POLITECNICO DI TORINO

DEPARTMENT OF CONTROL AND COMPUTER ENGINEERING

Master's Degree Course in Computer Engineering

Master Thesis

Creation of a web platform for event-flow visualization

Supervisor
prof. Maurizio MORISIO

Candidate
Concetto Antonino PRIVITERA

Company advisor
Criteo
Warren SEINE

APRIL 2022

Abstract

Context Big Data is becoming day by day one of the most crucial topics in the business world. An example of this is companies that collect a large amount of data of all kinds. However, having these large databases also means that their management is important, especially when the success of the company relies on them. Many tools allow for in-depth analysis such as the use of simple graphs or more complex software like Tableau. However, time-dependent data, for instance logs, are more complicated to analyse with the tools already available. Therefore, creating a tool for this specific case was necessary to include an additional variable in the analysis, time.

Objectives Data analysis is starting to be a main area. Indeed, it can be used to increase the revenue of a small or large company. Moreover, it can be employed for any machine learning model, as the latter needs to be trained by a reliable data source. This process is not easy and the training may not be perfect due to data noise or wrong configuration parameters used. For this reason, it is important to analyse the data and understand its meaning. Although there are several tools for this, it was necessary to create a specific application to manage time-based data. The objective is to create a new web platform that can show a flow of events through a timeline. In addition, it has to allow each visualization aspect to be modified for deeper analysis.

The solution This application was developed using the latest available technologies. In particular, the front-end is based on the D3JS and React frameworks that allow the interface to be automatically updated if the data undergoes a change. Secondly, the Scala-based back-end allows the application to contact the DBMS in a controlled environment. Since the amount of data is very large and a large amount of computation is required to recreate a timeline, the DBMS used is Vertica, one of the fastest DBMSs in the world in terms of query execution time. However, a timeline could be very long and the browser might not be able to handle so much information, especially since memory is limited and each request is limited by a timeout. Therefore, a compromise between functionality and complexity had to be found in order to allow sufficient customisation of the analysis and execution speed.

For this reason, each timeline is built on demand. In other words, everything visible is requested on Vertica and saved in memory, the rest is not displayed at all in order to reduce the requirements of the application.

Conclusions The software has been developed with the aim of fulfilling every functional and non-functional requirement. Not only is it possible to display data in the form of timelines, but there are also features that help the user understand what is being shown and what it means. Furthermore, a large part of this project was developed with the idea of making it modular by trying to automate the required steps. This will make each new function easy to implement.

Acknowledgements

English *After a long journey to obtain a degree in computer engineering, I want to dedicate all this work to those who have been there for me from the beginning and have believed in me. A warm thank you to my mentor professor Maurizio Morisio for his availability and for always being a guide in my studies. I would like to thank my family, especially my father and mother, who have allowed me to get this far by trying to support me not only financially, but also morally, especially in the most difficult times. I would also like to thank my brother Giovanni and my sisters Concetta, Mimma and Maria who have given me the courage and strength to accept and undertake new adventures. Last but not least, I cannot fail to mention all my friends and those I have worked with, near and far, who have spurred me on to improve day after day. Thank you all from the bottom of my heart!*

Italiano *Dopo un lungo viaggio per conseguire la laurea in ingegneria informatica, voglio dedicare tutto questo lavoro a coloro che mi sono stati vicini fin dall'inizio ed hanno creduto in me. Un caloroso ringraziamento va al mio relatore il professore Maurizio Morisio per la sua disponibilità e per essere sempre stato una guida nel mio percorso di studi. Vorrei ringraziare la mia famiglia, in particolare mio padre e mia madre che mi hanno permesso di arrivare fin qui cercando di sostenermi non solo economicamente, ma anche moralmente, soprattutto nei periodi più difficili. Inoltre, ci tengo a ringraziare mio fratello Giovanni e le mie sorelle Concetta, Mimma e Maria che mi hanno dato il coraggio e la forza di accettare ed intraprendere nuove avventure. Infine, per ultimi ma non meno importanti, non posso non nominare tutti i miei amici e coloro con cui ho lavorato, vicini e lontani, che mi hanno spronato a migliorare giorno dopo giorno. Grazie a tutti di cuore!*

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The analysis of data is playing a big role in many companies, especially in the last years since their business could lies on it. Different tools are used for performing it that range from simple charts, like line or pie charts, to more complex software like Tableau. However, they usually lack of specific functions or they are too complex to use for specific cases, so it was needed to create an application to analyze temporal data which allows to run a event-flow visualization.

