Combining usage and profile data for retrospectively analyzing usability of applications with funnels

Thijs Wiefferink
thijs@wiefferink.me, s1366564

Master Thesis

Master Computer Science (Software Technology specialization)

University of Twente

May 2017 - March 2018

Supervisors from the University of Twente
Marten van Sinderen (m.j.vansinderen@utwente.nl)
Arend Rensink (arend.rensink@utwente.nl)

Supervisors from Staying, located in Amsterdam
Emiel Mols (emiel.mols@staying.nl)
Geert-Jan Bruinsma (geertjan.bruinsma@staying.nl)

Abstract

This research project shows how usage data gathered using analytics tools from apps and websites can be stored in a structured and scalable way, so that it can be combined with profile data (from the application itself) when the company has questions about their users. Building on this setup this project shows how common application processes, like user sign up, can be tracked and visualized in funnels. A funnels shows how many users successfully made it to each step of a process, giving insight into their behavior. Next to that funnels can show context information to help companies determine why certain users continued or not. These funnels help companies to correctly track usage of their product, so that they know which parts to improve, and if their changes are actually an improvement.

After a system to store and combine usage data with profile data has been created it has been used at the company Staying. This has lead to improvements of the system based on their feedback and validated its usefulness in practice. Four funnels have been made that are now used by Staying, covering their most important processes. These funnels are accommodation sign up, guest sign up, email delivery and text message delivery. Throughout the research project care has been taken to provide alternative options and notes about implementing the system at other companies, which makes it interesting for companies that search for an analytics solution.

Keywords

Analytics, funnels, databases, aggregation, tracking, apps, websites, PostgreSQL, CitusData

Preface

This is the Final Project thesis for the Master Computer Science at the University of Twente, specialization Software Technology. It started with a Research Topics project of 10 ECTS credits to do domain research and come up with a research plan, continuing with this Final Project of 30 ECTS credits to execute this plan. During the project Marten van Sinderen is the main supervisor and Arend Rensink the second supervisor from the University of Twente. Emiel Mols and Geert-Jan Bruinsma are the founders of Staying, and are the supervisors from the company side.

Contents

1 Introduction

This chapter describes the research domain, illustrates the question that sparked this project and provides information about the environment in which the project has been executed.

1.1 Research domain

In this digital age we all use a lot of apps and websites [9], which we would like to be as easy to use as possible. To build applications that fulfill this wish, the companies working on them gather an increasing amount of data [23]. This data could contain information about the pages users visit, the features they use, and any other relevant actions. Together this data is called usage data, or in a more generic way; analytics data. In this project all usage data of customers using a product of a company is meant. Next to this usage data company also has application data, such as user profile data containing name, email, date of birth, anything else the company stores. The other types of application data that a company has for a product depend on the product itself. For a chat service it would be chat groups and messages, and for a file storage service it would be files and folders.

By analyzing usage and profile data the company can make changes to improve their Key Performance Indicators (KPI's). The most important KPI of a company is usually revenue, since that is the end goal of a company. Other secondary KPI's of a company could be the number of daily active users, purchases per week, etcetera. Even though companies gather more and more data, collecting the right data that is required to extract relevant statistics is still a challenge. Often the company comes up with a new question about the usage of their software product, and find out that they did not collect the necessary usage data to answer this new question. Section 1.2 illustrates the process of coming up with a new analytics question, gathering the required data and answering the question.

Even if all required data has been collected and can be extracted from an analytics system of a company, displaying and interpreting the data is still vital for it to become helpful. For simple questions a single number or yes/no answer is most likely enough, for example "How many chat messages are sent?" could be answered by "5048". This can be extended to graphs when the company wants to know how the answer of a question has changed over time. The question about the number of chat messages would in this case be displayed as a graph, showing the number of chat messages per hour/day/week/month. A next question of a company might be "At which step of the sign-up process of our website do most users stop?". This question could be answered by a simple "step X", but what a company really wants to know is how many users stop in each step, so that it can improve the sign-up process at the steps that perform the worst. A good way to answer this question is by creating a funnel, which is a visualization that shows how many users completed each step of a process. More details and an example funnel are described in Section 1.3.

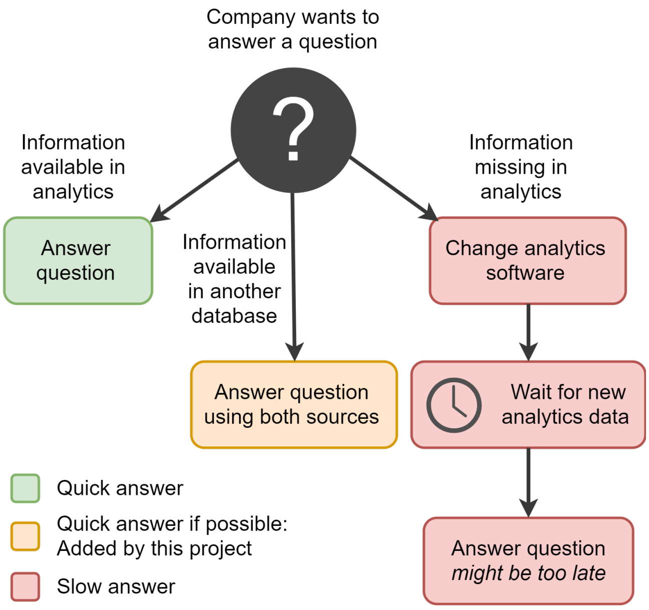

Company wants to answer a question
?
Information available in analytics
Information missing in analytics
Answer question
Information available in another database
Change analytics software
Answer question using both sources
Wait for new analytics data
Quick answer
Quick answer if possible: Added by this project
Slow answer
Answer question *might be too late*

Figure 1: Answering an analytics question: approach options

1.2 Analytics question example

For tracking users on a website and/or app there are numerous products available, each with their strong and weak points. Setting up one or more of these products is easy and can quickly give the company basic statistics about their product. However, after the basic questions (like how many users visited page X last week?) there will probably be more detailed questions these analytics solutions do not answer. To illustrate the process of answering an analytics question a basic flowchart has been created, shown in Figure 1.

There are three possibilities, the first is that all required information is already available in the usage data, which means that with a query of some sort (SQL or some visual tool) the question can be answered. This option is quick and can provide the answer almost immediately. For example, a company wants to know when an user was active for the last time, which could be answered immediately by finding the last activity of the user in the usage data. This option assumes that the company has full access to the usage data, which is the case with most, but not all, analytics tools.

Another possibility is that the required information is not available in the usage data, for example when a company wants to know if the selected language by users would affect how often they open the app. For example, consider an app to let guests communicate with hotels and a guest selects a language in which the hotel did not provide information, it might affect the usage of the app. Since the language of the user is not available in the usage data, it needs to be added before the question can be answered. After adding the language information to analytics events of app usage, the company needs to wait a week or two to collect new analytics with this addition, after which the question can be answered. The delay between coming up with a new question and being able to answer it is slowing down the improvement of the product, which is not good for the company.

For the above question there is another option that is in between the two given options, which is to immediately combine the usage and profile data. This means combining the usage data from the analytics tool and the profile data from the application at query time, which allows answering the questions immediately. In this case it would mean combining the data of a user with the app usage events of that user, so that usage events can be grouped or filtered by language. To be able to use this possibility, there are requirements for the storage of the two data sources and the format the data is stored in. These requirements are described in Chapter 3. The option of combining usage and profile data immediately is what this project researches, which leads to the requirements for this option, its effectiveness, limitations and performance. To demonstrate that the combination of usage and profile data works correctly a prototype system to answer analytics questions and build funnels (overview of analytics about a step-by-step process performed by a user) has been created.

1.3 What is a funnel?

A funnel is a visualization of a sequential business process of multiple steps, showing how many users completed each step. This definition is hard to understand, so let's start from the beginning, with a business process. Such a process could be sending a newsletter, a new user signing up, a purchase in a web shop, or anything else. Each of these processes consist of multiple steps, where each step is an action performed by a user that brings him closer to the end goal of the process. Let's walk through sending a newsletter as example, this process has the following steps:

1. The company sends a newsletter
2. Users open and read the newsletter
3. Users click a link in the newsletter, visiting the website

These three steps are the key steps of sending a newsletter. For each of these steps the company would like to have statistics. The company could have questions like "To how many people have we sent a newsletter last week?", "How many people opened/read the newsletter?", "How many people visited the website because of the newsletter?". To answer all questions about a process with multiple steps, like this one, a 'funnel' can be created to show all relevant statistics. A funnel shows the steps of a process, each with statistics about the number of people that made it to the step. Funnels are used to show how many of the users that started the process, make it to the end, while showing where in the process users drop out if that is the case. For individual steps in a funnel additional context information could be shown, for example the percentage of users that visited the website using a mobile device for the third step of the newsletter funnel.

Funnels visualize the processes users go through to perform actions in the product of the company. In this visualization they show key statistics to understand how far users get, and give the company insights about process. A huge stream of usage data is interesting, but hard to extract useful information from, that is why funnels are so valuable. Funnels condense the information to the statistics that are most valuable for the company, while hiding the raw data. Funnels are important for companies to measure how effective each step of their process is, so that more effort can be put into improving the weakest steps. For example if a high percentage drops out of the funnel after reading the email, it could mean that the email is not convincing enough, or that it gets caught by spam filters. So when the company knows the email is not effective it can redesign the email, improve spam scores or something else. To show that the systems works as expected funnels of the most important processes of a company could be created.

1.4 Problem statement

Collecting and visualizing analytics is easy for tracking simple statistics like page visits. However, customizing the collection and visualization is not as easy. Challenges that appear are scalability, flexibility, processing, reporting and missing data. Most of these challenges can be solved using existing tools, but combining them to get a system that works is not straightforward. The two key points of this research are solving the missing data problem and visualization using funnels.

The missing data problem occurs because companies don't know which questions they will have in the future, and therefore it is unknown which data needs to be collected. When information missing to calculate the desired statistic, it can be added to the collection software. The problem is that this new data is only collected from the point in time that the collection software is updated. This causes a delay because now the company needs to wait for more data to come in before the desired statistic can be calculated. To prevent this problem, profile and usage data could be combined to fill in the missing data immediately. The problem with combining usage and profile data is that these sources have different storage types (an event stream versus a relational database), are stored in different systems and might not have a common field that allows combining them (like a user id).

Funnels are used to visualize statistics about processes in products of companies. The first part of building funnels is collecting the required statistics, which can be done by querying the combined usage and profile data. The solution of the missing data problem provides a way to do that. The second part of creating funnels is displaying the results in a way that is easy to understand (also for non-technical employees of the company). For designing this interface different solutions by analytics providers could be compared, adding features where necessary. User testing with people of a company also can be done to improve the visualization step-by-step.

These problems are analyzed, solved and presented in this Master Thesis.

1.5 Goals

The first goal is to improve the chance that analytics questions about software products usage can be answered immediately by combining usage and profile data with a scalable setup. The second goal is to visualize processes with a funnel, answering more complex questions about the behavior of users of a software product.

A challenge of implementing funnels is tracking users through all steps, possibly connecting different identification methods (email, session id, user id, etcetera). Another challenge is building a clear visual representation, making it easy to understand for everyone in the company.

1.6 Research questions

The following questions are answered in this project:

1. How can usage and profile data of a software product be used more effectively for improving usability?
   (a) How effective is retrospectively combining usage and profile data to answer analytics questions?
   (b) How can business processes be tracked and visualized in a funnel?

1.7 Relevance

The problem that is investigated is relevant for most companies that develop an application where usage and profile data is involved. The application could be an app, web app (website with some state saved server-side) or general computer application. Companies that have a mature analytics solution probably already dealt with this problem in some way or form, most likely by adding (a part of) the profile data in the usage data. For example adding the language of a user to all his actions that are collected as usage data. This project wants to prevent duplication of the profile data and will combine data from both sources at query time. This would make it easier for, especially fast moving startups, to get the information they need from the analytics so that they can improve their product by focusing on the most important aspects of it.

Visualization of the analytics results makes the results accessible for all people in the company, instead of only the technical guys that know how to write SQL queries. This research helps the adoption of funnels, and in particular funnels for complexer process with additional context information.

1.8 Case study at 'Staying'

In order to validate the solutions proposed in the project, a case study at the company Staying [1] has been performed. This way the ideas and plans are tested in practice, which will reveal the practical challenges and provides data to evaluate and benchmark the solution. Each step of the research project will take into account how the presented solutions would work for companies in general, ensuring the end result is applicable for any company that faces the same problem. The visualization of funnels can also be tested this way, to improve it further. Staying is a startup company located in Amsterdam that creates an app for hotels to chat with their guests. Next to chatting, the app

[1] `https://staying.nl`

provides information like nearby tourist attractions, restaurant menu, wifi code and anything else the hotel wants to show. In addition to this app used by guests there is a web app, called the portal, where accommodations upload the information they want to show and answer the chat messages of the guests. Staying is trying to find the right market fit, and is working to optimize two main processes: property signup and guest invites. For both of these processes it is important to be able to measure the results correctly and quickly, so that changes can be made to improve them. Especially in the starting phases of a company it is important to be able to make and measure changes quickly, so that the turnover time is low.

More details about what Staying does and how it is technically organized follows in the next section.

1.9 Structure of the Staying application

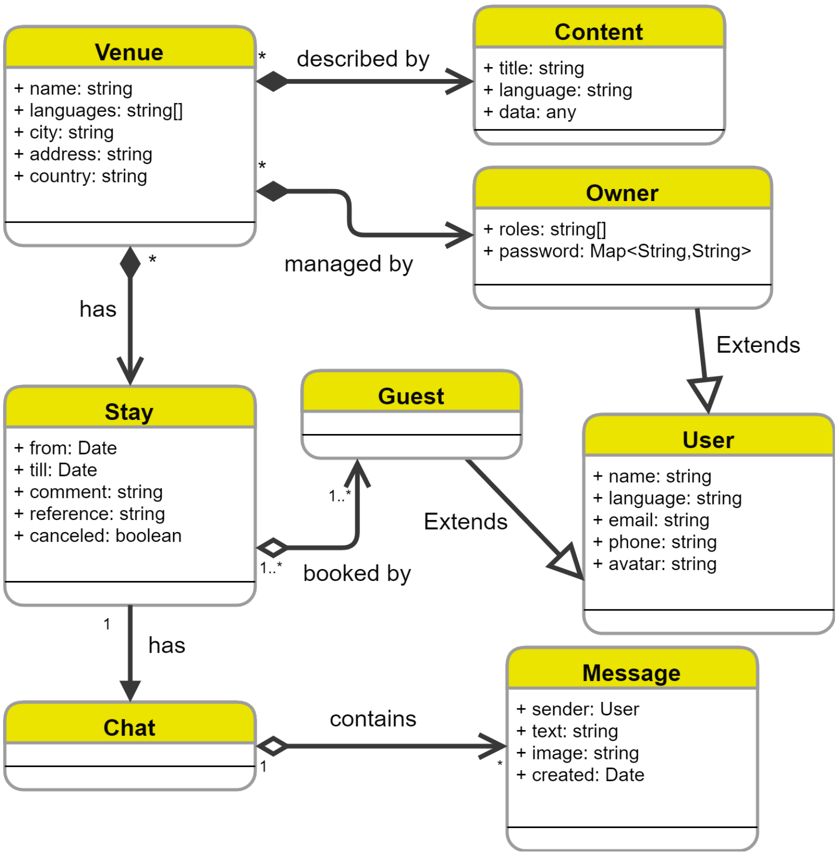


Figure 2: Staying application structure

To get a better idea about the application of Staying, which will be referenced throughout this thesis, the structure is described in this section. The important entities in the product of Staying are users, stays, venues, their content and chats with their messages, the relations can be seen in Figure 2. A venue represents a property like a hotel, apartment complex, etcetera. This venue is managed by one or more users, which are the owners (as a specialized type of user). A venue invites guests when they make a booking, this booking is represented as a stay. A stay has the arrival and departure date and is connected to a user representing the guest. Venues have content items linked to it, these content items are pieces of information like the wifi code, metro map, restaurant menu, etcetera. Content items can be created for different languages, so that the guest sees the content for his language in the app. Owners of a venue manage the content items and other venue settings in the web portal. Each Stay is linked to a Chat, in which the owners of the venue and the guests of the stay can chat. The Chat contains of a list of messages. Messages have a sender, time and platform (they could be send from the portal, venue app, guest app or by SMS message).

1.10 Approach

To solve the missing data problem the profile and usage data have been brought together into a single system. A common solution used in practice at companies is to add profile data to usage events when these events are inserted into the database, this means that the combination of data happens at write time. The approach in this research is different, it brings together the data at read time. Adding data to the analytics when writing it into the database can inflate the analytics storage a lot, that is why usually only a subset of the data is inserted into the analytics. Only adding a subset of the data leads to the exact missing data problem this research tries to solve. Instead only adding the required links between data to allow combining it at read time solves the problem, which has been researched, implemented and tested in this project.

When the solution for combining profile and usage data works a funnel visualization will be made. This implementation will track users even if they are identified by multiple different tokens. The visualization will support merge/split, so that processes that have multiple paths towards an end goal can be shown correctly. The main processes of Staying (property sign-up and guest invites) will be used as test cases. Together with Staying the test cases will be reviewed and the interface of the funnels will be improved upon using the feedback.

1.11 Report structure

First the status of the research field is described in Chapter 2, to understand what has been accomplished and where the challenges are. Chapter 3 describes the requirements of an analytics system that can combine profile and usage data. Chapter 4 describes the architecture of the system for combining analytics and applications data, and use it to answer questions and build funnels. Chapter 5 describes the implementation of the system as described in the architecture. After the system is build Chapter 6 describes how it is validated for answering analytics questions, Chapter 6.2 describes how it is validated for building funnels and Chapter 6.4 validates the performance of the system. The results of the project are described in Chapter 7.1. Related work about database solutions, analytics tools and immutable databases can be found in Chapter 7.2. Future work that can be done to improve the solution is described in Chapter 8.4.

Appendices can be found after these chapters with additional information, used as context for defining requirements and making architecture and implementation decisions.

2 Existing research and solutions

This chapter analyzes existing work in the field and the current analytics solutions used by companies.

2.1 Analytics systems

There are a lot of analytics systems available for companies to use, with varying features, price and possible uses. In Appendix A the most used tools can be found, with descriptions of their features. The features of these tools, from showing simple statistics to showing exactly how many users followed a certain process, are used for the requirements set for the system that will be build in this project. After the common features of these systems have been determined, the system architecture of analytics systems has also been identified. In Appendix B the requirements and architecture for a generics analytics system are described. This has been done to know the current possibilities of analytics and to gather information to create a system that can combine usage and profile data. The first feature that has been identified is answering simple analytics questions (like the number of visits of a web page, how many times a certain action happened, etcetera). Build on the concept of answering questions there are more advanced features, like funnels and charts. Funnels combine simple questions in a step-by-step process to get information about for example sign up of a user on a website. Charts are a logical extension of simple questions, because companies often also need to see the result over time.

There is already a lot of research about usage of analytics systems for tracking websites and apps, for example using Google Analytics in the food industry [25], tourism industry [26] or measuring and improving a library website [21]. These studies are relevant for knowing how companies and researchers use analytics tools, so that the system build in this project can also support these activities. There are also different tracking approaches like click heatmaps [19], which are used to track mouse movement of users. More tracking options are available, but after capture of everything that happens on a page almost any visual representation can be created using the available data.

Currently there are so many analytics tools used that people start using browser extensions like Ghostery [8] to block all trackers on visited websites. Often is not necessarily data collection itself that is the problem (at least if it is anonymous), but the bandwidth used by loading the trackers and the slow page load caused by them. This is especially the case on websites with advertisements, since each advertisement comes with a couple of trackers to measure their effectiveness.

2.2 Usage (analytics) data storage

For storing usage data a lot of different databases types and implementations are available. Single server options are available, but also multi-server options like Hadoop [27]. There are different database types in use: relational, key-value stores, document databases, wide column stores and graph databases. Each of these has numerous implementations that can be used of which the most used ones are listed in Appendix B. This appendix also describes requirements of databases used for analytics storage and their architecture. Multi-server file systems like Hadoop are intended for file storage, and are not suitable for a relational database. Key-value stores are used to map keys to values, which is really efficient for certain usage patterns, but storing all profile data in it will likely not work. They are also not meant for storing usage events, since they have no concept of time and order, therefore cannot take advantage of usage events that are sorted by time. Since especially for combining profile and usage data joining tables is important, document databases are also not ideal. Multi-server scaling is important for analytics storage, since the data will always keep increasing. Efficient time filtering is also important, since often getting result will be limited to a certain day/week/month, which should limit the search to only a small part of the collected analytics.

2.3 Profile (application) data storage

Storing profile data has different requirements as storing usage data. It depends a lot on the particular application that the company has, but generally speaking relational databases are used often. Different implementations exist, like MySQL, PostgreSQL and Oracle. The requirements for profile data will be discussed in Chapter 3, which are based on experience working at companies and the public data available for companies. There are plenty of research projects that design and implement new databases, but this is not relevant for this project. This is because to make the solution available for as much companies as possible it should work with the most commonly used database types, and not necessarily experimental or new databases.

2.4 Existing funnel visualizations

There are existing funnel visualizations available, which solve certain use cases already, but not all of them. Google Analytics offers some tools for this space, it allows defining a goal, which could be creating an account, visiting a certain page or something else. After creating this goal a path of actions can be defined that users are expected to take. After defining these the "Goal Flow" page shows the statistics for the defined funnel. An example can be seen in Figure 3. This example shows users going to the home page of the site, and after that going to the vote page, which is a common path for this data source. Google analytics shows the drop off per step, including which page the users dropped off to. This example also shows that a funnel is not necessarily without loops. Google Analytics draws a line from the vote step back to the home step, indicating that some users went back to home. This shows that funnels are actually graphs, and showing them as a sequential list of steps is a simplified representation. Google Analytics does show loops, but does not offer merge/split mechanics, so it is unable to draw the guest invite process of Staying. Another problem to get funnel in Google Analytics is that all events need to be sent there, and the missing data problem would occur again. Funnels can be made retroactively, but the events already need to exist, which makes it likely that there is an event missing.

