aangepast, en op deze gebieden is dan ook een verbetering in de deelscores te zien. Zeker bij Duplication is het jullie gelukt om een grote verbetering te realiseren. Ook is het goed om te zien dat er naast nieuwe productiecode ook nieuwe testcode is geschreven.

Uit deze observaties kunnen we concluderen dat de aanbevelingen van de vorige evaluatie zijn meegenomen in het ontwikkeltraject.