Another product that has been looked at is HeapAnalytics, which also offers funnel visualizations. This product has some advantages and disadvantages. First of all HeapAnalytics collects every user interaction on the websites/apps

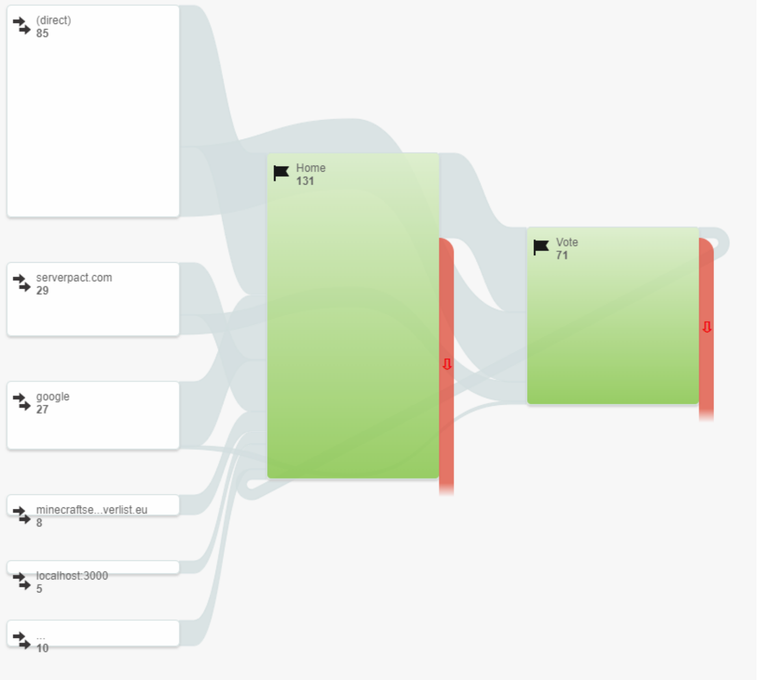


Figure 3: Google Analytics Goal Flow diagram

their software is added to. So because all data is captured, Heap allows defining events retroactively. This solves a part of the missing data problem, because events do not need to be sent explicitly as with Google Analytics. The funnel visualization of Heap, as seen in Figure 4, is however more simplistic. Context information is available for the users in a step, but will simply show a list of actions for each user. Heap still does not solve combining profile and usage data, since the only available option is to add all profile data to the events.

2.5 Challenges to be solved

Even though most areas of analytics usage and storage have been researched, combining them with profile data to get useful results is not clearly described and documented. The challenge in this field is to ensure combining these data sources is possible, is correct and is quick. Visualization using funnels is another challenge. Current solutions exist, covering basic use cases, but for slightly more complex scenario's they are not suitable. They also have no good solution for combining usage and profile data, except simply sending all user data along with each event (which is not guaranteed to be enough, and causes a lot of extra data traffic). Next to that current funnel visualizations do not support parallel paths, so if there are two ways to sign-up, they would need to be created and viewed separately. To get a solution that solves these challenges this research project implements a prototype system and evaluates it. Next chapters gather the requirements for such a system, create an architecture for the system, describe how it is build up and evaluate the performance.

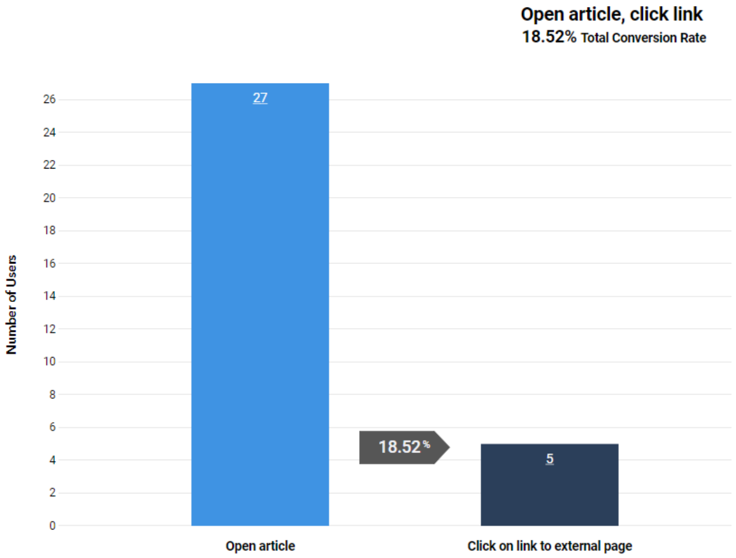


Figure 4: HeapAnalytics funnel visualization

3 System requirements

This chapter outlines the requirements that companies have for an analytics system that can answer questions, and can be used to create funnels with. First the type of questions a company would likely want to answer are detailed, which are used as a requirement for the system. Secondly funnels and the requirements for building a funnel are explained. Lastly the scalability and non-functional requirements are detailed.

3.1 Answer analytics questions

For the setup of the system that combines the profile and usage data trade-offs need to be made in terms of scalability (how much data can the system store, does it support multiple servers and multiple data centers), speed (how much events per second can it take in, how quick can the system combine and export data, how quick do SQL queries run), ease of use (setup time and complexity, can it be applied to existing systems, maintenance time), etc. To support these decisions it is important to know what kind of questions need to be answered by the system. These questions will be based on the needs of companies that build (web) apps, like Staying does. The identified questions will be used as a demonstration and benchmark for the system in the evaluation phase. The questions will be described for companies in general and for Staying specifically. This ensures the solution keeps in mind other companies and does not only work for the specific situation of Staying.

3.1.1 Generic questions

Below a number of questions are listed that could be asked by companies. These questions can either be answered using the usage data, the profile data or both. The base questions are as generic as possible, but can be extended by the additions listed below. The additions limit the scope of the question, making it more specific.

Base questions:

1. When did any action `<last/first>` happen?
2. How often did any action happen?
3. How long did an action last?
4. What is the distribution of property `A`? (like device size, android/ios, etcetera)

Additions:

- Specific action(s): limit to one ore more actions
- For entity: limit to an application specific entity (user, company, etcetera)
- In time period: limit to a specified time

3.1.2 Questions by Staying

In case of Staying the generic questions identified above materialize in the following questions:

1. When was the last activity from a venue on the portal?
   Indicates if the venue is still active, if not the venue could be approached to see why they did not continue to use the platform
   > derived from base question 1
2. When was the last activity of a user on the app and/or portal?
   For providing support and approaching inactive users.
   > derived from base question 1
3. How many times has each content item been viewed?
   To show on the portal, indicates which items are most important for the venue. For example it might be a good idea to hightlight the most visited items.
   > derived from base question 2
4. How long has each content item been viewed?
   Gives more information about the popularity of content items, could indicate if a guest actually reads the information in the item or immediately closes it again.
   > derived from base question 3
5. When did action X happen last in the app/portal for user/venue Y?
   Analytics events contain user actions, which have a type like 'login_success' or 'switched_venue', Staying often like to see when an action happened last to assist better when providing support or approaching a venue owner.
   > derived from base question 1
6. How often did action X happen in the app/portal for user/venue Y in time period Z?
   For example how often did a venue edit content items, how many times it visited the portal, etcetera. This can be used to focus the development on the most used parts, or focus marketing on features that are not much used.
   > derived from base question 1
7. Which device types are used per venue type?
   Find the percentage of mobile, tablet and desktop/laptop users per venue type (lodges, apartments, hotel). This information can influence the development of the portal to focus more or less on compatibility with mobile devices with a small screen.
   > derived from base question 4

3.1.3 Charts

While answers to the mentioned questions provides basic information about the state of the application and its users, a business often wants to know answers to these questions over time. Charts can be made using the question to show the answers for each day/week/month, whichever makes the most sense for the type of data. The effect of this addition to the demands of a business for the analytics system is that it needs to handle time based filtering and grouping efficiently.

To display charts of historic data a 'rollup' is often created, which means calculating results for completed time periods once and storing them for reuse. This rollup process can be used to store results per hour when an hour is over, after which the rollup of the current day can be either updated with partial results (if there are more hours to come in this day), or finished (if this was the last hour). This solution is well known and used to keep charts of historic data quick to calculate.

3.2 Build funnels: tracking of business processes

Besides questions as described above, most companies also want to track their important business processes. To track these processes a funnel can be created, as described earlier. The following sections explain which features/statistics are expected to be in funnels, and the challenges that need to be solved to implement these features in a funnel.

3.2.1 Funnel features

For a funnel to properly support the company in making decisions there are a number of important features that need to be available. Below is a list of features that, in this case Staying, needs:

1. Selection of the entities that should be tracked.
2. Tracking of single entities through the funnel.
3. Per step information:
   (a) Number of entities that made it.
   (b) Percentage of entities compared to the initial step.
   (c) Number of entities that stopped (possibly splitting into groups that have different causes).
   (d) Number of entities that continued.
   (e) Number of entities that rejoined (skipped the last step(s), but still did this one).
4. Custom statistics at steps (optional):
   (a) Distribution graph or table for a certain property of the tracked entity (for example user language).
   (b) Subsets of entities that performed a certain action (for example select 'remember me' checkbox)

3.2.2 Funnel challenges

The features that are expected of a funnel create a couple of challenges. The first challenge is to track entities through the complete funnel. In some cases there is already a unique way to identify an entity through a funnel, for example for existing users this might be their email or user id. However for the example funnel at the start of this section it is harder, their email could be used for tracking, but that would not work for sending a follow up email, since there is no way to distinguish the funnel of each email individually. To solve this challenge there needs to be a unique identifier for an entity that follows the funnel. This unique identifier might also be merged with other identifiers while going through the funnel. For example if the user signs up the created user could be linked to the identifier for further tracking. This allows for easier tracking since analytics after sign up could use the user information instead of the identifier from the first marketing email. A system for creating funnel needs to support multiple identifiers for each entity, allowing for better tracking.

A second challenge comes when a funnel splits or merges. Following a process could be drawn as a directed cyclic graph. It would mean the user can follow certain steps, in any order and repeat some steps when they want to. Tracking such activity and displaying it is really hard, so that is why funnels are often simplified. The most basic way would be a sequential list of steps, showing how many users made it to each step. This simplification already allows companies to track most processes. To support situations where a user can choose from multiple paths to get to the next step funnels need to be a bit smarter. This could mean introducing split and merge steps, to allow for multiple paths.

A funnel could split if two possible paths can be followed, meaning both need to be tracked and showed correctly. This complicates the result calculation of funnels and makes them harder to display. Care should also be taken to keep the implementation maintainable.

3.3 Non-functional: scalability and interface

For the system build in this research project there are a couple scalability requirements to ensure that the system can handle large data sets for big or quickly growing companies. The scalability requirements are as follows:

1. **Multi-server support:** spread data among servers, with a redundancy factor
2. **Redundancy support:** multiple physical locations.
3. **High throughput:** up to Booking.com scale (2 billion events per day, which is 83.3 million per hour, 23148 per second, where the average event is around 10kb in size).
4. **Low latency:** simple queries should be completed in 100ms or less.
5. **Good interface:** the interface showing the result of funnels should be clear and easy to use.
6. **Flexible:** the system should be flexible so that it can be adjusted to the situation of the majority of companies.
7. **Maintainable, secure:** changing and updating the system should be easy and taken into account with the design.

Multi-server support is a requirement because storing and using huge data sets on a single server is not possible. Spreading data among multiple servers spread the CPU load and scale the system horizontally (adding more servers) instead of vertically (using a single more powerful server).

Redundancy support is required because the data should be stored in multiple locations (data centers) too prevent data loss when one location is lost (due to fire, a storm, etcetera). Redundancy at the same location is also necessary, so that when losing a single drive or server does not stop the system from working. Therefore it should be possible to store data on multiple servers at the same time.

High throughput is the next requirement, which ensures that the system can scale with the company using it to handle large customer groups. High throughput is defined as the scale that Booking.com operates on, which is known from one of their blog posts [2] with some extrapolation. If the scale of Booking.com usage and profile data works with the system, it will be suitable for the majority of companies.

Low latency is another requirement for the system. This means for example a query getting data about a user should not take more than 100ms, but preferable a lot less. Since the acceptable response time depends a lot on the type of query that is tested, the validation chapter of this project will test a list of them to make clear that this requirement has been met.

The interface of the results that get out of the system needs to be clear. The interface showing the funnel should be easy to understand and read, following standard design principles.

The system should be flexible so that it can be adjusted to other companies than Staying, so that the solution is not too domain specific.

The last requirement is about the standard software principles, that a system should be secure and maintainable. For security some measurements can be taken to prevent problems in the future, so that the system will not cause harm to the company. Keeping the implementation clean and well designed will make maintenance easier, so that the system can be maintained by other people in the company without too much effort.

[2] https://blog.booking.com/using-riak-as-events-storage-part1.html

4 System architecture

This chapter describes the system architecture that will be used to store usage and profile data.

4.1 Structure

Figure 5 shows the architecture created for the system that will store usage and profile data, and can combine it to generate the desired statistics. At the top there is an 'Applications' block, these are the systems that will read/write the database. For example apps and website will get the data from the system for displaying to the user. Getting this data might go through a backend, which in turn connects to the storage system. The applications also generate usage data, for example the app sends which screens are visited.

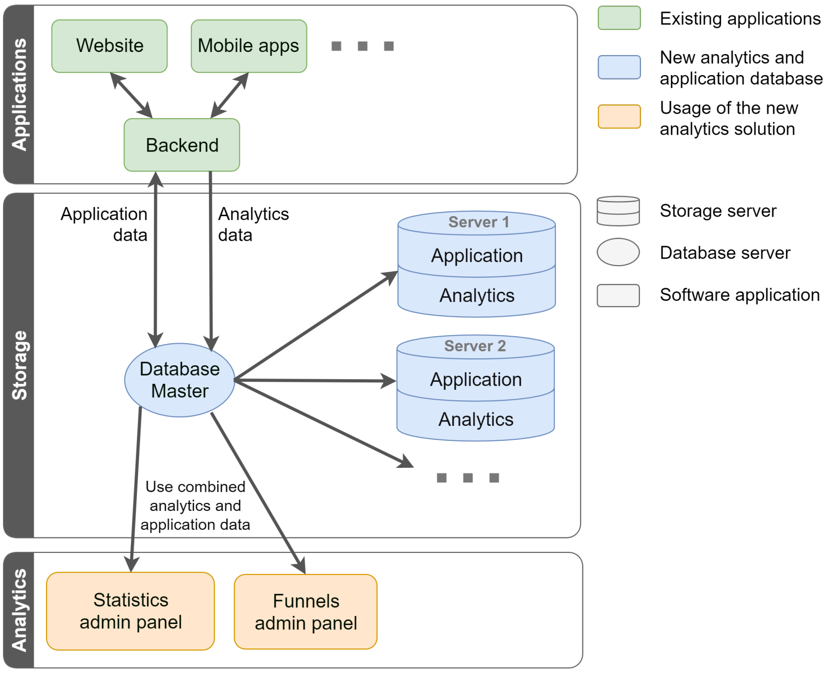


Figure 5: Usage and profile data storage architecture

The storage block of the figure shows the database servers, which hold the usage and profile data. The 'Database Master' is the server that applications talk to, while 'Server 1', 'Server 2', etcetera are hidden storage nodes. This multi-server setup uses the PostgreSQL relational database system software in combination with CitusData that allows it to scale to these servers. Section 4.2 explains this in detail.

The lowest block in the figure shows information retrieval from the storage system, used to generate reports. This operation combines information from the usage and profile data to generate statistics. Section 4.6 describes how analytics questions can be answered using the system, and Section 4.7 describes how to build a funnel.

4.2 Multi-server storage

The storage in Figure 5 is based on PostgreSQL. This is a commonly used relational database system that uses SQL for CRUD (create, read, update, delete) operations on the data. PostgreSQL on its own is meant to be used on a single server (with exception of replication servers for having a live backup). This means that the data size is limited to that of one server and the performance is also limited to one server. Nowadays quite powerful servers can be rented or bought, but this still imposes a hard limit on performance. In this case the most relevant performance aspects are CPU speed, RAM size, network speed and disk speed and size. At the time of writing Amazons EC2 Virtual Private Server (VPS) hosting offers a server type with 128 cores, 1952GB of memory and a 25 Gigabit network connection (called 'X1 Extra High-Memory 32xlarge', with internal name 'x1.32xlarge'), which would keep a database running for quite some time. However, vertical scaling is only a solution for companies that have a limited data set, that is growing slower than the technology of one machine improves. Therefore horizontal scaling is necessary for a lot of larger and data-heavy companies.

4.3 Database technology: PostgreSQL+CitusData

There are a lot of multi-server database solutions, of which the most popular ones are described in Appendix B. For this system PostgreSQL with the CitusData extension has been chosen. The first reason of this choice is that this solution supports almost the full SQL language, which is widely used and known, making it easy to adapt. The second reason is that importing data from any relational database is easy, with most likely only minor SQL schema changes. This means that if a company currently has their data in a relational database, it is an easy migration path to this solution. A third reason is that a PostgreSQL and CitusData combination can be setup on any server, making it easy to create a prototype for this research project. Even though the prototype created in this project uses PostgreSQL and CitusData, the architecture and solutions of this project can also be used with other technologies.

CitusData is an extension for PostgreSQL that introduces sharding, a way to spread data across multiple servers. It starts with the 'Database Master', this is a server used as entry point for the database, which is what all applications talk to. This master does however not store any actual data itself, it only knows where the data is located. The actual data lives on the worker servers, which are connected to the master and handle the commands (SQL statements) passed to them. Data in this database is organized into tables just like a regular PostgreSQL instance. Now these tables are distributed, using a certain value as distribution key. This means that for example the events table will have a column with data that is used as distribution key. The value of that column will determine which of the worker servers will store the data. A simple example would be that all even-numbered users are stored on server 1, and all odd-numbered users are stored on server 2. In practice it is a bit more complicated, it starts with computing a hash of the distribution key. The result of the hash operation determines which server it should be stored on, each server stores a part of the hash space.

4.4 Profile data storage

The database as described previously can easily store the profile data of companies. This database works for storing users, messages, chats and a lot of other data. Since the database setup supports all regular PostgreSQL data types (and anything provided by extensions), storing the profile data is easy.

An important aspect of profile data storage is that it can be distributed among multiple servers with a distribution key. A common way to distribute profile data is to group data per customer, CitusData calls this a 'multi-tenant' application. Staying provides its application to multiple 'venues' (hotels, apartment complexes, etcetera), so that is what the 'tenant' is for Staying. This means that data can easily be split per venue, so that is used as distribution key for most data tables in the application.

Most applications have such an entity that can be used to split all data, if not the performance of the application could degrade depending on the queries sent to the system. Performance of queries that only use one tenant is as good as with a single PostgreSQL database since the query can be sent to the server that has all data for that particular tenant. If a query use data of multiple (or all) tenants it can either let each server calculate results for its own tenants and combine the results in the database master

With this setup the profile data can be stored and queried, fulfilling this requirement.

4.5 Analytics events storage

Analytics storage will, just like the profile data storage be in a simple table. To keep analytics storage flexible, but at the same time have structured data that can be used for answering questions and building funnels a storage format has been designed. Analytics normally consists of simple events, each time something happens an event is put into the system, describing what happened (including relevant context information). A simple schema can be used, with just an id, distribution column and data column. The data column is a `jsonb` column, which can hold any JSON data. There are a few reasons for using a single data column, the first is to prevent a table with 100+ mostly empty columns. A second is that inserting a single JSON document into the database is a lot easier and quicker than putting everything into the right column. Since PostgreSQL has good support for JSON columns, with indexes and a bunch of JSON functions it does not hard usability. The flexibility this storage format offers is welcome, it allows for storing events with the complexity and size companies need.

To get consistent data in the data column a schema has been defined to structure it, this schema can be found in Listing 1. First there is an `at` key, which stores the timestamp of the event. This timestamp is the number of seconds since 1970, with decimals if needed. This timestamp should reflect when the event happened, and not when the event was collected or put into the database. So this could for example be the exact time that a user clicked a button. Next there is a `type` key, which is a simple string to indicate the type of event. To provide more information about the event an object can be added using the `type` as key. Other keys in the top-level can be used to note down globally relevant information, like the system that generated the event, the user that the event is about and other references to application tables.

Listing 1: Schema for data of analytics events

```
# Event generated at (seconds since 1970)
at: number
# Top-level type of the event
type: string
# Object with more information about this type of event
<type>: object
# User that this event is about, reference to an application table
user?: number
```

To structure events further, the information object located under the key with the name of the type can again contain a `type` key and more information. Listing 2 describes the structure of this information object, which can recursively be used again. This organization allows placing information at the level where it is relevant, without polluting other levels.

Listing 2: Sub-schema for type-specific data

```
# Sub-type of this events
type: string
# Object with more information about this type of event (same schema as
    this listing)
<type>?: object
# Any other key can be used to store information that is relevant for
    all events with the same parent type
```

To make it more clear, see the example in Listing 3. This example shows an event for an email that has been sent. At the top level it indicates which user and venue this is for, which are profile data tables. The top-level type is `"email"`, indicating that it has something to do with an email. In the `email` key more details are given, the provider that handled this email is `"mailgun"` and the id of the email is `"abcdef"`, this is information relevant for all email event types (for example sending, receiving, reading, etcetera). A sub type `"sent"` is in the event to indicate that this email has been sent. The `sent` key indicates more details about where the email is sent to and from, and what the subject is, this is specific information only relevant for sending email.

Listing 3: Example of using the event schema

```
{
    "at": 1498780800,
    "user": 123,
    "venue": 45,

    "type": "email",
    "email": {
        "provider": "mailgun",
        "id": "abcdef",

        "type": "sent",
        "sent": {
            "to": "Thijs",
            "from" : "University of Twente",
            "subject": "abc"
        }
    }
}
```

Usage of the event data schema as described above is up to the company implementing it, but this structure helps with the combination of application and event data in a couple of ways. Firstly, all references to profile data are consistently at the top-level of the structure, making it easy to join with entries in those tables. Secondly, events are structured and can easily be filtered by type, making aggregating results easy.

The described architecture of analytics storage can easily store a lot of events and introduces structure to the events to be able to combine them with profile data and clearly type the events with a recursive type structure. This setup provides flexibility while keeping data structured, this is a middle ground between fully 'schema-less' event storage and completely typed event storage.

4.6 Answering analytics questions

With the architecture and data storage figured out, it is time to use the data for answering analytics questions. The type of questions that the system should be able to answer have been determined in Section 3.1. To answer these questions SQL queries can be used. Some of the questions are simple to answer, because they use only a single table and use the distribution key to group the result. Other queries however are combining multiple tables, which is the interesting case this project is geared towards. A common combination of tables is to combine the usage data with profile data. To combine these data sources there needs to be a common field, in this case the ids used in the application database end up in the data of the event. The previous section showed that for example the user id and venue id are added as top-level keys in the event data. This means that combining usage data with venue data is a simple join on the id of the venues table, and the `venue` key of the data in the events table. The same applies for all other profile data tables, the only thing that needs to be added to events about them is the id, all other data can be combined when reading the data to answer a question.

Because of the architecture of the combined database for usage and profile data answering questions is now reduced to simply performing queries in the database. This completes the required architecture for answering the defined analytics questions.

4.7 Building funnels

For building the funnels as described in the requirements of Section 3.2 there needs to be a way to get, store and display the required information. Below the general structure of funnels and how they are calculated is explained.

4.7.1 Required funnel data

To explain which data is required for calculating and displaying a funnel with the features as described in Section 3.2 the following example funnel will be walked through step-by-step.

1. A company sends a newsletter email to subscribers.
2. The subscriber opens and reads the newsletter email.
3. The subscriber clicks on a link to the website.

This funnel will measure how often newsletters sent by the company are opened, and how often this results in visiting the website. Since a funnel always tracks a certain entities through all steps, the first step is to determine which type of entities the funnel will track. In this case the funnel needs to users, these users will receive a newsletter, might open/read it, and could visit the website by clicking a link. In this case a user is identified by its email address, this is however not specific enough to gather data for the funnel. Because the company sends a newsletter each month, the email address alone is not a good way to identify users through the funnel. To properly track users through the funnel a combination of the email address and email itself is required, this could be a unique number attached to each email.

Step 1: Which emails have been sent?
Now that it is clear which entity is tracked in the funnel, a couple of funnel principles need to be explained before carrying on with the architecture variants. First the set of users to track in the funnel is selected, this could for example be based on the time the email is sent, for example to see how the November 2017 newsletter did. For each email that is sent to customers the company puts an event into the database as shown in Listing 4. Important is that it has a unique email id and contains the email template and the newsletter year and month (for filtering).

To start the funnel, a selection of entities (in this example, emails to users) can be made using these email events. Data from the event and combined profile data of the user can be used for filtering. When the emails that will be tracked in the funnel are selected, the starting count of the funnel is known (for example 100 emails). These same email+user combinations will be followed through all steps of the funnel. So in each next step of the funnel the system needs to find out which of the original email+user entities have completed that particular step. So the result of tracking the first step is a list of email identifiers have started this first step.

Listing 4: Example of an event for sending an email

```json
{
    "at": 1498780800,
    "user" 123,

    "type": "email",
    "email": {
        "id": "2017-11-123456",

        "type": "sent",
        "sent": {
            "template": "newsletter",
            "year": 2017,
            "month": 11,

            "subject": "Newsletter November 2017"
        }
    }
}
```

Step 2: Which emails have been read?

The next step of our example funnel is reading the email, to track this the company collects events like the one in Listing 5. For tracking email reading it is possible to add an image to the email, that will be downloaded when the user opens it. When the server detects that the image is downloaded it creates the event. The event contains the same `id` field inside the `email` object, which can be used to see which email has been read.

Now a list of email identifiers needs to be selected from the database using this email read event. It is important that only email identifiers are selected that have joined this funnel in the first step, otherwise results will be wrong. Note that the initial step of the funnel might be limited to a certain day/week/month, but following steps are generally speaking not limited, unless the company only wants to count reading the newsletter if it is within a week. Care also has to be taken that opening the same email twice only counts once, the funnel is about how many users go through, not how often they complete certain parts.

Listing 5: Event for reading an email

```
{
    "at": 1498780801,
    "user" 123,

    "type": "email",
    "email": {
        "id": "2017-11-123456",

        "type": "read"
    }
}
```

Step 3: Which emails lead to website visits?

The last event of the funnel looks like in Listing 6. It contains the url that has been visited, and the parameters of the visited page. In this case the parameters contain the email id so that this visit can be linked to the link that has been clicked in the email. Getting results for this step again needs to take into account which email identifiers have started the funnel, just like the previous step.

Listing 6: Event for visiting the website

```
{
    "at": 1498780802,
    "user" 123,

    "type": "website",
    "website": {
        "url": "https://company.com/article",
        "params" {
            "email_id": "2017-11-123456"
        },

        "type": "visit"
        "visit": {
            "session": "abcdefghijklmnopq"
        }
    }
}
```

Calculating results

With the email identifiers of each step the results can be calculated. The following calculations can be done to gather the results for displaying the funnel:

1. Users that made the step: Count the number of email identifiers.
2. Users that stopped between two steps: Identifiers that are in the first step, but are missing in the second step.
3. Users that rejoined in a step: Identifiers that are in the current step, but are missing in the previous step.

The information above can be used to display how many users made it to each step. The percentage of users that made it to a step can also easily be calculated by comparing it to the users that joined the first step.

4.7.2 Funnel data format

To separate the calculation of the required data from the visualization, an intermediate data format is used. This ensures that the software to get the results from the data source is decoupled from the visualization. This allows to easily swap the interface with different visualizations, making it more modular. It also supports swapping out the calculation of results so that for example, support for a different database product can be implemented.

Theoretically funnels could be graphs, but a simplification is applied to create a readable result. Other analytics tools simplify to a sequential list of steps, but this project has a more advanced structure. It allows a step to have parallel tracks, and these tracks again have steps. The structure of this format is shown in 7, the definition is given in TypeScript types [3], since these are clearly documented, modular and could be used when building a visualization.

It starts with the `Funnel` at top level. This defines the start and end time of the funnel, and a list of options. These options are fields that can be filled in by users, to be displayed in the visualization. `Track` contains a title and a list of steps. The steps are a mix of type `SingleStep` or `ParallelStep`.

`SingleStep` contains absolute and relative numbers for identifiers that made this step, how many came from the previous step, how many continued to the next step, how many stopped, and how many rejoined. It also contains exit data, which can split the people of the `stopped` key into sub groups.

It also contains an array of `Note`s, which provide context about the step. A `Note` can be of different sub types, which each have their own data. `StatisticNote` contains one simple number and title. `SubsetNote` contains a number and percentage, it could be used to indicate that a certain percentage of users in this step have completed a certain action or meet a particular criteria. `TableNote` contains data for a table, the data array maps column names to values (which are any string). `GraphNote` contain data for a simple bar graph and defines the x-axis name, y-axis name, maximum axis values, and has a data array. This could be expanded with different graph types in the future, like pie charts, line graphs, etcetera.

`ParallelStep` defines a step that has multiple parallel tracks. Each track again has a title and a list of steps, as described earlier.

Listing 7: Structure of funnel data (TypeScript data format)

```
interface Funnel {
    start: number
    end: number
    track: Track
    options: Option[]
}

type Option = NumberOption | SelectOption
```

[3] https://www.typescriptlang.org/docs/handbook/basic-types.html

```
interface NumberOption {
    type: "number"
    label: string
}
interface SelectOption {
    type: "select"
    label: string
    options: string[]
}

interface Track {
    title: string
    steps: Step[]
}

type Step = SingleStep | ParallelStep
interface SingleStep {
    type: "step"
    title: string
    subTtile?: string
    result: number
    totalPercent: number
    fromPrev?: number
    fromPrevPercent?: number
    toNext?: number
    toNextPercent?: number
    stopped: number
    stoppedPercent: number
    rejoined: number
    notes: Note[]
    exit: ExitData[]
}
interface ExitData {
    title: string
    result: number
}
interface ParallelStep {
    type: "parallel"
    tracks: Track[]
}

type Note = TableNote | GraphNote | StatisticNote | SubsetNote
interface StandardNote {
    title: string
}
interface StatisticNote extends StandardNote {
    type: "statistic"
    result: number
}
interface TableNote extends StandardNote {
    type: "table"
    data: Array<{[key: string]: string}>
}
interface SubsetNote extends StandardNote {
    type: "subset"
    subset: number
    subsetPercent: number
}
interface GraphNote extends StandardNote {
    type: "graph"
    graph: {
        x: string
        y: string
        xMax: number
        yMax: number
        data: Array<{
            x: number
            y: number
        }>
        xMap: {[key: number]: number}
    }
}
```

In this project the data for the funnel is generated using PHP code that connects to the database. The implementation of this PHP code and how to define funnels with it will be explained in the System implementation chapter. Instead of PHP code to build the structure and data of a funnel, a GUI could be created. In this project the goal is to make a flexible tool, that is used by people that know how to program. Since programming a funnel in a programming language offers a lot of flexibility this is suitable in this case. For other companies it might make sense to let marketing or sales employees build funnels, in which case PHP would not be suitable. My recommendation for those companies would be to either create a DSL (Domain Specific Language) to create funnels, which is at a higher level than raw PHP code. You could also go a step further and build a graphical user interface on top of the DSL or other implementation, so that not code needs to be typed to create funnels. I think this would reduce flexibility though, and make the solution more limited.

Another interesting idea is to use an existing business process modeling language, like BPMN (Business Process Model and Notation), to generate the basics of the funnel. This might give a good starting point for defining the funnel, but you would still need to map events in the usage data to the steps of the funnel. So this will not fully replace a programmatic or visual user interface for defining funnels.

4.7.3 Multi-identifier tracking

In the example used above there is a single identifier for each user in the funnel, the number that assigned to the newsletter email. This identifier is used in each step to check which users made it. In more complex funnels there is often not a single identifier, or it is hard to make sure the identifier is carried all the way through the funnel. To make it easier to track these long funnels, multiple identifiers could be used per user.

Using multiple identifiers per user works as follows, first there is a 'base' identifier, which all tracked users start with (for example the number of an email).

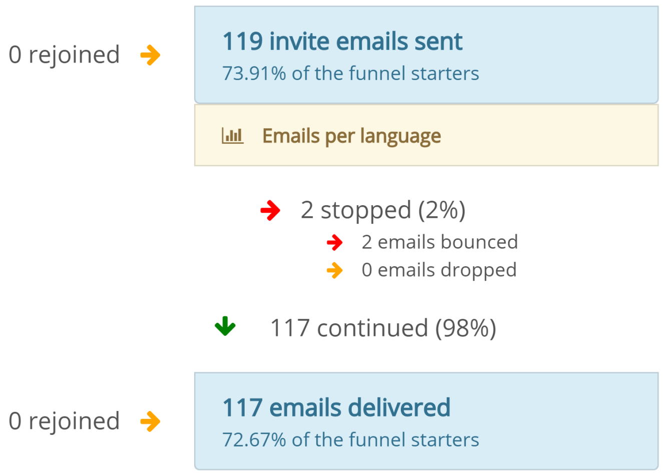
After following the funnel for a couple of steps with this identifier (for example email delivered and email read) there is a step where another identifier is introduced. In that step the first, and a new second identifier is known, so they can be linked to together (for example user creates an account on the website). After the second identifier is linked to the first one, all following steps can use the second identifier to indicate which users made it. The system will translate the second identifier to the first one, and can calculate and show the results as usual.

This concept will be implemented into the funnel to make them suitable for complex and longer funnels.

4.8 Visualizing funnels

This section will discuss how the interface of the funnel visualization has been designed.

The funnels of Google Analytics and HeapAnalytics have the time axis of the funnel horizontally, where the first step is on the left and further steps are right of that. This has as downside that horizontal scrolling is needed if the funnel is more than about four steps. Because my implementation will include notes for each step, and merge/split mechanics this would not be good to display horizontally. Therefore the time axis is displayed vertically for this project. Meaning that first there is a horizontal bar for the first step, with title and description. Attached to this bar there could be notes of different types. These notes should not be bigger or more prominent than the steps themselves, because that could confuse the user about them. Therefore larger notes like tables and graphs will be collapsed by default, only showing their title, and will open when clicked.



0 rejoined

119 invite emails sent

73.91% of the funnel starters

Emails per language

2 stopped (2%)

2 emails bounced

0 emails dropped

117 continued (98%)

0 rejoined

117 emails delivered

72.67% of the funnel starters

Figure 6: Funnel arrow usage

To show how a user goes through steps arrows will be used. From one step to the next there is a green arrow pointing down towards the next step. This arrow will have the absolute number of users and how much percent of all users of the previous step next to it. Next to this green arrow there will be a red arrow pointing to the right, indicating how many users stopped, again with an absolute number and percentage. If more is known about why a user stopped, these groups of users could be displayed with smaller red arrows below the large red arrow, to indicate the large arrow splits into several subgroups. Red arrows could be made orange if the number is zero, giving a hint that nobody stopped. As last there could be rejoining users from previous steps, this is visible by an orange arrow at the left of a step, pointing to the right towards the step title. Figure 6 shows the usage of all these arrows. Together these arrows show all paths users can take, and indicate if the path is good or bad. Google Analytics also displays the bad paths as red.

Merge and splits of funnels will be displayed by first showing a split bar, to indicate something special happens in the funnel. After that two or more parallel funnels will be displayed next to each other horizontally. This works nicely for 2-3 parallel paths, but requires horizontal scrolling when using more of them. Having more than three parallel paths is however a much more uncommon scenario than having a funnel with more than three steps, as would cause horizontal scrolling for existing products. After the parallel paths a merge bar is displayed, indicating that the funnel is back to a single track.

5 System implementation

This chapter describes the implementation decisions and details of the system that has been created to test the architecture of the previous chapter. First the required existing systems are described, after that the application and analytics storage is set up. Next the implementation of funnels is described in detail.

5.1 System environment

Before the implementation starts it needs to be clear in which environment the system will be integrated. The assumption is that there is an existing (single-server) relational database for the profile data, or that the used database can be migrated to a relational database. There could be one or more application (like websites, mobile apps, backend services) that use this profile data. Next to that there is the need for an event based analytics system that tracks certain actions. There could be existing analytics solutions, in-house or by an external company, but that should not interfere with implementing the system as described in this thesis.

The following assumptions are made about existing systems:

1. Profile data can be migrated to a relational PostgreSQL database (either because it is already stored in a relational database or because a migration path is possible).
2. Profile data has unique identifiers for all important data types (like users, chats, messages, etcetera).
3. There is a clear 'tenant' in the profile data, which can be used to split the data among multiple servers.
4. Applications that use the database can connect to a PostgreSQL database.

Note that assumptions about compatibility with PostgreSQL are specific for this implementation, in general the solutions in this thesis can also be implemented using a different database technology.

5.2 Database: PostgreSQL and CitusData

First of all the servers that run the database need to be set up. There are a couple of options available to do this with CitusData, which will be explained briefly below.

Managed Citus Cloud Deployment
The easiest deployment is to order a Citus Cloud deployment, which is simply selecting the size and storage capacity of the cluster, and it is ready to go. With Citus Cloud there are options to select the amount of RAM and CPU speed of the database servers, and the available storage size. Up to 20 servers can be ordered in the cluster, with servers up to 32 CPU cores and 244GB of RAM. The storage capacity per server can be selected from 512GB to 2TB. Next to this a 'High Availability' option can be selected, which duplicates a set of server to another datacenter.

The setup of Citus Cloud is easy, but there are limited options and the price is higher than a manual setup.

Amazon CloudFormation Deployment
A more customizable option is to use the CloudFormation template that Citus provides to setup a database cluster using Amazon. CloudFormation works with a JSON template, which specifies the servers to start, which software to run on them, files to put on them and network and organization connections that should be made. The template sets up a cluster using an automatic scaling group of Amazon, which makes it easy to scale up the cluster. Downsides of this setup is that upscaling does not automatically add the new servers to the database, and that the company itself is responsible for the instances of the database, instead of a database service provider.

This setup is more customizable, but also introduces more responsibility for the user of the database to maintain and monitor it correctly.

Custom Deployment
The most flexible option is installing it on servers bought/rented by the company. In this case the type of hosting, servers and software can be chosen freely. The downside is that linking the servers together and maintaining the cluster is all up to the company. A company would probably need to create some tools to manage a custom cluster like this, so that they can scale it up or down, run updates and do other required changes.

For the tests in this project the CloudFormation template deployment has been used. Instructions to use each of the options described above can be found on the documentation page[4] of CitusData.

5.3 Profile data storage

After installation of CitusData on the master and worker servers following the documentation, the tables need to be created. The first step of creating a distributed table is to create a normal SQL table, as shown in Listing 8.

Listing 8: Create venues table

```sql
create table venues(
   id bigint primary key not null,
   name text,
   city text
);
```

Now that the table is created it is still a regular table, living on the master node. To change this the `create_distributed_table` command provided by CitusData is used as shown in Listing 9. The first argument of the command is the table name, the second column to be used as distribution key. Upgrading a table to a distributed table will move all existing data (if any) to the worker nodes, according to the distribution key.

It looks a bit odd to use a select to perform this operation, but that is an easy way for an extension to let users execute custom commands.

[4] https://docs.citusdata.com/en/latest/installation/production.html

Listing 9: Promote venues table to a distributed table

```
select create_distributed_table('venues', 'id');
```

After the distribution as done in the last step, running a query that uses this table will get send to worker nodes. Listing 10 shows the execution plan of `select count(*) from venues;`. Line 3 shows that the aggregate count will be calculated. Line 4 shows that it will do a custom scan using the "Citus Real-Time" executor. This executor divides the query among the workers, in this case it simply passes the query to the workers and adds the result of each worker together. It shows that there are 32 tasks, meaning that there are 32 shards, which is a fancy way of saying that the venue ids have been divided into 32 buckets. And each buckets gets assigned to one of the servers. On line 7 and further it shows one of the 32 tasks (since the plan will normally be the same for all tasks). Line 8 shows which node (database server) this specific plan comes from. Line 9 and 10 show that that is will do a scan on the venues table and count the rows, which is exactly the plan that shows up if this would be a regular table.

Listing 10: Running `explain select count(*) from venues;`

```
1  QUERY PLAN
2  ------------------------
3  Aggregate (cost=0.00..0.00 rows=0 width=0)
4    -> Custom Scan (Citus Real-Time) (cost=0.00..0.00 rows=0 width=0)
5      Task Count: 32
6      Tasks Shown: One of 32
7      -> Task
8        Node: host=<hostname of worker node> port=5432 dbname=postgres
9        -> Aggregate (cost=355.30..355.31 rows=1 width=8)
10         -> Seq Scan on venues_102818 venues (cost=0.00..314.64
               rows=16264 width=0)
```

The process above for the venues table can be repeated for more profile data tables, in case of Staying the 'stays', 'users', 'chats', 'message' and 'content' tables can be added. A table like the users table cannot be distributed by the venue, since a user can be the owner of multiple venues. So this table can be made a reference table, which means it will get stored completely on each database worker. Listing 11 shows the query to create the users table (simplified users with only an id and a name).

Listing 11: Create the users table

```
create table users(
    id bigint primary key not null,
    name text
);
```

To upgrade the users table to a reference table the query shown in Listing 12 is used.

Listing 12: Promote the users table to a reference table

```sql
select create_reference_table('users');
```

5.4 Analytics storage

Now that the venues table is created, an events table can be added. This table will also use the venue id as distribution key, will have its own id and have a data column. The SQL statement for creating this table can be found in Listing 13. Note that the primary key consists of the id column and the venue column, having the venue column in this primary key is required to use it as distribution key.

Listing 13: Create venues table

```sql
create table events(
    id bigint,
    venue bigint,
    data jsonb,

    primary key (venue, id),
    foreign key (venue) references venues(id)
);
```

Just like the venues table, the events table is promoted to a distributed table, see Listing 14.

Listing 14: Promote events table to a distributed table

```sql
select create_distributed_table('events', 'venue');
```

Now both tables are distributed by venue id, meaning that the venue data and events of that venue will be on the same worker node. To show that this setup works, lets follow a simple example. The company wants an overview of the number of events per venue, lets get that from the database. The query as shown in Listing 15 is used to get the result. This query is correctly distributed, meaning that each worker calculates the result for the venues it stores, and the master only combines the results of all the workers.

Listing 15: Get the events per venue

```sql
select
    venues.name as venue,
    count(*) as events
from events, venues
where
    events.venue = venues.id
group by venues.id;
```

5.5 Funnels

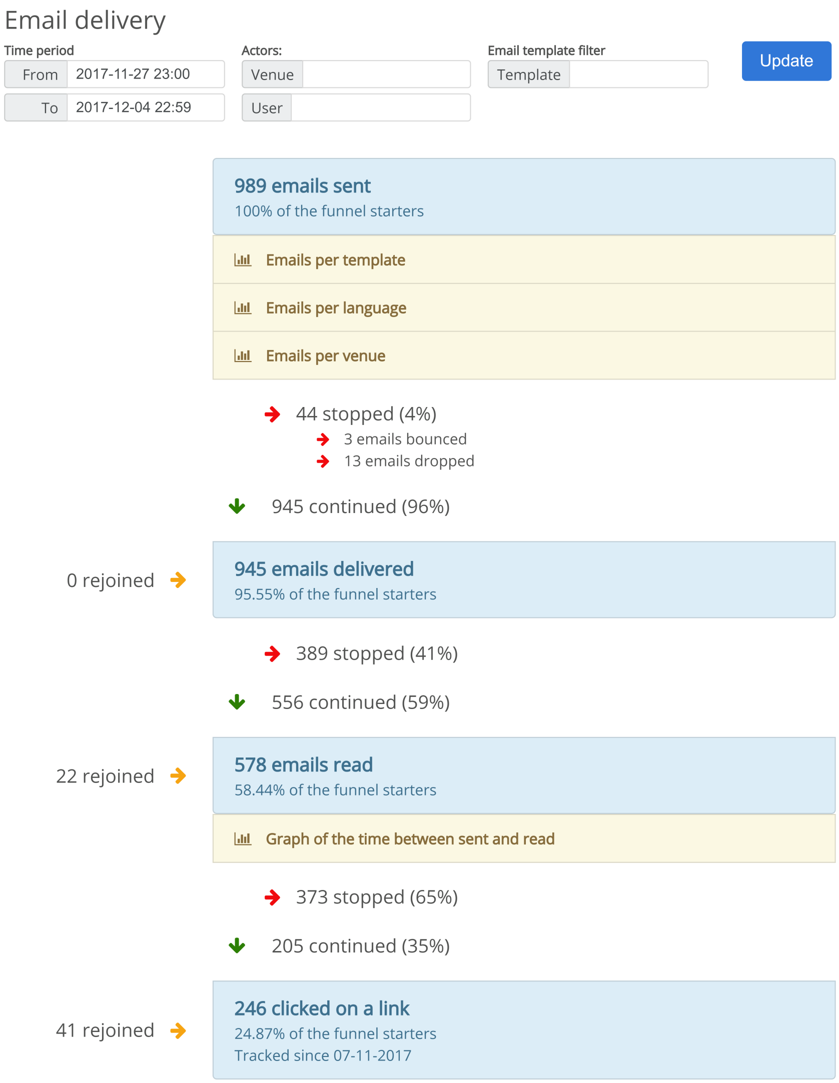
Two different implementations have been made to get the funnel data. The first implementation is simple and easy to build, but does not scale well to a large number of events. This implementation is described first to show how results of a funnel are gathered and calculated. The second implementation is a more scalable implementation, built to solve the scalability issues found in the first implementation. Both implementations use PHP for application code, connecting to the PostgreSQL database to gather data and using Smarty (simple HTML template engine) to display the results. Both implementations follow the structure as defined Figure 8 of the architecture chapter. First the implementation of the funnel display is explained, since that gives a good idea about the results the implementations need to provide. Afterwards the implementations used to gather these results are explained in detail.

5.5.1 Funnel display

To display the results of either of the funnel implementations described above there are plenty of options. A native interface, an image or a web interface are main options of displaying the funnel results. To let it work on any device with a web browser the display for the funnel of this research project is a web interface. A web interface is also practical because user input is easy with forms, which allows for filtering the funnel to a certain time period (or another criteria). This project uses PHP with the Smarty template engine to create a simple web page that displays the results. Any other backend that produces the funnel data or frontend that displays the data and gathers user input could be used, below the implementation used for this research is discussed.

Steps
Figure 7 shows the result of a funnel. This steps of this funnel are sending an email, delivering the email, reading the email and finally visiting the website using a link in the email. These steps are displayed in blue on the web page. The funnel starts at the top of the page, and following steps are displayed below. Each step shows how many users made it to the step, and which percentage of users this is compared to the users in the first step. This percentage is important because it is what companies look for. In the example of the image the company is interested in how many people visit their website, while an absolute number shows how many visit the website, the percentage can be used to compare the funnel. For example this can be compared to a previous or future newsletter, another audience or something else.

Continued and Stopped
Besides the number shown inside each step, there are numbers that connect the steps. The ”continued” number shows how many users did next step after the previous one. It also shows a percentage, because that indicates how good the conversion from one to the following step is. For example if only 5% continues to the website after reading the email, the email is not interesting enough for the audience. But if for another step 95% continues, that indicates that almost anyone will do the next step, which could either be because it is an easy or trivial step, or because the user is invested and wants to continue. The users that do not continue, have stopped following the steps in the funnel, these are shown in the ”stopped” statistic between steps.



Email delivery

Time period

From 2017-11-27 23:00

To 2017-12-04 22:59

Actors:

Venue

User

Email template filter

Template

Update

989 emails sent

100% of the funnel starters

Emails per template

Emails per language

Emails per venue

→ 44 stopped (4%)

→ 3 emails bounced

→ 13 emails dropped

↓ 945 continued (96%)

0 rejoined →

945 emails delivered

95.55% of the funnel starters

→ 389 stopped (41%)

↓ 556 continued (59%)

22 rejoined →

578 emails read

58.44% of the funnel starters

Graph of the time between sent and read

→ 373 stopped (65%)

↓ 205 continued (35%)

41 rejoined →

246 clicked on a link

24.87% of the funnel starters

Tracked since 07-11-2017

Figure 7: Results and display of an email delivery funnel

Stop groups

In some cases there is a clear reason for why a user stopped, for example when sending an email there might be feedback about why the email has not been delivered. This could for example be because the email address does not exist, the inbox is full or because it is rejected by a spam filter. If this kind of detailed information is available it can be displayed as a stopping group. Stopping groups are displayed below the ”stopped” statistic, to split the people that stopped into multiple groups. Figure 7 shows these stopping groups between ”emails sent” and ”emails delivered”. This kind of information helps to find out why a user stopped, in this case it is clear that some email never reached the users inbox, and therefore the user did not read it or visit the website.

Filters

At the top of funnel there are some form fields to limit the funnel results. The first filter is the time period, in this case filtering emails by the time they are sent. Other filters are also possible, in this case filtering for a Venue and a User are possible, limiting the emails to a hotel or user. Another filter shown in the example is the email template, so that the funnel can be shown for emails of a certain type (newsletter, password reset, invite, etcetera). These filters can be added easily to filter data of the first step of the funnel. Because following steps of the funnel will only show results for users that are included in the first step this will effectively filter results for the complete funnel.

Rejoined
Since gathering analytics is not always possible or reliable (could also call it best-effort collection) some users could skip steps in the funnel. This happens in this funnel example because the "emails read" step cannot 100% reliably detect when an email is read. Reading an email in this case is detected by including an image in the email, which will be downloaded by the users when opening the email. There are however email clients that do not download images at all or require clicking a button, so some users that read the email are not actually marked. These users could still visit the website though, which is the next step of the funnel. Because these users did not come from the "emails read" step, they are not included in the "continued" statistic, but instead are shown in the "rejoined" statistic at the left of the "clicked on a link step". So the "rejoined" statistic means that those users ended up at a step, without doing the step before it.

Step notes Each step in the funnel display page can show additional notes, displayed with a yellow background. These notes provide context information to inform about the step. For example the "Emails per template" note shows how many emails of each type have been sent. In this case this helps to use these types as filter for the funnel. At the "emails read" step there is a note that contains a graph of the time it took the people in the funnel to read an email after it has been sent (grouped by hour). This information could for example be used by a company to decide the best time to send a reminder to users about the message in the email.

5.5.2 Funnel structure

Figure 8 shows the class structure used for building the funnel and their relations. This structure will be used for both implementations, which will be discussed in following sections

First all components shown in Figure 8 need to be explained. Each class in the figure is a class in the PHP code that is used to create funnels. The main object is `Funnel`, which represents the process that is getting tracked. The `Funnel` for example has a title, description and filters to describe what the funnel is about and what it should track. A `Funnel` has one `Track`, which is what contains all the steps of the process that the funnel tracks. A `Track` has a name, and a list of steps. Steps can be of different concrete types, but share a common interface `iStep`. This common `iStep` has a name and can tell which identifiers started and finished this step.

Normally steps are of the type `SingleStep`, which represents a single action, like reading an email as in the newsletter example. For a single step the `getStartIDs()` and `getEndIDs()` have the same return value, because an email

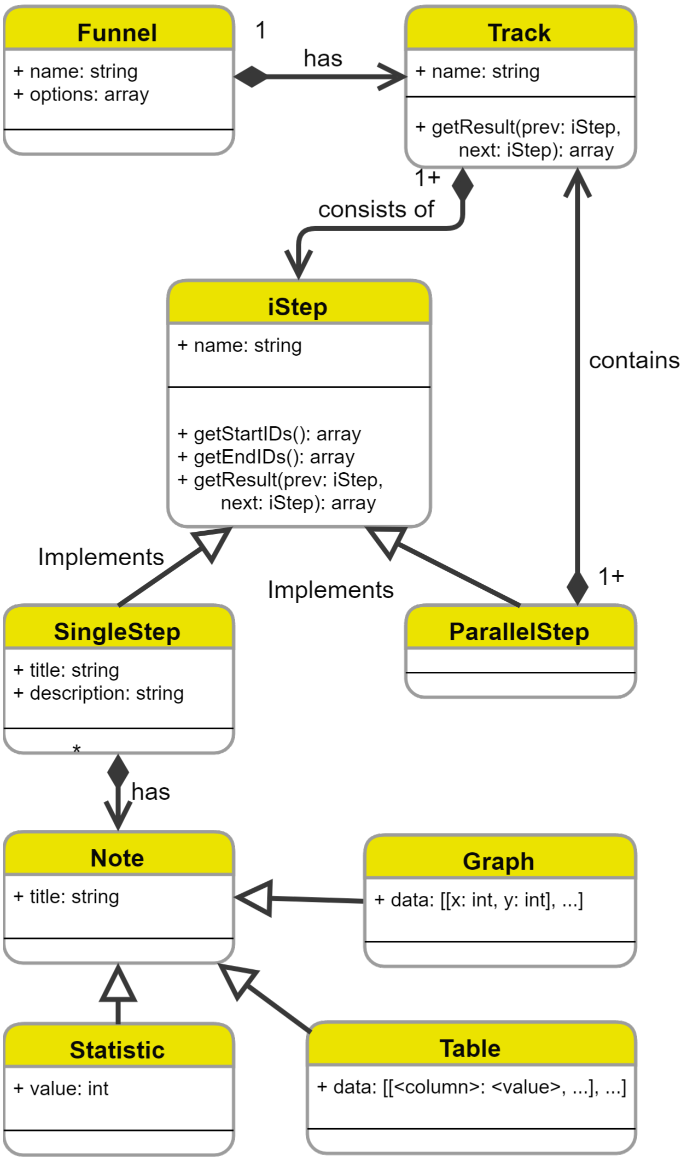


Figure 8: Funnel implementation structure

cannot be half-read. SingleStep has a title, description, and zero or more Notes. The title and description are to display along with the result, the notes are for providing context information. The content of the note is determined by the result of a SQL query placed in the funnel PHP file. Notes can be of different types, a simple Statistic that shows a certain number, a Table that shows a list of results in columns and rows, or a Graph which can for example show the distribution of a certain value or the change over time. Other note types can be added easily with this abstraction, to show the context information suitable for the company.

Next to the SimpleStep there is a ParallelStep, this step represents a split and merge, which is used when a user can take two or more possible paths. For example a website might have two possible ways to sign up, one where an email is filled in, email is read, and finally a link is clicked in the email, and another where a 'Facebook login' button is clicked, and clicking the confirm button to let the website log in. This means that for tracking the funnel it needs to split into two paths, with a certain number of steps per path, and merge afterwards. That is what the `ParallelStep` is for, it allows to split into multiple tracks, each with their own `iStep`s (which can also be parallel again), and merge back into the main funnel. This allows to define more complex funnels, improving the flexibility of the funnel system.

The `getResult()` methods in `Track` and `iStep` are to recursively build the result of the funnel, used to display a web page. It has two arguments, an argument for the previous `iStep` and one for the next `iStep`. These can be used to calculate results between steps, like how many users continued to the next step. How results are calculated and the result is displayed will be discussed in Section 5.5.

5.5.3 PostgreSQL jsonb syntax

The following steps will show how the implementation of the funnel works, which uses SQL queries in PostgreSQL. The SQL syntax is standard, and is assumed to be known by you, if not, it is easy to find an introduction online. The `jsonb` syntax is specific to PostgreSQL, so that is briefly explained below.

The events table has a `data` column that is of the `jsonb` type. This column stores a JSON document in a structured way, to make data storage flexible. To get specific information from the data column there is special syntax. To get a value of a key at the top level `data->>'<name>'` can be used, which retrieves the value of the key using the specified `<name>` as text. If instead of the double arrow (`->>`) the single arrow (`->`) is used, the type of the returned value will be as it is in JSON. So that could return a number, array, or jsonb for nested structures.

To get a value of a nested key the `data#>>'{<firstLevel>,<secondLevel>}'` is used. So the minus sign is changed to a pound sign (hashtag), and the name of the column is replaced by a list of strings. This list of strings indicates the levels it needs to follow to get the value. As with the top-level syntax, a single arrow returns as the type as encoded in JSON and a double arrow returns a text value.

More details of the jsonb syntax can be found in the PostgreSQL documentation, which can be found on their website[5].

5.5.4 Step-by-step funnel implementation

This first implementation is a straightforward version, doing all calculations in PHP while the database provides the identifiers of the users that made certain steps. To describe the implementation details of the funnel the results of the example presented in Section 4.7.1 will be calculated.

Data structure In each step of the funnel a query will be used to get a list of identifiers of the emails that made it to that particular step. This means that

[5] https://www.postgresql.org/docs/current/static/functions-json.html

the query returns one row for each user that made a particular step, which is saved to a variable in PHP. To support multiple identifiers the name given to column as result of the query, will be the name of the identifier. The name `base` is special, this is where all other identifiers should map to, and is the identifier that is known in the first step for all users. In this example the id of the email is the `base`, and no other identifiers are used. If multiple columns are returned in a step, the PHP code will map all identifiers given in a row to the same base. So if a column called `base` and a column called `userID` are in the result, all `userID` values are mapped to the `base` value. After this mapping has been done, all following steps can simple return a single column with `userID` to indicate which users made it, and the system will figure out which `base` id this belongs to. Secondary identifiers (any identifier that is not `base`) do not need to exist for each user, for example the `userID` is only available if the user has an account, so it might be missing for some.

Step 1: Which emails have been sent?

Listing 16 shows the query that is used to get the result of step 1. It queries the events table, with a couple of conditions. The first condition is that `type` is set to `email`, which means the event is about email. To narrow it to emails that have been sent a condition with `email,type` that is set to `sent` is added. Finally the `at` field of the event is used to only return emails from a certain time period. In this case a column with `email,id` is returned as result, with the name `base` to use those as identifier for this funnel.

Listing 16: Query to get the sent email events

```
select
    data#>>’{email,id}’ as base
from events
where
    data->>’type’ = ’email’
    and data#>>’{email,type}’ = ’sent’
    and data->>’at’ > ’$funnel->start’ and data->>’at’ < ’$funnel->end’
```

Step 2: Which emails have been read?

In this step the funnel will show the emails that got read, which can be found using the query in Listing 17. This query again uses the events table and checks for email events. Now the sub-type to check for is `read`.

This time the result is not filtered by time, since if the email is read any time after it has been sent, it has been read. This could be within the time period of the first step, or after it. Companies using funnels need to be aware about when the funnel is viewed, if a funnel of yesterday is viewed, a lot of emails might not be read yet. If however a funnel is viewed for a day that is twee weeks ago, a lot more emails might be read. It is important to be aware of this fact, so that results are not looked at prematurely. A company could for example decide that if an email is not read after 2 weeks, it does not have much chance anymore to get read at all, therefore viewing funnels from time periods at least 2 weeks ago is correct. This is a decision the company has to make, depending on the data in the funnel.

The email read events still need to be filtered though, since they should only count if an email that is also in the first step, and not for other emails. This is why the last filter in the query is added, which tests if the id of the email is in the set of identifiers of the first step. The set of identifiers can be retrieved from the funnel class, with the `inIds(<name>)` method. In this case `base` is used as name, since that is what the first step saved the email ids with. The result of this method will for example be `('2017-11-123456', '2017-11-789', '2017-11-08747')`. Using the `in` operator of PostgreSQL the query checks that the id of the email the read event is for is in this set.

The query for this second step again returns the `email,id` rows, which the application stores. For calculating the results for the continued, stopped and rejoined statistics the application compares the ids of this step to the previous one.

Listing 17: Query to get the read email events

```
select
    data#>>'{email,id}' as base
from events
where
    data->>'type' = 'email'
    and data#>>'{email,type}' = 'read'
    and data#>>'{email,id}' in $funnel->inIds('base')
```

Step 3: Which emails lead to website visits

The last step of the funnel is to check how many users click a link in the email to visit the website. Listing 18 shows the query. It queries for `website` events, with sub-type `visit`. The links in the email contain a parameter with the email id, so the link to the website would for example be `http://company.com/product?email_id=abcdefghijklmnopq`. The server of the website creates an event for visits, putting the url parameters in the `params` part of the event. So to filter these website visits for the emails of the funnel the `website,params,email_id` is used. This `email_id` is, as in previous steps, returned as `base`.

Next to the usual column called `base` there is an additional column `website_session`. This column contains the identifier of the website session, this session identifier is consistent through a series of page visits of this user. The system maps this website session to the base identifier that is present in the same row of the result set. This example funnel does not have any more steps after the first website visit, but it would make sense to have additional steps with for example creating an account on the website or ordering a product.

Listing 18: Query to get website visit events

```
select
    data#>>'{website,params,email_id}' as base,
    data#>>'{website,session}' as website_session
from events
where
    data->>'type' = 'website'
    and data#>>'{website,type}' = 'visit'
    and data#>>'{website,params,email_id}' in $funnel->inIds('base')
```

Pros and cons

1. Pro: Relatively simple queries, simply return the identifiers that made it as rows.
2. Pro: Step-by-step queries, making the funnel modular (easy to reuse steps for other funnels).
3. Con: Identifiers of all users in the first step are inserted into the query (sending it from PHP to PostgreSQL and parsing it will take longer), this causes poor scalability.

PHP funnel builder

To build the funnel as described in the above steps in PHP the classes from Figure 8 have been created. For constructing the funnel a builder pattern is used. The funnel as explained above, with additionally some parallel steps, can be found in Listing 19.

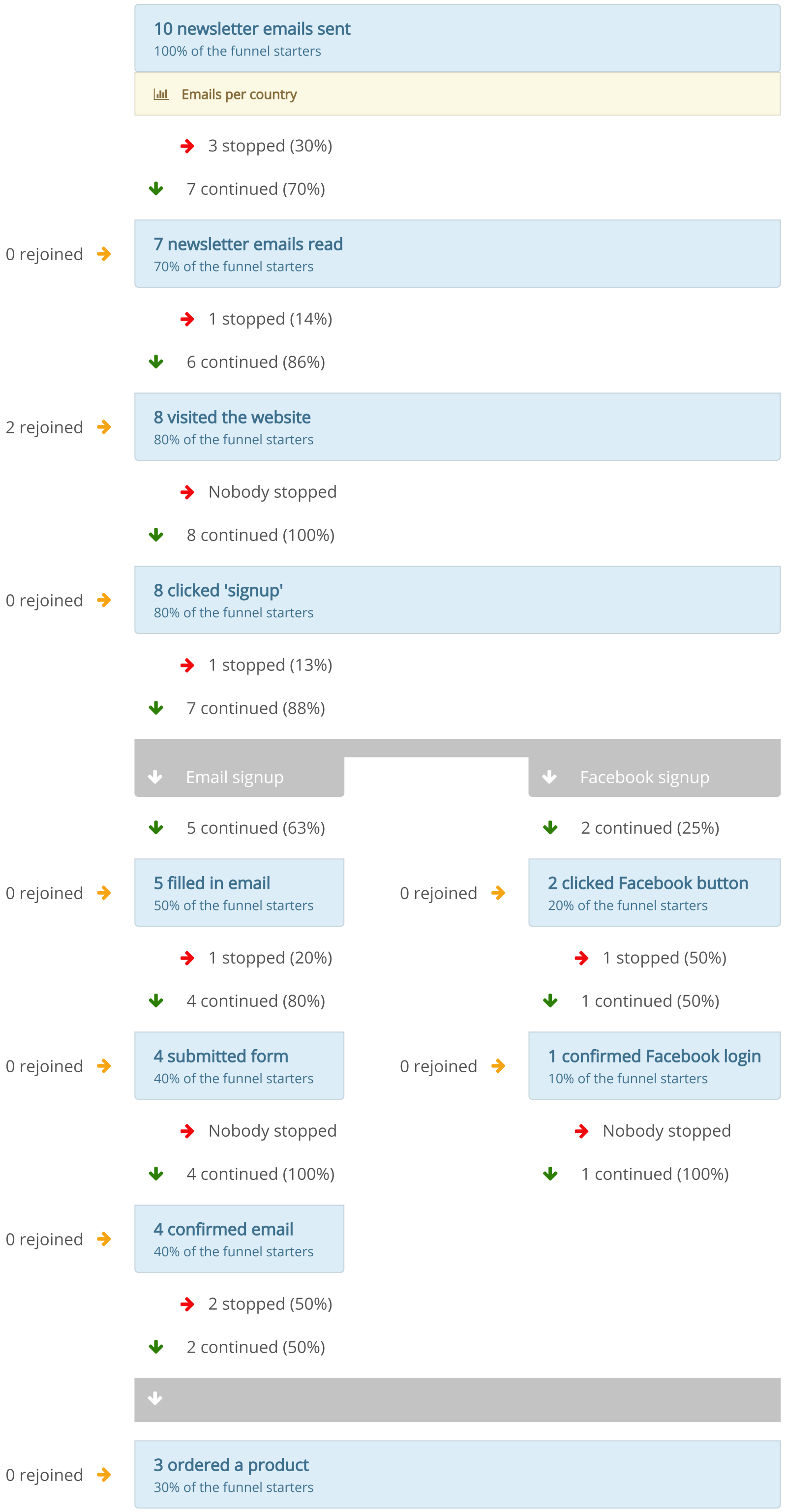
Listing 19: Usage of the builder pattern to define a funnel

```
$funnel = new Funnel("Newsletter effectiveness");
$track = $funnel->getTrack();

$track->step("Newsletter emails sent", "<query>")
    ->note("Emails per country")
        ->table("<query>");
$track->step("Newsletter emails read", "<query>");
$track->step("Visited the website", "<query>");
$track->step("Clicked 'signup'", "<query>");

$signupTracks = $track->parallel();
$emailSignup = $signupTracks->track("Email signup");
$emailSignup->step("Filled in their email", "<query>");
$emailSignup->step("Submitted the signup form", "<query>");
$emailSignup->step("Confirmed their email", "<query>");

$facebookSignup = $signupTracks->track("Facebook signup");
$facebookSignup->step("Clicked Facebook button", "<query>");
$facebookSignup->step("Confirmed Facebook login, "<query>");

$track->step("Ordered product", "<query>");
```

First a Funnel object is constructed with a title, after which the main track of the funnel is retrieved with `$funnel->getTrack()` and stored in a variable. This track will be used to add all steps to. Next a step is added by calling `$track->note()`, providing a title and a SQL query to get the identifiers for that step. For this first step a note is added with information in a table. As designed, more notes can be added by calling `note()` again.

After the first three steps the funnel is splitting into two different tracks. First `$track->parallel()` is called to introduce a parallel step. On this parallel step an email signup track is created with `$signupTracks->track()`. On this email signup track three steps are added. Another track for a Facebook signup is added, with two steps. After these parallel signup tracks an order product step is added, which can be done after either of the signup options.

Figure 9 shows how this funnel shows when executed. The split is clearly indicated with a grey horizontal bar, with names above each track. After the parallel steps another grey bar is shown to indicate the tracks have merged again.



10 newsletter emails sent
100% of the funnel starters

Emails per country

➔ 3 stopped (30%)
↓ 7 continued (70%)

0 rejoined ➔ 7 newsletter emails read
70% of the funnel starters

➔ 1 stopped (14%)
↓ 6 continued (86%)

2 rejoined ➔ 8 visited the website
80% of the funnel starters

➔ Nobody stopped
↓ 8 continued (100%)

0 rejoined ➔ 8 clicked 'signup'
80% of the funnel starters

➔ 1 stopped (13%)
↓ 7 continued (88%)

↓ Email signup

↓ 5 continued (63%)

0 rejoined ➔ 5 filled in email
50% of the funnel starters

➔ 1 stopped (20%)
↓ 4 continued (80%)

0 rejoined ➔ 4 submitted form
40% of the funnel starters

➔ Nobody stopped
↓ 4 continued (100%)

0 rejoined ➔ 4 confirmed email
40% of the funnel starters

➔ 2 stopped (50%)
↓ 2 continued (50%)

↓ Facebook signup

↓ 2 continued (25%)

0 rejoined ➔ 2 clicked Facebook button
20% of the funnel starters

➔ 1 stopped (50%)
↓ 1 continued (50%)

0 rejoined ➔ 1 confirmed Facebook login
10% of the funnel starters

➔ Nobody stopped
↓ 1 continued (100%)

↓

0 rejoined ➔ 3 ordered a product
30% of the funnel starters

Figure 9: Parallel funnel display

5.5.5 In-database funnel implementation

The second implementation focuses on reducing the traffic between the database and application by keeping intermediary results in the database. This way queries don't need to return huge lists of identifiers back to the application, but only the final numbers required to display the funnel. This implementation has been made to combat the weak points of the first implementation, which is the scalability.

Data structure

Instead of storing lists of identifiers for each step in PHP as done in the first implementation, this implementation keeps the identifiers in PostgreSQL. The identifier mappings are stored in a temporary table. Listing 20 shows the query to create this temporary table. The temporary table will have a column for the `base` identifier, and each secondary identifier. For the base identifier a constraint is added to make sure it is never `NULL`, so that mistakes while creating the funnel are noticed. Secondary identifiers can be null, because not every user in the funnel might reach the step where a second identifier is introduced.

The create table query has a special clause at the end `ON COMMIT DROP` which deletes the temporary table when the transaction end. This requires that at the start of the funnel the `BEGIN TRANSACTION` command is used, and at the end the `COMMMIT` command. Using this structure it is ensured that the temporary table is deleted when the funnel results are calculated.

Listing 20: Query for creating the mapping table

```sql
CREATE TEMP TABLE map (
    base TEXT NOT NULL,
    user TEXT DEFAULT NULL
) ON COMMIT DROP;
```

Two more temporary tables are used, one to store the identifiers of the current step, and one for the identifiers of the previous step. Listing 21 shows the query used for creating both of these tables (where the table of the previous step used the name `previous`, and the current one uses `current`). Usage of these tables will be made clear while going through the funnel steps.

Listing 21: Query for creating the table for the current step

```sql
CREATE TEMP TABLE current (
    base TEXT NOT NULL
) ON COMMIT DROP;
```

Step 1: Which emails have been sent?

Listing 22 shows the query used to get the first results. The select query is exactly the same as in the first implementation, but instead of returning the results to PHP it inserts them in the temporary table.

Listing 22: Query to get the sent email events and put them into the temporary table

```
insert into current (
    select
        data#>>'{email,id}' as base
    from events
    where
        data->>'type' = 'email'
        and data#>>'{email,type}' = 'sent'
        and data->>'at' > '$funnel->start' and data->>'at' <
            '$funnel->end'
);
```

Because this is the first step, the map table also needs to be updated, this is shown in Listing 23. Now the map contains all the base identifiers that will be followed during this funnel.

Listing 23: Copy the result of the first step into the map table

```
insert into map
select *
from current;
```

The resulting value for the current step can be retrieved using a simple query that counts the number of rows in the `current` table, like shown in Listing 24.

Listing 24: Get the result count for a step

```
select count(*)
from current;
```

Since this is the first step, it does not need to be compared to a previous step to get the continued/stopped/rejoined numbers. Before moving on to the next step the result of this step needs to move to the `previous` table, for which two queries are listed in Listing 25

Listing 25: Move results of the current step to the previous table

```
insert into previous
select * from current;

truncate current;
```

Step 2: Which emails have been read?

The query to get the results of the second step has some changes compared to the one of the first implementation. Listing 26 shows the resulting query. It selects the email read events from the table, and joins them with the map table to limit the results to only the identifiers that are present in that table. The resulting identifiers are again inserted into the `current` table.

Listing 26: Query to get the results for the emails read step

```
insert into current (
    select
        events.data#>>'{email,id}' as base
    from events
    inner join map on
        events.data#>>'{email,id}' = map.base
    where
        events.data->>'type' = 'email'
        and events.data#>>'{email,type}' = 'read'
);
```

Getting the count for the second step is the same as the first step, simply counting the number of rows in the `current` table. For getting the number of people that continued from the first to the second step the query in Listing 27 is used. This selects the rows from the current step, that directly came from the previous step (to ensure rejoined people are excluded). With this continued number the stopped number of the previous step is simply subtracting it from the result of the previous step.

Listing 27: Get the number of people that continued from the previous step to this one

```
select count(*)
from current
left join previous on previous.base = current.base;
```

One result is still missing, the number of people that did not do the previous step, but did do the current one, which is the rejoined statistic. Listing 28 shows the query used to calculate it. It counts the number of identifiers in `current` that do not appear in `previous`.

Listing 28: Get the number of people that rejoind

```
select count(*)
from current
left join previous on previous.base = current.base
where previous.base = null;
```

Again the data for this step is moved to the previous one, and the `current` table is cleared, as shown in Listing 25.

Step 3: Which emails lead to website visits?

This step looks like the previous one, the only difference is the query that gets the results, which is shown below in Listing 29.

Listing 29: Get the results for website visits

```
insert into current (
    select
        events.data#>>'{website,params,email_id}' as base
    from events
    inner join map on
        events.data#>>'{website,params,email_id}' = map.base
    where
        events.data->>'type' = 'website'
        and events.data#>>'{website,type}' = 'visit'
);
```

Pros and Cons

This system solves some of the problems in the first implementation, but also has some negative sides of its own. Below the positive and negative points of this implementation are listed.

1. Con: The query used to select the identifiers for each step needs to be wrapped with an insert query, adding complexity (this can however automatically be added by the application).
2. Con: Temporary tables need to be created and managed, adding complexity.
3. Pro: Small query size, it is not necessary to insert all identifiers into the query, reducing query parsing time.
4. Pro: Improved performance, since indexes could be added to the temporary table and speed up usage.

5.5.6 Real-time funnels

This project does not focus on showing funnels real-time, but for some companies this might be useful. A way to create a real-time system would be to match each incoming analytics event for matching funnel steps. If a matching step is found the tracked user in that funnel could be updated to store that he has made it to that particular step. This could be combined with sending an update to the people that are watching the funnel result, so that the interface can show updated numbers. This could be a future work project to investigate.

6 System validation

For validation of the system a combination of techniques is used. The positive and negative aspects of the system are discussed, with possible alternatives. A couple of benchmarks are used to test the claims of the system. Also validation while implementing it for Staying is used, to indicate how it works for a company.

6.1 Fitness for purpose of the database

This section will validate the database setup created for this research using PostgreSQL and the CitusData extension. First the technical possibilities and limitations of the combination of PostgreSQL and CitusData are evaluated, after which the usage of the database is analyzed.

6.1.1 PostgreSQL limitations

PostgreSQL works quite well for this project. It offers all regular SQL features useful for relational data storage, with some advanced additions. One such addition is the `jsonb` column type, which has become critical to this project. It offer the flexibility that is required, while still allowing good performance because particular fields inside a JSON column can still be used with indexes. Already on its own PostgreSQL can handle quite big data sizes, up to 32TB per table, as found on their about page[6]. Table 1 shows the other limits PostgreSQL defines.

Table 1: PostgreSQL theoretical limits

Limit	Value
Maximum Database Size	Unlimited
Maximum Table Size	32 TB
Maximum Row Size	1.6 TB
Maximum Field Size	1 GB
Maximum Rows per Table	Unlimited
Maximum Columns per Table	250-1600 depending on column types
Maximum Indexes per Table	Unlimited

Overall PostgreSQL has proven to be a good choice, that worked out to do the job. Because of the extensibility of PostgreSQL it could be scaled to multiple servers with CitusData, which is discussed below.

6.1.2 CitusData limitations

Setting up CitusData was quite easy and using it in a basic way was also straightforward. CitusData however does have some limitations, as documented on their documentation pages. It does not support all features of SQL, like subqueries in the where condition. Often this can easily be worked around, but this does mean migration to a database that uses CitusData from a regular SQL database gets harder. The exact limitations can be found in their documentation[7]. One limitation to consider is that certain commands

[6] `https://www.postgresql.org/about/`

[7] `https://docs.citusdata.com/en/latest/`

When using CitusData there is also a dependency on this extension, which brings certain risks. Luckily CitusData is open-source and can be found on Github[8], which at least provides a way to continue using it when Citus would disappear. It also has a substantial user base, considering the activity and stars on the Github repository, which helps to ensure continuity.

The performance of Citus has already been proven by benchmarking of their own as shown in their documentation[9], which has been verified. Similar number have been reached when testing this myself, with slight difference since the benchmark of the documentation uses Citus Cloud and my own was on a CloudFormation cluster on Amazon (with similar, but possibly different EC2 instances). Because my number were close, I believe that the benchmarks done by Citus themselves are accurate. This means that scaling the Citus cluster to a level that is suitable for the desired throughput. After scaling redistribution of the data among the worker notes is required, this operation can be done with a single command. This however does require a subscription to Citus Enterprise, which provides 24/7 support and other enterprise features.

6.1.3 Retrieving events from the system

For Staying the events about the usage of its products by guests and property owners is stored in the analytics database. Staying is interested in interested in lists of events on a number of levels:

1. Global: all events of all products
2. Venue: events of a venue
3. User: events of a guest or property owner
4. Custom: specific filters for checking other cases

To show event lists for these levels a simple interface with a couple of filters has been created, as shown in Figure 10. With these filters it is easy to filter by venue, user, or other actors. Next to that filters on the top-level types and certain sub types have also been added. As escape hatch, custom filters can be used, where comma-separated paths can be filled in and matched to a certain value.

The structure that has been defined for events has helped to effectively display these events. Events are simply displayed one per row, where the event is displayed with some simple styling. For each event type a Smarty template can be defined to render the event, which can show the details in the event that are the most useful. If a template for a top-level type does not exist, but the event has a sub-type, it will try to render the template defined for that sub-type. Templates are simply placed into a directory structure that follows the types and sub-types of the events.

This system serves Staying well, because it is easy to investigate usage of features, to see if they are used effectively.

[8] https://github.com/citusdata/citus

[9] https://docs.citusdata.com/en/v7.1/performance/scaling_data_ingestion.html

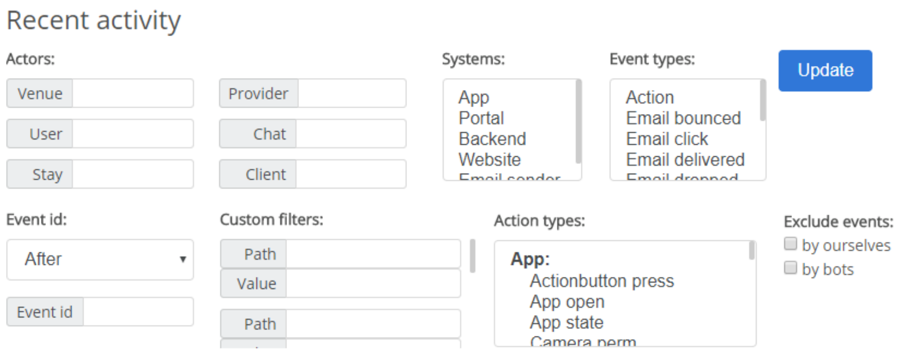


Figure 10: Options for filtering events

6.1.4 Using the system to answer questions

To show that the questions that have been defined in Section 3.1 (analytics questions requirements) work in the system, they are shown below. The seven domain-specific questions asked by Staying will be answered below using SQL statements.

Question 1: What was the last activity from a venue on the portal? Query found in Listing 30. It simply selects the highest timestamp from the events table for selected venue. This assumes there is an ordered index on the `epoch` key in the `data` column, which is also useful for a lot of other queries. Since this query uses a single venue, Citus will send this query to the worker that holds the data of this venue directly, and it will execute it.

Listing 30: Question 1

```sql
select max((data->>'epoch')::float)
from events
where venue = 123;
```

The result has been checked by manually causing new events to come in for a certain venue, to see if the result of the query reflects this new time, while other venues are unaffected.

Question 2: When was the last activity of a user on the app? This question can be answered with almost the same query as question 1, see Listing 31. For important columns like the user it is also good practice to add an index. This would let this query do an index scan on the user index, which limits the results quickly and makes this query efficient enough. Since data one a user can be on multiple server this query will run on all workers and the results will get combined.

Listing 31: Question 2

```sql
select max((data->>'epoch')::float)
from events
where data->>'user' = '123';
```

The result has been checked by causing events as a certain user and checking if the result has updated as well, without affecting results of other users.

Question 3: How many times each content item has been viewed? The query for this question can be found in Listing 32. This one is more complicated, it starts by selecting from the `content` table, which has content items for each venue. The content items are joined with their venue, which is required later. Then the events table is joined with the content+venue rows. Events are filtered to the action that represents viewing a content item (which of course might be different to monitor other activity). At the end the results are grouped by the content it, so that the select can count the number of events for each content item. The results rows are ordered by the number of views. The select adds the title of the content item, which can be found in its data under the `title` key, with the language appended. To append a valid language of a venue the first language is used which is found in `venues.langs[0]`. Since the venue is specified in the query, it will be pushed to a single worker and execute nicely.

Listing 32: Question 3

```sql
select
    content.data->>('title_' || (venues.langs->>0)::text) as title,
    count(events.id) as visits,
    content.id as id
from content
inner join venues on
    content.venue_id = venues.id
    and venues.id = 4
left join events on
    content.venue_id = events.venue
    and events.venue = 4
where
    events.data->>'content' = content.id::text
    and events.data->>'type' = 'action'
    and events.data#>>'{action,type}' = 'navigate'
    and events.data#>>'{action,navigate,to,name}' = 'ContentX'
group by content.id, content.data->>'title_en', venues.langs
order by count(events.id) desc;
```

The result has been checked by creating a new venue and doing visits to content items while constantly checking the result of the query.

Question 4: How long has each content item been viewed? This query is harder to create, since it needs to calculate the time between opening the content item and closing it. The result is shown in Listing 33. The way to get to a content item is by navigating to it, so first the navigate event is selected that goes to a content item. To get the closing event an inner join on the events table is used, selecting events that indicate leaving the content item. For example this could be navigating to something else, or closing the app. It could be hard to determine all leave event types here, so this could lead to inconsistencies. Finally the results are grouped by the start event, and the first end event is selected to make sure the closest one is used.

It would be wise to limit the time difference to a certain number of seconds, because it is not realistic that someone is actively viewing a content item for longer than an hour, so higher than that is likely a measurement error.

Listing 33: Question 4

```
select
    start.data->>'epoch' as start_time,
    min(e.data->>'epoch') as end_time,
    min((e.data->>'epoch')::float) - (start.data->>'epoch')::float as
        diff
from events as start
inner join events as e on
    e.venue = start.venue
    and e.data->>'type' = 'action'
    and e.data#>>'{action,type}' in ('navigate', 'close_app')
    and e.data->>'user' = start.data->>'user'
    and (e.data->>'epoch')::float > (start.data->>'epoch')::float
where
    start.venue = 1
    and start.data->>'type' = 'action'
    and start.data#>>'{action,type}' = 'navigate'
    and start.data#>>'{action,navigate,to,name}' = 'ContentX'
group by start.id
order by diff desc;
```

The result has been checked by starting with a new venue and doing visits of a certain time of certain items.

Question 5: When did action X happen last in the app/portal for user/venue Y?
The resulting query for this question is found in 34. This query is straight forwarded, simply selecting the maximum time from the events table with certain conditions. This scenario would often be useful for a list of venues/users, in which case showing metadata of these users/venues would be relevant. To do that a join could be done on the users or venues table, so that this data can be used. This is a typical combination of usage and profile data.

Listing 34: Question 5

```
select
    max((data->>'epoch')::float) as last
from events
where
    data->>'type' = 'website'
    and data#>>'{website,type}' = 'visit'
    and venue = 123
    and data->>'user' = '456'
    and data#>>'{system,type}' = 'app'
```

The result has been checked by causing events for certain venue/user combinations, checking if the time updates while not affecting other event types/user/venues.

Question 6: How often did action X happen in the app/portal for user/venue Y in time period Z?
This question can be answered much like question 5, with a few small differences, see Listing 35. Instead of using a `max(...)` as result, `count(*)` is used. Filtering for time period are two simple where conditions, already used earlier: `(data->>'epoch')::float < 456` and `(data->>'epoch')::float > 123`.

When answering this question for a list of different event types, the result could be grouped by event type to generate a list of types with how often they happened. This could give a good idea about the most common actions a venue performs.

Listing 35: Question 6

```sql
select
    count(*) as times
from events
where
    data->>'type' = 'website'
    and data#>>'{website,type}' = 'visit'
    and venue = 123
    and data->>'user' = '456'
    and data#>>'{system,type}' = 'app'
    and (data->>'epoch')::float < 456 and (data->>'epoch')::float > 123
```

The result has been checked by setting up certain events at consistent intervals, and testing if the result of the query matches the expectation.

Question 7: Which device types are used per venue type?
This question is answered with the query from Listing 36. The query detects the device type (ios, android, desktop in this case) from the agent of the client that produced the event. Next the events are grouped by venue type and platform, and ordered by venue type, platform and usage. This generates a list of how many events each venue type has on each device type. Of course these results could be visualized using a pie chart to make them easy to read.

Currently the events are not filtered on their type, but normally it might make sense to for example filter on app events.

Listing 36: Question 7

```
select
    case
        when data#>>’{system,user_agent}’ like ’%iPhone%’ then ’ios’
        when data#>>’{system,user_agent}’ like ’%Android%’ then ’android’
        else ’desktop’
    end as platform,
    venues.type as venue_type,
    count(*) as usage
from events
inner join venues on events.venue = venues.id
group by platform, venue_type
order by venue_type, platform, usage
```

To check the result a sample set of real data has been used, manually counted and compared to the query result, which matched.

Conclusion
Answering the questions above has shown that it is possible, and in most cases easy to answer these types of questions using the system. Since the usage and profile data is in the same system joining them is easy and works quickly.

6.2 Defining funnels using the system

A system has been made to build funnels, using the database setup of this project. The choices made while implementing the systems of Section 5.5 have an effect on the easy of use, scalability and maintainability of the system. This section describes the trade-offs that have been made for the system to define funnels.

6.2.1 Defining the funnel: builder pattern

A positive side of the implementations is the builder pattern of the PHP application. This makes it easy to add steps, notes and other options. It helps to make the system flexible, since adding a new function option to a step or note is simply adding an extra method on them. Instead of a builder pattern a different approach could be used, like a domain specific language (DSL). This would mean defining a structure of a text file to describe a funnel and all its options. Using a DSL to define funnels would limit the people using the system to create funnels to the features of the DSL, which could quickly become a problem. Of course the DSL could be expanded with a new feature, but that can easily bloat the DSL, making it hard to understand, use and maintain. Instead using a programming language like PHP allows to do basically any processing required, while still allowing for a simple funnel definition if additional processing is not needed. Another options would be to create a visual interface to build the funnel. It would make creating simple funnels easier, but would again limit the funnel to what the interface can do. These are however trade-offs that need to be made with the target audience in mind. If the intent is to let anyone in the company create funnels, an interface might be the best choice, however if only people with a Computer Science (or related) background need to be able to create them a programming language might be more suitable.

6.2.2 Flexibility and modularity

For both implementations the steps of the funnels are independent from the previous steps. What is meant with this is that the implementation itself will do the comparison with the previous step, calculating the continued/stopped/rejoined statistics, so the step itself only needs to worry about its own result set. This means that funnels can easily be adapted, like adding or removing a step. It also means that certain steps or sequences of steps can be reused, which improves modularity. For example when sending an email to a user there is usually a need to track of "emails delivered", "emails read" and "clicked link in email" steps, which could be split into a separate file to reuse where useful. This concept has already been used for Staying, splitting the email steps including all notes into a separate file to reuse in multiple funnels. This also keeps the code in the funnel definitions DRY (don't repeat yourself), which makes maintenance easier.

Flexibility of the funnel implementation is an important aspect, since it is intended for as much companies as possible. Implementing it specifically for Staying could lock it too much into a domain-specific feature set, which harms the possibility to use it for other companies. Flexibility also comes back in the following paragraphs.

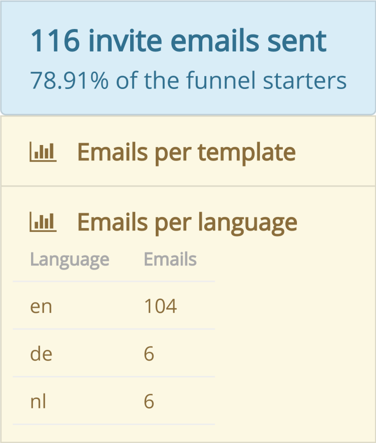
6.2.3 Context information: notes

Even though the numbers in the funnel itself say a lot about the process it tracks, context information is important to get more insight. Often certain aspects of the users entering the funnel are important to know, like the language of the email they received. This is often in conjunction with filters, to be able to show the funnel for emails of a certain language. Since these kind of aspects of the users starting the funnel could have different results, it is important that the person using the funnel can show them.

Another aspect that could be important in the example funnel is the type of device used to visit the website (laptop, tablet, smartphone, etcetera). If a lot of the users that visit the website on a smartphone stop, it could mean that the mobile version of the website does not work good enough. To provide this information notes are critical, in this case a table with the number of users per device type would help a lot.

Charts could also be used, for example for the distribution of the number of hours between sending and reading the email. This gives insight in what a usual time period is before the majority has read an email, which means that viewing the funnel results before this time period is over would show incomplete results.

Finally, other note types can easily be added and displayed, contributing to the flexibility of the funnel building application to show any information the company requires.



116 invite emails sent

78.91% of the funnel starters

Emails per template

Emails per language

Language	Emails
en	104
de	6
nl	6

Figure 11: Collapsed and expanded note

6.2.4 Funnels created for Staying

The funnel system has already been used by Staying for a few months. It is however used with a regular PosgreSQL database, without the CitusData extension, since that is not yet required for the scale Staying operates at currently. A couple of funnels have been created like venue signup, guest invites, email delivery and sms delivery to get insight into these topics. While creating funnels for these topics the need for more advanced features became apparent, with one of the first being the notes with context information. Later filters have been added to show results of certain sub groups. The interface also developed over time, for example when realizing the 'rejoined' statistic was necessary when 'email read' tracking was proven to be inaccurate. When expanding the funnels the multi-identifier tracking was required at some point, and added for it.

Currently there are 4 distinct funnels in use at Staying, each showing a different process. Below each of these funnels is shown, and how it tests the funnel system is explained.

Property marketing funnel
For Staying it all starts with inviting properties to use the product. This process consists of the following main steps:

1. Invite a property by email
2. Visit the Staying website
3. Start and go through the sign-up wizard (3 steps)
4. Visit their own guest invite page (called the trampoline)
5. Install their app, as a guest would after a booking
6. Visit the admin portal using a link of the sign-up email
7. Click through the admin portal welcome wizard (4 steps)
8. Use the product: invite guest, add content, etcetera

Figure 12 shows the complete funnel as used by Staying.

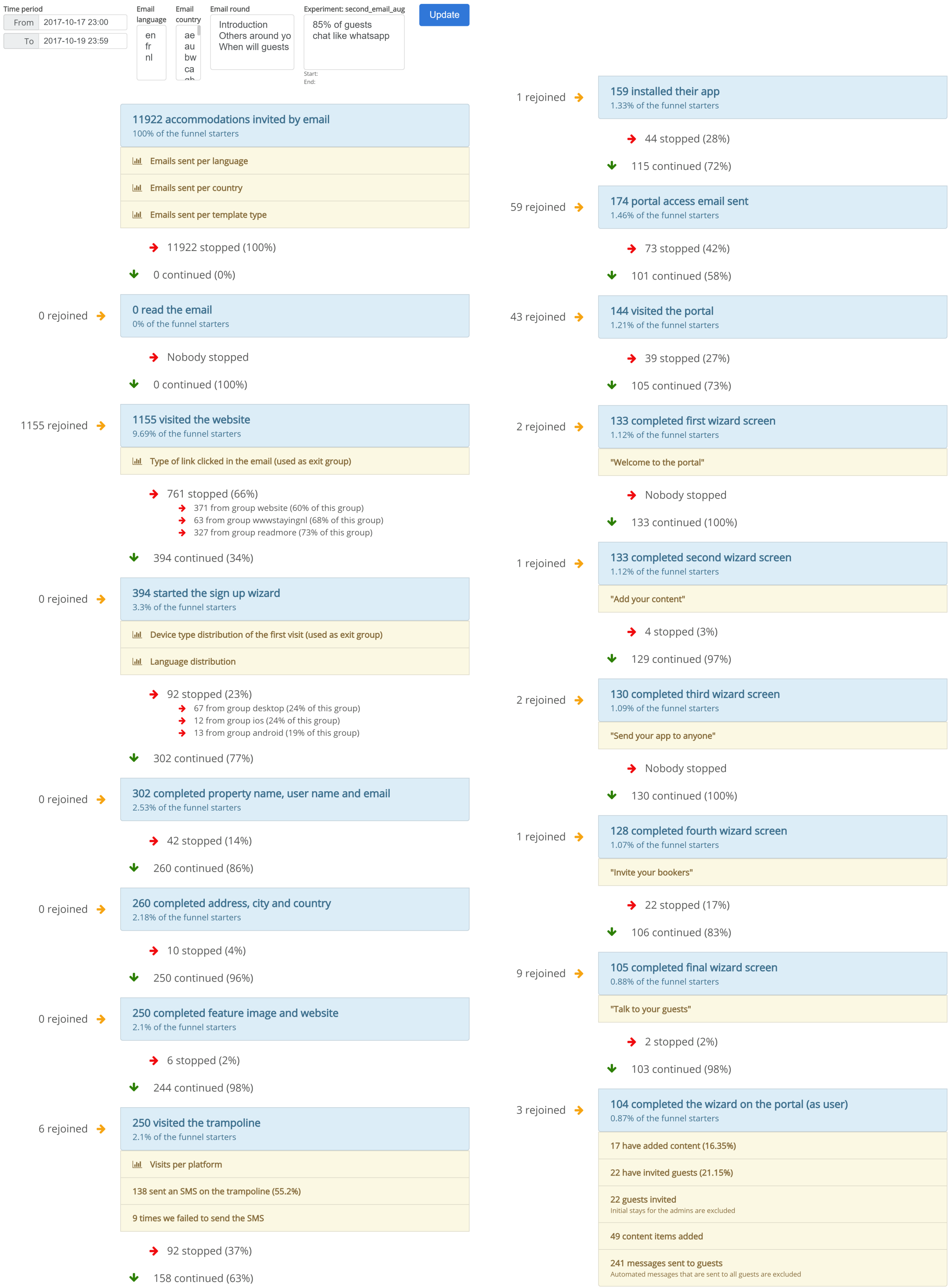
One of the first challenges for this funnel was tracking the user from the start to the end. At the start the user is identified by their email address. After that the sign-up is started, which uses a random id assigned to the current session. When the sign-up is done a user and venue number will be known, these can be used for further tracking. All of these different identifications of the same user that is going through the signup of a property for Staying need to be linked together. The task of Staying is that when switching to another identifier it needs to be logged at least once with the previous identifier, which allows for combining them. The funnel system handles tracking the different identifiers through the funnel together, effectively combining their paths.

This funnel helps Staying to track the effectiveness of the marketing emails to properties. The goal of Staying is to get active properties on the platform to get feedback on the product. To reach this goal all the steps of this funnel need to be completed by properties. Writing down the complete process as a funnel helped Staying to improve the process. Even though the funnel is still really long, a couple of steps have been scrapped or made a lot easier after Staying saw that a log of people were lost at that step.

Context information as notes in the funnel have been helpful, Staying for example saw that which link people clicked in the email did not matter for the percentage that actually signs up. Another thing Staying found out is that a surprisingly high percentage uses a mobile device to start the signup, which encouraged Staying to improve the mobile signup.

The funnel has a couple of filters. First there are options to filter by email language and country of the property. These allow Staying to compare results between target countries, since properties of certain countries will react in a different way to marketing. Staying also tried different email content, which can be filtered on. After the first marketing email Staying will send follow up emails, which can also be filtered on.

Staying has used this funnel to improve the sign up process of new properties. When the designer Agustina wrote down the complete process on post-its on a wall, it became clear that there were a lot of steps. After implementing the funnel into the system, the numbers showed the most critical steps of the process. Staying has shortened the process, eliminating as much steps as possible. Another improvement they made is automatically filling in forms, so that the new potential user only needs to click continue. After this improvement, users will continue around 98% of the time in these steps.



Time period
From 2017-10-17 23:00
To 2017-10-19 23:59

Email language
en
fr
nl

Email country
ae
au
bw
ca

Email round
Introduction
Others around yo
When will guests

Experiment: second_email_aug
85% of guests
chat like whatsapp
Start:
End:

Update

11922 accommodations invited by email
100% of the funnel starters

- Emails sent per language
- Emails sent per country
- Emails sent per template type

→ 11922 stopped (100%)

↓ 0 continued (0%)

0 rejoined → **0 read the email**
0% of the funnel starters

→ Nobody stopped

↓ 0 continued (100%)

1155 rejoined → **1155 visited the website**
9.69% of the funnel starters

- Type of link clicked in the email (used as exit group)

→ 761 stopped (66%)
→ 371 from group website (60% of this group)
→ 63 from group wwwstayingnl (68% of this group)
→ 327 from group readmore (73% of this group)

↓ 394 continued (34%)

0 rejoined → **394 started the sign up wizard**
3.3% of the funnel starters

- Device type distribution of the first visit (used as exit group)
- Language distribution

→ 92 stopped (23%)
→ 67 from group desktop (24% of this group)
→ 12 from group ios (24% of this group)
→ 13 from group android (19% of this group)

↓ 302 continued (77%)

0 rejoined → **302 completed property name, user name and email**
2.53% of the funnel starters

→ 42 stopped (14%)

↓ 260 continued (86%)

0 rejoined → **260 completed address, city and country**
2.18% of the funnel starters

→ 10 stopped (4%)

↓ 250 continued (96%)

0 rejoined → **250 completed feature image and website**
2.1% of the funnel starters

→ 6 stopped (2%)

↓ 244 continued (98%)

6 rejoined → **250 visited the trampoline**
2.1% of the funnel starters

- Visits per platform
- 138 sent an SMS on the trampoline (55.2%)
- 9 times we failed to send the SMS

→ 92 stopped (37%)

↓ 158 continued (63%)

1 rejoined → **159 installed their app**
1.33% of the funnel starters

→ 44 stopped (28%)

↓ 115 continued (72%)

59 rejoined → **174 portal access email sent**
1.46% of the funnel starters

→ 73 stopped (42%)

↓ 101 continued (58%)

43 rejoined → **144 visited the portal**
1.21% of the funnel starters

→ 39 stopped (27%)

↓ 105 continued (73%)

2 rejoined → **133 completed first wizard screen**
1.12% of the funnel starters

"Welcome to the portal"

→ Nobody stopped

↓ 133 continued (100%)

1 rejoined → **133 completed second wizard screen**
1.12% of the funnel starters

"Add your content"

→ 4 stopped (3%)

↓ 129 continued (97%)

2 rejoined → **130 completed third wizard screen**
1.09% of the funnel starters

"Send your app to anyone"

→ Nobody stopped

↓ 130 continued (100%)

1 rejoined → **128 completed fourth wizard screen**
1.07% of the funnel starters

"Invite your bookers"

→ 22 stopped (17%)

↓ 106 continued (83%)

9 rejoined → **105 completed final wizard screen**
0.88% of the funnel starters

"Talk to your guests"

→ 2 stopped (2%)

↓ 103 continued (98%)

3 rejoined → **104 completed the wizard on the portal (as user)**
0.87% of the funnel starters

- 17 have added content (16.35%)
- 22 have invited guests (21.15%)
- 22 guests invited
  Initial stays for the admins are excluded
- 49 content items added
- 241 messages sent to guests
  Automated messages that are sent to all guests are excluded

(continues in the second column)

Figure 12: Property invite funnel as used by Staying

Guest invite funnel

The second main funnel for Staying is inviting guests for installing the app of a property. This process contains the following steps:

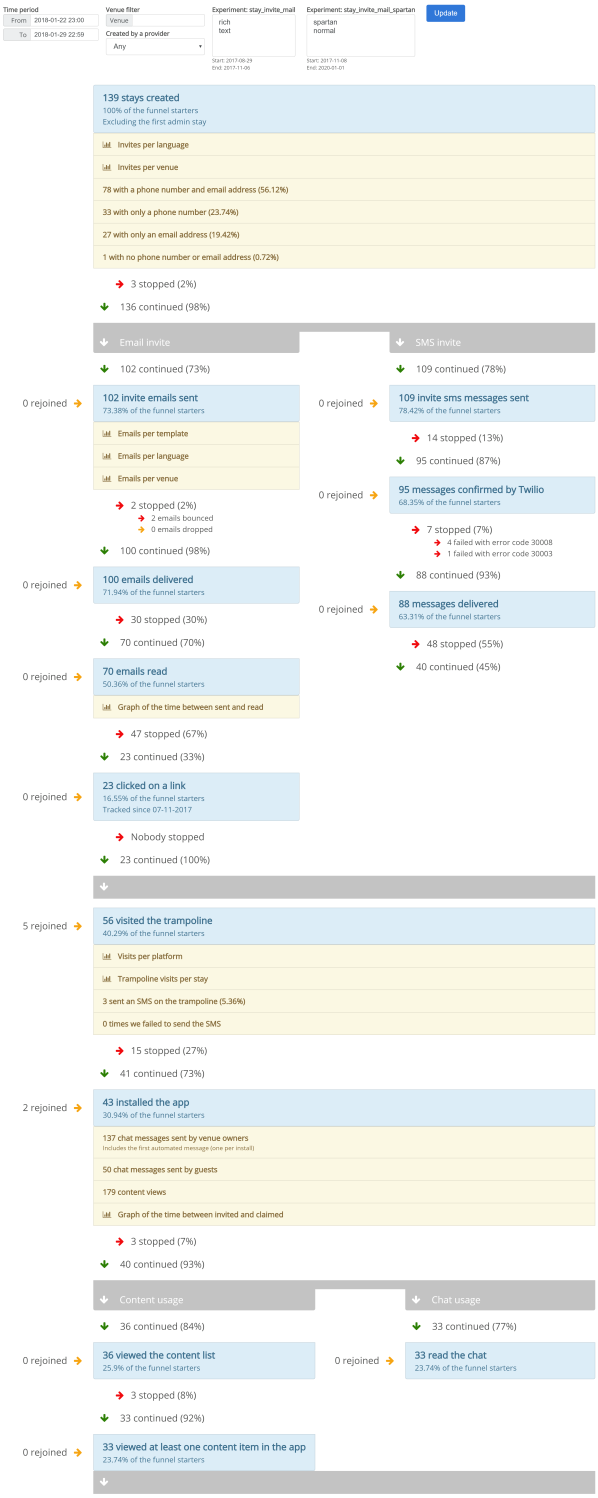
1. Booking added by a property owner
2. Invites sent out by email and/or sms (with delivery tracking)
3. Guest visits the trampoline (showcase of the app, with install links)
4. Guest installs the app
5. Guest views content, uses the chat, etcetera

Figure 13 shows the complete funnel as used by Staying.

Tracking is easier for this funnel, since right at the start a stay identifier is created, which is used through the complete funnel. It does contain a split/merge for inviting the guest, after creating a 'stay' the guest invited by email and sms (if both filled in by the property). These invites are both tracked for delivery, and are displayed next to each other.

After the invite the guest will visit the website and possibly install the app. In the app the guest can do multiple things, currently viewing content and using the chat are displayed in the funnel, which are the most important actions.

Staying used this funnel to improve the app install rates. Problems with email and SMS invite delivery were found and fixed. Also improvements were made to automatically add the stay of the user to the app, instead of letting them enter a code, to prevent them stopping at that stage (users would need to go back to the invite to find the code).



Time period
From 2018-01-22 23:00
To 2018-01-29 22:59

Venue filter
Venue
Created by a provider
Any

Experiment: stay_invite_mail
rich
text
Start: 2017-08-29
End: 2017-11-06

Experiment: stay_invite_mail_spartan
spartan
normal
Start: 2017-11-08
End: 2020-01-01

Update

139 stays created
100% of the funnel starters
Excluding the first admin stay

Invites per language

Invites per venue

78 with a phone number and email address (56.12%)

33 with only a phone number (23.74%)

27 with only an email address (19.42%)

1 with no phone number or email address (0.72%)

3 stopped (2%)

136 continued (98%)

Email invite

102 continued (73%)

0 rejoined

102 invite emails sent
73.38% of the funnel starters

Emails per template

Emails per language

Emails per venue

2 stopped (2%)
2 emails bounced
0 emails dropped

100 continued (98%)

0 rejoined

100 emails delivered
71.94% of the funnel starters

30 stopped (30%)

70 continued (70%)

0 rejoined

70 emails read
50.36% of the funnel starters

Graph of the time between sent and read

47 stopped (67%)

23 continued (33%)

0 rejoined

23 clicked on a link
16.55% of the funnel starters
Tracked since 07-11-2017

Nobody stopped

23 continued (100%)

SMS invite

109 continued (78%)

0 rejoined

109 invite sms messages sent
78.42% of the funnel starters

14 stopped (13%)

95 continued (87%)

0 rejoined

95 messages confirmed by Twilio
68.35% of the funnel starters

7 stopped (7%)
4 failed with error code 30008
1 failed with error code 30003

88 continued (93%)

0 rejoined

88 messages delivered
63.31% of the funnel starters

48 stopped (55%)

40 continued (45%)

5 rejoined

56 visited the trampoline
40.29% of the funnel starters

Visits per platform

Trampoline visits per stay

3 sent an SMS on the trampoline (5.36%)

0 times we failed to send the SMS

15 stopped (27%)

41 continued (73%)

2 rejoined

43 installed the app
30.94% of the funnel starters

137 chat messages sent by venue owners
Includes the first automated message (one per install)

50 chat messages sent by guests

179 content views

Graph of the time between invited and claimed

3 stopped (7%)

40 continued (93%)

Content usage

36 continued (84%)

0 rejoined

36 viewed the content list
25.9% of the funnel starters

3 stopped (8%)

33 continued (92%)

0 rejoined

33 viewed at least one content item in the app
23.74% of the funnel starters

Chat usage

33 continued (77%)

0 rejoined

33 read the chat
23.74% of the funnel starters

Figure 13: Guest invite funnel as used by Staying

Email delivery funnel

A more generic funnel is the one for email delivery. This funnel contains the following steps:

1. Email sent
2. Email delivered (confirmed by the provider)
3. Email read
4. Clicked a link in the email

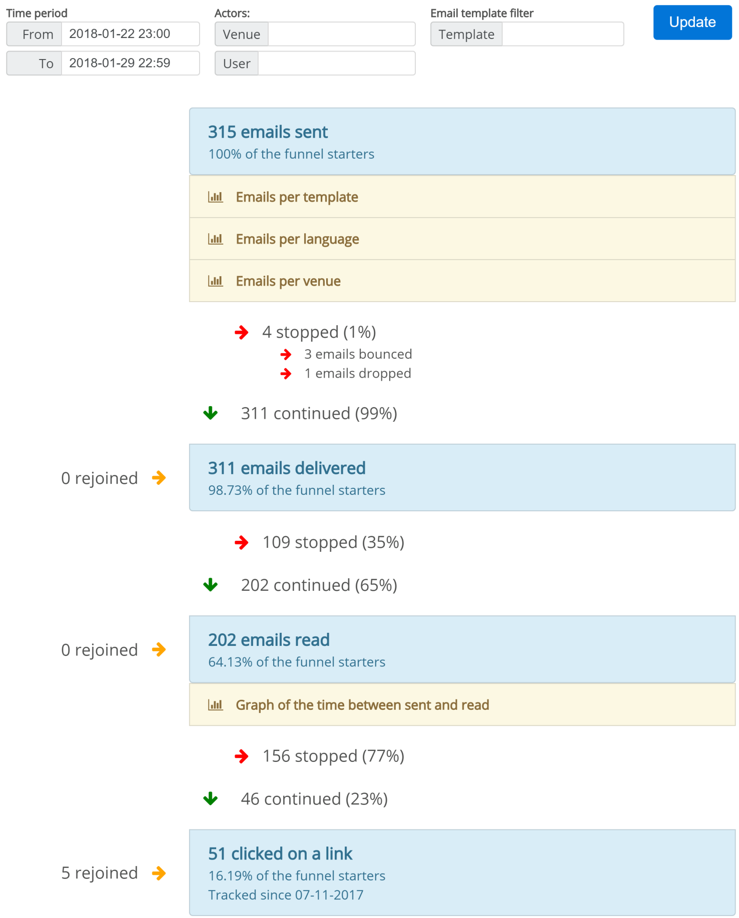
Figure 14 shows the complete funnel as used by Staying.

This funnel was created after Staying discovered a part of the emails they are sending are ending up in spam folders. Because the email provider used by Staying (mailgun[10]) gives feedback about undelivered emails, it is possible to track if emails are getting lost. After some improvements to the emails Staying is sending, and using a separate domain for marketing email, the percentage of delivered emails is around 98%.

With this funnel Staying also found out that a percentage of users click on links in emails, while the email has not been marked as read yet. This means that sometimes read tracking does not work, for example if images in emails are not displayed to the user.

The funnel has a filter for a venue, a user and the email template. The email template allows you to select an email from a list of options, like `guest_invite`, `unread_messages` etcetera.

[10] `https://mailgun.com/`



Time period
From 2018-01-22 23:00
To 2018-01-29 22:59

Actors:
Venue
User

Email template filter
Template

315 emails sent
100% of the funnel starters

Emails per template
Emails per language
Emails per venue

➔ 4 stopped (1%)
➔ 3 emails bounced
➔ 1 emails dropped

↓ 311 continued (99%)

0 rejoined ➔ 311 emails delivered
98.73% of the funnel starters

➔ 109 stopped (35%)

↓ 202 continued (65%)

0 rejoined ➔ 202 emails read
64.13% of the funnel starters

Graph of the time between sent and read

➔ 156 stopped (77%)

↓ 46 continued (23%)

5 rejoined ➔ 51 clicked on a link
16.19% of the funnel starters
Tracked since 07-11-2017

Figure 14: Email delivery funnel as used by Staying

SMS delivery funnel

This funnel tracks if SMS messages are delivered correctly, this funnel contains the following steps:

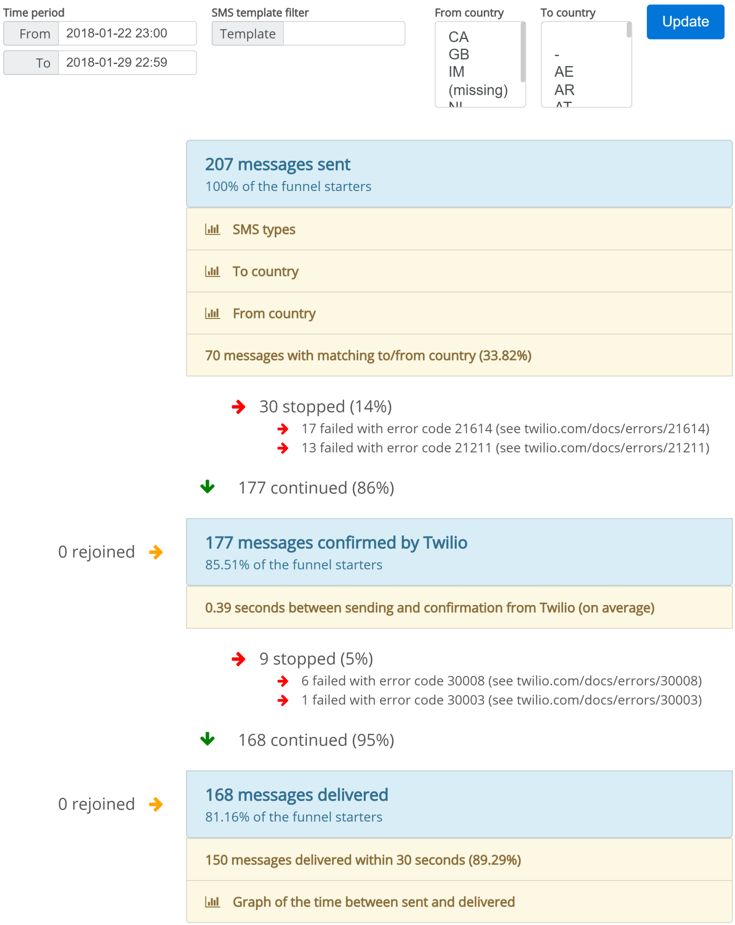
1. SMS sent
2. Message confirmed by the provider (Twilio[11])
3. Delivery confirmed by the provider

Figure 15 shows the complete funnel as used by Staying.

This funnel gives insight into how many messages are sent are properly delivered. It is often the case that phone number are entered wrongly, causing the message to get lost. With this funnel a couple of improvements to phone number handling have been made, for example stripping all non-number characters, so that entering spaces/dashes/braces does not cause the message to get lost. Even with these changes a percentage of the message still gets lost because of phones that are off, or providers that fail to deliver the message for other reasons.

[11] `https://twilio.com`

Staying also found out because of this funnel that sending messages to certain countries only works if a number from that same country is used, which helped to improve the delivery percentages.



Time period
From 2018-01-22 23:00
To 2018-01-29 22:59

SMS template filter
Template

From country
CA
GB
IM
(missing)

To country
-
AE
AR

207 messages sent
100% of the funnel starters

SMS types

To country

From country

70 messages with matching to/from country (33.82%)

30 stopped (14%)
17 failed with error code 21614 (see twilio.com/docs/errors/21614)
13 failed with error code 21211 (see twilio.com/docs/errors/21211)

177 continued (86%)

0 rejoined

177 messages confirmed by Twilio
85.51% of the funnel starters

0.39 seconds between sending and confirmation from Twilio (on average)

9 stopped (5%)
6 failed with error code 30008 (see twilio.com/docs/errors/30008)
1 failed with error code 30003 (see twilio.com/docs/errors/30003)

168 continued (95%)

0 rejoined

168 messages delivered
81.16% of the funnel starters

150 messages delivered within 30 seconds (89.29%)

Graph of the time between sent and delivered

Figure 15: SMS delivery funnel as used by Staying

6.3 Funnel display evaluation

As with the implementation of funnels, displaying them also has some trade-offs. As explained previously, funnels could in theory be full graphs. Although there are enough algorithms to layout and display graphs, it is not user friendly for a company that wants to read statistics about how many users read their newsletter. For the implementation splits and merges have been implemented, but not a full graph.

6.3.1 Testing in practice at Staying

As soon as a first version of the funnel display was working, it was shown on the admin dashboard Staying uses for statistics and support information. This has been done to immediately start testing how the employees of Staying use the funnel interface. Staying has four employees, with different levels of technical knowledge, as described below:

- Geert-Jan:
    - Developer and business, has knowledge about databases, PHP programs, frontends and event systems in particular
    - Uses the funnels for evaluating how the company is doing and to steer development
- Emiel:
    - Developer, has knowledge about databases, back-end and front-end development, and user experience
    - Uses the funnels to identify the steps to improve and spot problems in the product
- Agustina:
    - Designer, has knowledge about designing and user experience, has no technical knowledge about funnels
    - Uses the funnels for determining problems in the process the property owners and hotel guests go through
- Hugo:
    - Marketing and support, has knowledge about approaching new customers, dealing with feedback and identifying (user-experience) issues in the product
    - Uses the funnels to determine how well the invites of a property are going, giving feedback to them and identifying problems in the user experience

These employees of Staying all gave feedback about the interface and functionality of the funnels during the project. They have shown which information in the funnel is the most important, which in turn has been made more prominent in the interface. The notes functionality, and its different types are directly because of their feedback. For example, the effectiveness of SMS messages was low, leading to the demand of a graph displaying delivery times (number of seconds on the x-axis, number of messages that took that long to deliver on the y-axis). This graph in turn lead to adding some extra local phone numbers to send SMS messages from, which lowered the delivery time.

Currently the employees of Staying are happy with the funnels and the information they provide. The funnels have been crucial to determine the course of the company.

6.3.2 Basic statistics display

The most important statistics of a funnel are how many users made it to each step, which are displayed prominently in the implemented display. The next important thing is how many users continued, stopped and rejoined in each step (with percentages). This is displayed clearly using green arrows for continued, red for stopped and orange arrows for rejoined. Such a visual display is a good choice, and except for the ”rejoined” statistic users seem to understand it immediately. The rejoined statistic is harder to explain, and there is no good way to visually display it to be clear. So it seems like the current display is as good as it can get.

6.3.3 Splits and merges display

Displaying splits and merges clearly is important because if read incorrectly the user might overlook an alternative path and think that not much users make it, while in fact they just took an alternative path. Currently if the funnel splits it simply displays multiple funnels next to each other, which makes it clear they happen at once. Another option would be to display only one of the paths, and offer some kind of selection box to display the others. This would complicate the interface though and I expect that the other tracks would most likely be missed completely by the user reading the results in that case (considering that horizontal scrollbars on the web are not common). That is is why displaying the next to each other (possibly with a horizontal scroll bar, depending on screen size and the number of tracks) has been chosen. Displaying the merge clearly is also important, because it shows that the users got together in this step, so the statistics are again for the complete user group.

The split funnel display is not perfect, if the parallel tracks have a different length the shorter path does not reach until the merge bar. This might be solved with some long-stretched arrows that connect the two, so that it clearly displays the flow of users.

The merge/split mechanics have been implemented when at Staying I found out that funnels with a single path are not enough anymore. At some point I started making separate funnels for different invite flows, which all have the same first and last steps, which inspired the merge/split mechanics. This expands the simplified sequential funnels and gives more options to track users, which has proven to be useful at Staying. For example when inviting guests to install the app they are invited by email and sms, which is a perfect use case for this feature.

6.3.4 Notes display

Notes are displayed as one normal size text, a bit smaller than the results of the step itself, with optionally a secondary small text line (used for details about the statistic, like how it is calculated or since when it is valid). The fact that the notes are smaller than the step they belong to makes them less visually important, which is the desired effect. This does however mean that notes with bigger pieces of information like tables and charts do not fit. To solve this collapsible notes are used, which show one line by default to indicate what they contain, and expand when clicked. Figure 11 shows a step in blue, a collapsed note below it, and an expanded note at the bottom. Not all users initially found out that notes could be expanded, so this could be improved with arrows to indicate they can be expanded. Notes do however already change the cursor to a hand, indicating they can be clicked, which provides a good hint that something will happen when clicked.

6.4 Performance and scalability

This section describes the expected and realistic performance and scalability that the system has, and discusses ways to improve in the future.

6.4.1 Scaling

The intent of the project was to create the system while ensuring it can scale to a huge (Booking.com-scale) size. A couple of aspects of scaling will be discussed below to show how the system handles them, and what limits can be expected. It is however impossible to know for sure that the system can actually scale to a huge scale, because there will always be unforeseen issues when putting a way higher load onto a system. Even the government systems in The Netherlands often have trouble handling a high load, which shows that scaling is not a completely solved problem, and will always play a role when upgrading and maintaining systems.

Scaling in terms of processing power and storage can be done by adding more worker servers to the database. Depending on the way the database has been created the time this takes to do differs. For a Citus Cloud database it is a couple of clicks in a web interface to add servers. For a CloudFormation setup the 'AutoScaling' group can be changed to spin up extra servers, which still need to be manually added to the master with an SQL command (this could be automated with a small effort though). For a manual installation it depends on how the setup works, it could be simply running another machine and placing a certain image on it with all required software, or a server might need to be installed from scratch. After new servers have been added, they still do not contain any data. To change that the shards of the database need to be rebalanced, for that there is an automatic command `SELECT rebalance_table_shards('distributed_table_name');`. This command redistributes the shards, so that each server has a similar load.

A typical scaling problem is that certain tenants are a lot larger than the others, for example a large customer with a lot more traffic. This could cause certain workers be overloaded, while others are mostly idle. To solve this problem Citus allows to isolate a tenant to its own server, which solves this problem. The procedure to do this isolation is documented in the documentation of Citus[12].

6.4.2 Query performance

Query performance of a database is always an interesting measurement, but doing a raw performance benchmark of a database does not suite this research project. The performance of PostgreSQL has been benchmarked countless of times, for a lot of different use cases, so I don't think doing a benchmark would contribute much in that regard. There is however enough to be said about the expected performance of queries.

A couple of scenario's can happen with a CitusData setup. The first is that a query hits a single tenant, which results in execution on a single server, resulting in a speed like in PostgreSQL (with exception of the added latency by the *master* → *worker* → *master* communication). A second scenario is that the

[12] https://docs.citusdata.com/en/v7.1/admin_guide/cluster_management.html#tenant-isolation

query hits multiple workers, but the query is grouped by the tenant. This would hit all servers, which calculate a result, which is then combined at the master. This scenario scales the query nicely, since adding more tenants and servers would lead roughly the same performance since the query scales correctly. A last scenario is when a query does not scale nicely among tenants, which will cause high network load to calculate the results, this scenario is expected to be a lot slower. For the questions defined in the requirements the third scenario has not been hit, so that is at least an indication that often a good scaling query can be used to get the results.

6.4.3 Funnel performance

For the funnels there are two implementations. First the performance of the funnels in general are discussed, after which the good and bad performance of the two implementations will be discussed. Generally speaking the performance of calculating the funnel depends a lot of the first couple of steps. It is often the case that each step reduces the results a lot, sometimes even by 80%. Also the filters applied to the first step of the funnel influence the performance a lot. If the time period looked at by the funnel is small, only a small slice of the data needs to queried. It is especially important to have a good index on the time of analytics events, since that will make filtering data in a time restricted query quick. Following steps are mainly limited by the identifiers that resulted from the first step, which can greatly benefit from an index on the field that holds the identifier. For example when querying email identifiers, an index on that can make filtering following steps a significantly better.

Now on to the first implementation, which was proves to have trouble with scaling. This implementation normally worked within a second with thousand or less identifiers in the first step. However when moving to 10000 or more identifiers in the first step the performance got worse linearly. This is caused by the insertion of all identifiers into the query itself, which is unable to use indexes defined for that identifier on the table. This leads to sequential scans through the data, which will scale badly when the time period or number of events in the usage data increases.

The second implementation solves the problems of the first implementation by putting the identifiers into a (temporary) table instead of inserting them back into the query for the next step. This ensures indexes can be used correctly and prevents transferring the identifiers back and forth between the database and application. Creating temporary tables with a lot of rows could get problematic at some point, but generally speaking these can easily expand to millions of entries. The important factor for these tables are to keep them in RAM, because disk performance is a lot worse. So if the system has sufficient ram these temporary tables will not cause problems for a long time.

While the second implementation improves the scalability of the funnels by a lot, it does not makes them 'infinitely' scalable. To achieve better scalability a streaming solution could be implemented, which could even be made real-time. This is however outside of the scope of this research project and could be investigated in a future project.

7 Discussion

In this chapter the results of this project are presented and related work is highlighted.

7.1 Results

With this research project a solution is presented to combine usage and profile data to analyze applications of companies. The solution contains a database setup guide to bring the usage and profile data into a single system, a structured format for analytics events to make them suitable for analyzing, and a funnel implementation to show how to use this setup. For each of these aspects the design decisions have been documented with possible alternatives, so that they can be made suitable for a lot of companies. Next to designing and describing this system it has been tested with the company Staying to find out how it performs and which aspects can be improved. A list of improvements has already been made during the project, and the list of features and improvements that can still be done are documented below in the future work section.

7.2 Related work

[22] describes how data could be made immutable to keep history of all data and be able to recover values for any point in time. This principle will be required for profile data storage to combine it with usage data retrospectively without introducing errors. The paper [20] describes immutability in a more practical sense and provides ideas for implementing it into a database system. This principle can for example be used in HBase, which can store historic copies of column values (Google uses for webpage indexing and Maps).

The paper "Models and issues in data stream systems" [17] explains the fundamental problems for processing stream data and proposes query languages to process stream data and produce results on the fly. These kind of techniques are relevant for real-time processing that analytics systems do.

8 Conclusion

In this chapter the research questions will be answered, the limitations of the solution will be shown, the research domain and practical value outlined and the improvements that can be made in the future are listed.

8.1 Answers of the research questions

Now that the system is implemented and evaluated the research questions can be answered.

The main question asks how usage and profile data of a software product can be used more effectively for improving the usability of the product. The direction this research chose is to combine usage and profile data on-the-fly, so that a larger portion of the possible questions a company has about their product can be answered immediately. Retrospectively combining usage and profile data has been solved quite easily, using existing database techniques. To answer the first sub question about how effective combining the usage and profile data is for answering analytics questions, this has proven to help a lot for Staying. It has enabled the company to answer more advanced questions without additional waiting time caused by the need to change the usage data collection software. Other companies will most likely have similar types of questions that can be solved with this combination of both data sources.

The most important improvement for using tracking and profile data more effectively is building and visualizing funnels. The second sub question asks how business processes can be tracked and visualized in a funnel. For building a funnel first all steps of the business process need need to be written down. After that for each step the company needs to determine what counts as completing the step. Next to that the company can write down the important context information it would like to have for each step, to further determine which user types have continued or stopped there. After that a funnel needs to be defined with code, translating the process description into SQL queries that get the required data from the database (combining usage and profile data where necessary). After that the system visualizes the results of the funnel with a clear interface, showing the flow of users through the process, indicating where they stopped and continued.

8.2 Limitations of the solution

As with any system, there are some limitations that come with this solution. Below a couple of the most important limitations are shown.

The extension CitusData allows the database to scale to multiple servers, but also has its limitations. Citus cannot use the full SQL language, since some concepts do not work correctly when the data is divided among multiple servers. One limitation is that a Cartesian product is not supported, so joining the events table with itself without grouping or limiting it to the tenant column is not allowed. Since most of the time the tenant column will be used, this limitation does not often impact the usage of the system. There are also some workarounds listed in the Citus documentation, which helps to work around it. Sharding in

Citus also has some consequences, for example if certain events do not have the tenant column they cannot be placed anywhere, therefore it is hard to store and use.

There are also scaling limitations in the system. Generally speaking this database system needs a master node that is more powerful than an individual worker (about 3-4 times as shown in the insert benchmarks of Citus). This causes that the master node is the first to be problematic in terms of scaling. However as shown in the system validation chapter the system can support high loads already with some simple servers. The master node also takes in all traffic, routing it to the workers, so this means the network link of the master determines the maximum throughput. When using (multiple) fiber connections this does not need to be a bottleneck for most companies, assuming the Citus database is used for relational data and not for image or video storage (for which far better solutions exist). Another scaling limitation is that a single tenant cannot be split into multiple parts. This means that if one customer is a lot bigger than others, the best the company can do is isolate it to its own server. This separate server could however be made more powerful than a standard worker to handle the load, so that is a nice option Citus offers.

To apply this solution to existing companies there are a couple of steps to take, which influences how easily this solution can be retrofitted. The analytics database could simply be next to their current solution, and applications that generate events could be update to use the new analytics one by one. This could be a big operation depending on the number of applications and how they are deployed, but generally this should be possible. An option here is to parse the old analytics generation by an application and transform it to a new style event to insert it. The profile data is harder to import, since the requirement that it needs to be relational (or something that can be mapped to that). This solution also assumes there is a clear tenant in the application that it is applied to, which might not always exist.

The current system does not implement a history for profile data. This means that combining usage data with profile data will use the latest version this data, instead of the data that was there when the usage event was generated. This could lead to a skew in the results, but in practice this aspect turned out to be insignificant. In some cases combining the usage event with an up-to-date version of the profile data even works out better. For example when grouping results about users by their language, it is better to combine it with the users record where the language has been filled in later, than with the initial one that has no or the default language. This is however a concern that the company should take into account. It can be solved by storing old versions of the profile data and combining with the correct version. This does however complicate the system and increase profile data storage demands.

8.3 Research value and practical value

The research value of this research is in the experience with combining two data sources and making sure there is a good connection. There is also value in the analytics domain about structuring data, while making sure the structure does not limit the options of a company to collect anything they need. In the field of extracting statistics from a system and displaying them a lot has been learned.

One lesson learned is that usage data will always change and expand. This means that the storage solution for usage data needs to have flexible data format, that allows to add extra fields into the usage data. Another lesson learned is that classifying usage data is very useful for answering analytics questions and building funnels. Without classification of usage events, it is not possible to effectively filter and group the data, meaning certain questions cannot be answered.

During the case study at Staying it became even more clear that analytics questions companies have about their products are unpredictable. This fact needs to be taken into account during the design of the usage data storage. This is another reason for proper structure and classification of usage data events, which helps a lot when a new question by a company is about a data type that has not been used before. There is still a chance that the already stored usage data is not sufficient, but the chance will be lowered significantly.

A lot has been learned about funnels, in the area of collecting the required data and in the visualization domain. For collecting the right data it became clear that for properly tracking users through funnels, it is first required to be able to uniquely identify users. For simple funnels this was not a big issue, often there was a clear way to identify users through the complete funnel (such as an email address for simple email tracking). However for more complicated funnels, like the ones created for Staying, it was required to identify users with multiple identifiers at once. For example, for the long property sign up funnel of Staying, it is required to switch identifier a couple of times during the funnel. First they are identified by the marketing email number, after that a session number on the website, and at the end by the accommodation they created during the sign up. Next to this issue, splitting and merging funnels was a problem that appeared in practice. This has also been solved in this project, making the funnels more versatile.

For funnel display there are a couple of interesting insights. While funnels of other analytics tools display the steps horizontally, displaying them vertically does not degrade readability. The vertical layout also offers better options for displaying context information, which has proven to be quite important for Staying. For example, the guest invite funnel of Staying became a lot more useful for them when they could see the difference between mobile and desktop/laptop users. It became clear that a higher percentage of mobile users stopped, which lead to more testing of the invite flow on mobile. The result is that now the mobile invite flow works a lot better, improving the install rate of the app (which is an important KPI of Staying). Another lesson has been about displaying large context information blocks. If a viewer of the funnel cannot see at least two steps at once, it becomes a lot harder to see where users have stopped in the funnel. So it is important that big context information blocks are collapsed by default, so that they do not take up too much room. As last point, color and font size has proven to be important. The size of text indicates how important it is, together with the thickness and contrast compared to the background. This has been used to highlight the most important numbers in the funnel, while making context information more dim. Colors are used to indicate stopped and continued, which helps to immediately identify 'good' and 'bad' paths that users can take.

These insight can help future funnel building and visualization tools, ultimately helping companies to get a better view of the processes in their product.

Staying has used the funnels build during this project a lot. The first version were with live data directly, integrated into the admin dashboard. The results were immediately used to steer development, streamline marketing and improve support to users. The guest invite funnel and property invite funnel where used to improve both of these processes. For example, reducing the number of steps and making steps easier to understand. The guest invite funnel has been used by support to contact properties where conversion was low. For example, some properties had little to no content added to their app, causing guests to install the app, but rarely use it after the first install. Next to that the email funnel has been used to identify email delivery problems, leading to improvements to the emails so that they have a lower chance of ending up in the spam folder. The SMS delivery funnel showed that messages to the US and Canada were consistently failing, which identified a crucial issue: a phone number from The Netherlands cannot send message to these countries. This issue has been fixed by using an additional US phone number, improving the conversion of guests in the US and Canada.

8.4 Future work

To improve the presented solution there could be further research and development, the directions that I deem most interesting are described below.

The current solution can only provide results for a certain time period, and generates these results on each request. It would be interesting to investigate the possibility to build a real-time funnel. This would mean that an addition needs to be made to process incoming events directly, and that the interface needs to be adapted so that it can be updated immediately. Emiel from Staying already experimented with some techniques in PostgreSQL that could form the basis for a real-time system. He proposed to define aggregate functions on the events table that return which steps a certain user (or other identifier) has made, which could be reprocessed when a new event comes in that matches this identifier. For updating the interface in real-time there are plenty of options, for example a backend that accepts a websocket, with a reactive front-end that connects to that backend.

Instead of the current funnels that are sequential with the addition of split/merge semantics a full graph solution could also be implemented. This would give some challenges in tracking all paths and displaying them properly (maybe smartly hiding paths that are not useful).

Another possibility is to do more performance testing and benchmarking with the system to see how it performs under different circumstances. This could mean applying it to another company to find out how the system would work there and to see which adaptations need to be made to build the system to the companies needs.

9 Acknowledgments

I want to thank Marten van Sinderen from the University of Twente for the feedback, support and critical questions during the complete project. Thank you Arend Rensink from the University of Twente for providing your viewpoint of my project and learning me to clearly explain the project. Next I want to thank Staying for the great opportunity to do this project with them, in the center of Amsterdam. Thanks Emiel Mols and Geert-Jan Bruinsma of Staying for the technical advice, feedback and support during the project. Also thanks to Hugo and Agustina for their use of the funnels and feedback about them. Lastly, thanks to all other people that helped me shape this project and influenced my view during the architecture and implementation phases.

10 References

[1] Alexa. http://www.alexa.com/. Accessed: 6 April 2017.

[2] Amazon redshift. https://aws.amazon.com/redshift/. Accessed: 29 March 2017.

[3] Bigtable. https://en.wikipedia.org/wiki/Bigtable. Accessed: 29 March 2017.

[4] Citus (multi-server postgresql). https://github.com/citusdata/citus. Accessed: 31 March 2017.

[5] Cloudflare scaling postgresql with citusdb. https://blog.cloudflare.com/scaling-out-postgresql-for-cloudflare-analytics-using-citusdb/. Accessed: 31 March 2017.

[6] Db-engines. http://db-engines.com/en/ranking. Accessed: 31 March 2017.

[7] Document-oriented database. https://en.wikipedia.org/wiki/Document-oriented_database. Accessed: 31 March 2017.

[8] Ghostery, tracker blocking extension. https://www.ghostery.com/. Accessed: 9 October 2017.

[9] Global internet usage. https://en.wikipedia.org/wiki/Global_Internet_usage. Accessed: 29 March 2017.

[10] Google file system. https://en.wikipedia.org/wiki/Google_File_System. Accessed: 29 March 2017.

[11] Graph database. https://en.wikipedia.org/wiki/Graph_database. Accessed: 31 March 2017.

[12] Key-value database. https://en.wikipedia.org/wiki/Key-value_database. Accessed: 31 March 2017.

[13] Mysql cluster cge. https://www.mysql.com/products/cluster/. Accessed: 31 March 2017.

[14] Relational database management system. https://en.wikipedia.org/wiki/Relational_database_management_system. Accessed: 31 March 2017.

[15] Usage of traffic analysis tools for websites. https://w3techs.com/technologies/overview/traffic_analysis/all. Accessed: 6 April 2017.

[16] Wide column store. https://en.wikipedia.org/wiki/Wide_column_store. Accessed: 31 March 2017.

[17] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 1–16. ACM, 2002.

[18] K. Boda, Á. M. Földes, G. G. Gulyás, and S. Imre. User tracking on the web via cross-browser fingerprinting. In *Nordic Conference on Secure IT Systems*, pages 31–46. Springer, 2011.

[19] K. Choroś. Further tests with click, block, and heat maps applied to website evaluations. *Computational Collective Intelligence. Technologies and Applications*, pages 415–424, 2011.

[20] P. Dadam, V. Y. Lum, and H. Werner. Integration of time versions into a relational database system. In *VLDB*, volume 84, page 509522, 1984.

[21] W. Fang. Using google analytics for improving library website content and design: A case study. 2007.

[22] P. Helland. Immutability changes everything. *Queue*, 13(9):40, 2015.

[23] A. Lerner, A. K. Simpson, T. Kohno, and F. Roesner. Internet jones and the raiders of the lost trackers: An archaeological study of web tracking from 1996 to 2016. In *25th USENIX Security Symposium (USENIX Security 16)*. USENIX Association, 2016.

[24] A. Moniruzzaman and S. A. Hossain. Nosql database: New era of databases for big data analytics-classification, characteristics and comparison. *arXiv preprint arXiv:1307.0191*, 2013.

[25] H. Pakkala, K. Presser, and T. Christensen. Using google analytics to measure visitor statistics: The case of food composition websites. *International Journal of Information Management*, 32(6):504–512, 2012.

[26] B. Plaza. Google analytics for measuring website performance. *Tourism Management*, 32(3):477–481, 2011.

[27] K. Shvachko, H. Kuang, S. Radia, and R. Chansler. The hadoop distributed file system. In *Mass storage systems and technologies (MSST), 2010 IEEE 26th symposium on*, pages 1–10. IEEE, 2010.

[28] R. Upathilake, Y. Li, and A. Matrawy. A classification of web browser fingerprinting techniques. In *New Technologies, Mobility and Security (NTMS), 2015 7th International Conference on*, pages 1–5. IEEE, 2015.

Appendices

A Analytics tools features

This chapter which types of analytics tools are available, which features they offer and how they are organized technically.

A.1 Analytics tools selection

To find out the software architecture of commonly used analytics tools, it is important to first know which analytics tools are used most. Based on the market share calculated by W3Techs [15] (which is based on the trackers used by the top 10 million websites determined by Alex [1]) the tools below have been selected. Next to that some tools I encountered in practice have been added (like Amplitude used by Happening[13] and Heap suggested by Emiel Mols from Staying). The list of tools is by no means exhaustive, but does represent a significant market share of used analytics tools and should give a good picture about common features.

A.2 Investigated analytics tools

Table 2 shows the analytics tools that have been investigated, with their most relevant properties. The application and app types the tool can track are listed to ensure compatibility with Staying. The used database implementation is shown (when known) to use for the architecture.

Table 2: Analytics Tools

Name	Platforms	Storage	Remarks
Google Analytics	Tracking of websites, iOS apps, Android apps and custom applications.	Stored in BigTable [3] on Google File System [10].	
Yandex Metrica	Tracking of websites and iOS, Android, Windows and Unity apps.	Unknown	Compatible with Google Analytics clients.
Amplitude	Tracking of websites, iOS apps and Android apps.	Stored on Amazon RedShift [2].	
MixPanel	Tracking of websites, iOS apps, Android apps and custom applications.	Unknown	
Heap	Tracking of websites and iOS apps.	Stored on Amazon RedShift [2].	Collects all user interaction, define events. retroactively.

The following sections will look at the features these tools offer, the data they store and the software architecture. These aspects are most important since

[13] `https://happening.im`

combining the usage and profile data requires us to know what will be combined and how this could be achieved technically.

A.3 Analytics tools concepts and features

In this section relevant analytics definitions are explained, which are related to the features offered by the investigated tools.

Events, sessions and users: On a low level events are collected, which represent a single action performed by the user on the website or app. These events identify what happened and provide context information to give more detail about the why, where, when and how. Events are grouped into sessions, which are collections of events that happened short by each other by a certain user/device. Other actions like killing an app or leaving a website can also close sessions.

- Event: opening a page (also called pageview or hit), clicking a button, etc.
- Session: aggregation of all events by a certain user that happened close to each other in time.
- User: individual person recognized by a login (unique identifier for a user, like a user id, username or email) or device (if the website does not use logins or the user did not create an account yet).

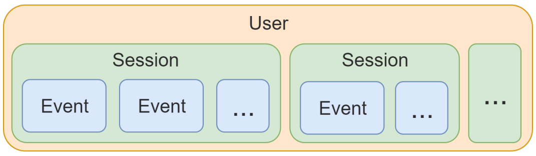


Figure 16: Users, sessions and pageviews

These events, sessions and users are to provide charts with the number of events per time period, number of users visiting per time period and statistics like returning versus new users.

User data: Details about a user to user for filtering of results.

- Age (approximated or indicated)
- Gender (guessed or indicated)
- Location (based on ip address)
- Language
- Device information:
  - Device type: desktop/laptop/mobile/tablet
  - Screen resolution
  - Screen size
  - Operating system (including version)
  - Browser

Segments: Subsets of the user data. Segments are defined using the information available about users, sessions and events. For example a segment could be created for users with a paid subscription. For these users different statistics might need to be tracked than for free users and they probably have different behavior (accessing paid-only features on the website/app).

Analytic tools offer extensive segmenting options for all graphs and statistics, to allow companies to view this information for a certain audience. This audience segmentation could for example be based on the purchases the user made in the application, subscription they have or location.

Funnels: Flows defined by the user of the analytics tool to analyze a certain row of actions in the website/app. For example a flow to track the signup process consisting of entering email/password, adding an avatar and start using o. After this flow it could continue by checking how many of these people perform another action based on if they uploaded an avatar.

All of the listed tools offer functionality that allows tracking funnels. This means defining the steps of the funnel, after which the number of users entering the funnel, users completing the funnel and users that stopped can be seen.

A/B testing: Tracking the reaction to two or more different versions of a website, to find which version works better. For example letting the user enter the password once or twice during signup, to see in which situation more people continue.

The analytics tools can show statistics based on segments as described above, which can also be used to show statistics for users that are in a certain A/B testing group, so that a comparison between the versions can be made.

User tracking: To track users the analytics system needs to know their identity. This is often achieved by sending a user id of some sort along with the usage data. This does not cover all cases though, for example during the signup process the user id is not yet known, since it is not yet created. Therefore analytics tools often use other heuristics to combine data for a user. This could be using ip address, device details (brand, model, supported technologies) or other information. In this area there a lot of research, on the fingerprinting side and also on the prevention side. Paper [28] classifies different techniques to fingerprint users and [18] describes how users can be tracked even when they use different browsers. Using these techniques users could be tracked from the initial visit of the website of the company, through the sign up form to the analytics produced after they have a user id.

B Analytics system internals

B.1 Requirements

Now that the data analytics tools collect and store is known, the software architecture of these tools is investigated. The main requirements for an analytics system are based on the important features and technical details of existing systems. These requirements will stay relevant when designing the system that will combine the usage and profile data. The requirements are listed below, along with the reasons they are relevant.

R1: **Technology independent event source:** support websites, apps and if possibly anything else that can send data using HTTP (for example backend servers that run the application and report status information and errors/warnings).
Required because companies want to track all facets of the application, therefore it should be possible to collect information from all sources. Now could the company use multiple analytics solutions, but generally having a single one is preferred (less dependencies, can combine data from different sources, less maintenance and education cost).

R2: **Scalable event intake:** Should be able to handle a big number of events to support tracking big websites and apps.
Required to make the tool usable with a large number of clients, supporting big apps and websites.

R3: **Scalable storage:** Store a huge amount of data (multiple servers and data centers).
Ensures that storage is 'infinitely' scalable and all events from the past can be kept for later analysis.

R4: **Real time filtering and analysis:** Process incoming events directly, for example monitoring errors.
Facilitates real-time results from the incoming events, for example notification when errors occur.

R5: **Historic data analysis:** Process a subset of events, selected by time and/or contents.
Ensures that there is an efficient way to analyze saved events, for example to generate usage charts for the last month or year.

R6: **Result export:** Exporting and saving the results of real-time and historic processing.
Required to be able to use the results in a meaningfull way, like showing them on a website.

Now that the most important requirements are known an abstract system architecture of an analytics tool will be created in the next section.

B.2 System architecture

Keeping in mind the requirements of the previous section an abstract system architecture has been created, visualized in Figure 17. This architecture shows what kind of system most companies that use analytics have, either as in-house solution or provided by one of the companies investigated in Chapter A.2. Having a clear idea how an analytics tool is organized technically is critical for developing a solution that can combine the usage and profile data, because now I can ensure the requirements of an analytics tool are still met in the final system. The architecture starts at the top where the events are generated, after which they are stored, processed by a real-time filter or historic scanner, and in the end results (reports, charts, etc.) are stored. Below these components are explained in detail.

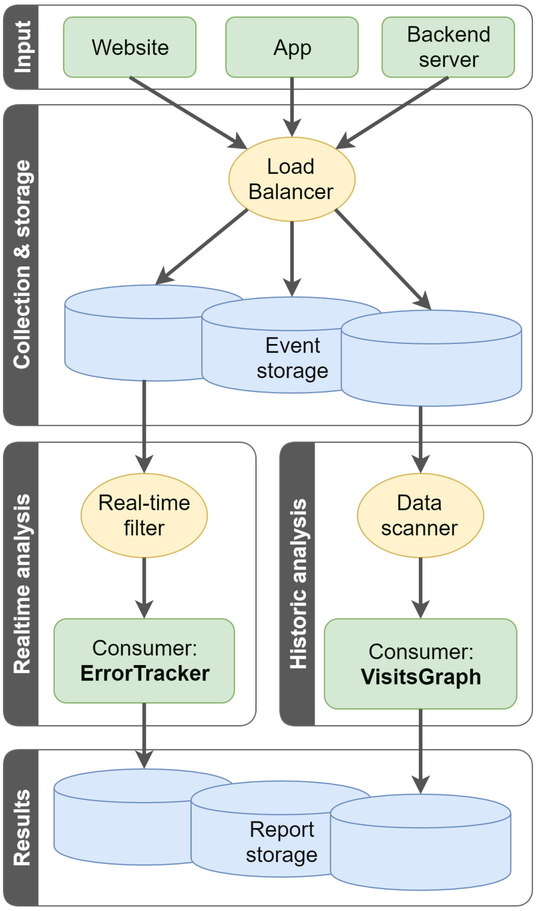


Figure 17: Event collection and processing system

B.2.1 Input

At the top Figure 17 starts with the input. These are the applications like websites, apps and servers that generate events. Apps can send data about its usage (startup, screen views, encountered problems), a web server can send data about its operation (response time, uptime, encountered problems) and a notification server can track notification usage (sent, read, discarded, unable to send). Data from these different sources should have some common properties, like the source and time of the event. But all other data depends on the context and should be flexible. For example a web server should report its name, ip address and data center, while apps should report the logged in user, device type (brand, model, size, etc.) and currently visible screen. This part fulfills R1.

B.2.2 Storage

After events are generated by a server or client, they need to be stored. The pictured load balancer is there to spread the intake on these servers and data centers, fulfilling R2. After the load balancer a database needs to store the events, for this there are many options. Below are the most used database implementations listed per type, based on current popularity of them calculated by DB-engines [6] and a comparison of NoSQL databases [24].

Relational [14]: Oracle, MySQL, Microsoft SQL Server, PostgreSQL, DB2, Microsoft Access, SQLite, RedShift (Amazon), MariaDB, Teradata.

Key-value stores [12]: Redis, Memcached, Riak, Voldemort (LinkedIn), BerkeleyDB (Oracle).

Document databases [7]: MongoDB (BSON), Dynamo (Amazon), Couchbase, CouchDB (JSON).

Wide column stores [16]: Bigtable (Google), Hypertable, Cassandra (Facebook; used by Digg, Twitter); SimpleDB (Amazon), DynamoDB (Amazon), HBase (by Apache).

Graph databases [11]: Neo4j, InfoGrid, Sones GraphDB, AllegroGraph, InfiniteGraph.

Not all of these databases are suitable for analytics evens storage. For example Memcached and Redis only store data in-memory, which will not work because of the required persistence and capacity. Graph databases also seem unsuitable, since event storage has no strong need for relationships and therefore does not fit the purpose of these databases. Some databases do not scale to multiple servers and/or data centers (as required by R3). Examples are MySQL (needs MySQL Cluster [13]), SQLite (limited to a single file), PostgreSQL (needs something like CitusDB [4], as used by CloudFlare [5]).

B.2.3 Analysis

Now that events are stored they can be analyzed. There are two main aspects in event processing, doing it real-time as the events come in, and processing historic data. For real-time processing it makes sense to put a filter at the event storage intake, which forwards relevant events to an application that processes them. This could for example be used for monitoring the number of errors that come from the web server, to send email notifications if there are more than X per hour. This system implements R4.

Historic processing could be monthly app usage reports, long-term usage statistics or results of A/B testing. This can be organized by searching through all events within a certain time period (assuming events can easily be filtered by time, which makes sense for analytics systems). This processing does not need to be very quick, but does need to be scalable to for example be able to generate yearly reports. R5 is implemented with this system.

In this project the focus will be on historic data processing to limit the scope of the project. Real-time is kept in mind as future feature in the design of the system, but will not be tested and benchmarked.

C Staying analytics solution

In this appendix the analytics solution Staying currenlty uses is described, to get an idea what the starting situation of Staying is.

C.1 Architecture

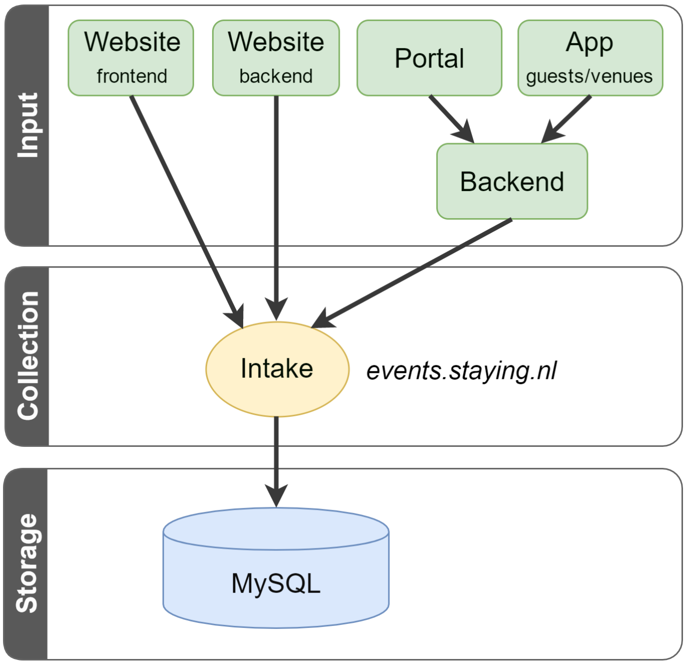


Figure 18: Staying analytics overview

First an event is generated in one of the applications, as seen at the top of Figure 18, an event is a piece of JSON data containing information about what happened. The website backend events are generated by the web server when the web page is requested, the website frontend events are generated in JavaScript at the device of the visitor. The portal events are generated in JavaScript and bundled together before sending them to the backend, which will unpack them and send them to the event collection server. There are two apps, one used by guests to see their stays and use the chat, and one for the venue owners to answer the chat messages. These apps are deployed on Android and iOS, and both send bundles of events to the backend, like the portal does. The events are send to `events.staying.nl` where they are collected and stored in a database.

C.2 Content

Events have a defined structure as shown in Listing 37. This structure is defined to organize the different event types and the information they hold. The event is a JSON object, in which at top-level there is an `epoch` key indicating the time of the event.

Next there is `system` top-level key in the event. Inside this key there is once again a `type`, which in this case indicates the application that has generated the event (portal, app, backend, website, etc.). Generally each of these have an ip, which is why the ip is stored immediately under the system key, but all other details are in an object placed at a key which has the name of the type.

Because events will often get filtered by user and/or venue these have a place at the top-level of the event structure. The `user` and `venue` fields contain an id that links the event to the user and/or venue in the application database. Both of these fields are optional, since both of them could be unknown or irrelevant. For example events that report performance statistics of a server don't have either of these.

The above explained the basic event metadata, but the main content is stored as follows. The top-level `type` key contains a string value indicating the type of event. In a top-level key with the same name as the set type there is an object with more information it. The structure of this type-specific data is also defined, for example a `request` has a `method` and `url` to indicate which resource has been retrieved or updated. Another example are the `action` events the portal and apps send which have information about the user action that happened. The action top-level key takes the same approach as the top-level, it has a `type`, and a details object that uses the given type as key. This details object for example contains the title of the button in case of of a `button_click` and a modal title in case of a `modal_open`.

Listing 37: Analytics event structure rules

```
##### Rules
# - 'type' indicates that an object with that key can exist with more
    information
# - keys are lower case, with underscores
# - derived data has a trailing underscore, all derived data can be
    regenerated from non-derived data

# Generic information
epoch: number # Event time (seconds since 1970)
system:
   type: string # portal|app|backend|website|..., each running
       application
   ip: string # ip of the event sender
   agent: string #
   app:
      agent: string
      bundle_id: string
      # Other device information...
   website:


user?: number # user id
venue?: number # venue id
stay?: number # stay id
guest?: number # guest id

# Type and information for that type
type: string # action|request|email|..., object with type as key should
    exist (or save as generic 'data' key?)
```

```
action?:
    url: string # url the portal/app is showing
    type: string
    client: token[0:4]
    image:
        url:
        # context...
request?:
    method: string
    url: string
    took: number # milliseconds
    status: number # HTTP status code
    length: number # bytes written
    cookie?: object
    user_agent: "Mozilla 123"
    user_agent_info_: # user agent information lookup
        ua_type:
        ua_name:
        # ...more
    ip: string
    ip_info_: # Ip information lookup
        city:
        region:
        isp:
    referrer?: string # referrer string
email_send?:
    id: string # Identifier to track this email
    from:
    to:
    content:
email_delivered:
    id: string # Same identifier as 'email_send'
email_dropped:
    id: string # Same identifier as 'email_send'
email_bounced:
    id: string # Same identifier as 'email_send'
email_read:
    id: string # Same identifier as 'email_send'
```