The project concerning this thesis was reasoned and developed at **Criteo**, a company specialized in **advertising** with various headquarters spread around the world. Specifically, it displays targeted ads on the web.

The goal was to design and develop a web platform to visualize event data such as user interactions and server-side events. In particular, I had to create a timeline able to display time-based data in a chronological order based on some filters and user interface preferences. If we consider for instance a banner shown by Criteo in a website, a user could interact it by clicking elements, scrolling, or opening and closing menus. All of them are events triggered in a certain time that make together a sort of user history. However, there are many more users who interact with the banner. So, the timeline must be able to get the single histories of events generated by each user and put them together to make a unique timeline that shows the general trend of all users.

It was needed to find a good compromise between functionality and complexity in order to have a big range of features and great performance at the same time. Indeed, given the large amount of data, like billions of records of any kind, it was not easy to display it, especially because the browser cannot handle so much data. So, a way had to be found to load and visualize these datasets without affecting the performance of the browser so much. This way, the timeline can run on any device and be used without any particular problems. Therefore, research on user interface issues was underway to find a solution for this.

In this chapter it will be introduced the company, the platform Relook in which this project depends on and finally the general structure of this thesis.

1.1 Introduction Criteo

To give context to the project and understand why it is useful for analyzing data, it is important to describe Criteo's internal organization and how they gather information.

It is an advertising company with the goal to provide target ads on the web. It was founded in Paris in 2005 by Jean-Baptiste Rudelle, Romain Niccoli and Franck Le Ouay[6]. After that, further branches were opened in various other countries such as Germany, Italy, Spain, USA and Japan [10], so that today there are about 3000 people employed[6]. Each of them focuses on one or more departments, which will be explained in the next section.

1.1.1 Infrastructure

Criteo is equipped with a distributed hosting infrastructure (50000 servers in bare-metal) with various features about code automation, design, deployment and maintenance. So this infrastructure has many data centers, namely a physical infrastructure to house servers and everything needed for the company's services in multiple areas of the world.

In addition, the company has huge traffic of requests, which is more than 10 thousand displays per second. As a result, Criteo collects a lot of data (several Petabytes), which is stored in a large Hadoop Cluster, hence the hardware is upgraded[9] every 6 months by experts. This means that the servers are more powerful after every upgrade and therefore can handle the same load with less power. This results in fewer servers being needed.

The software to develop depends a lot on it since one of its main aim is to get data from these servers. The amount of data and the performance of the servers could affect a lot the final result on user experience. Then, it was needed to analyze the possible solutions in comparison to the infrastructure of the company.

Criteo has many hosting partners and the number of servers in each area is based on the number of customers and other aspects, important for the analysis tools when you want to filter data on basis of them.

1.1.2 Advertisement

Criteo is a company that derives its main revenue from advertising, so it aims to show banners of all kinds on the web, both in the browser and mobile app. To clearly explain how it works, it is important to introduce two main characters that are closely related to Criteo's business: Partners and Publishers.

Partners are Criteo customers who are willing to pay to see their ads on the web. There are several tools they can use to describe how a banner and which products should be displayed to a user. They can enable or disable the recommendation system based on their preferences, and they can also add or remove some campaigns for specific products. These are just a few examples of the features they have. For instance, it is possible to decide if they want to limit a certain campaign to some countries.

Once a banner is created and configured correctly, it must be displayed on the web using these parameters. Of course, the banner is not visualized on just any website, but on the websites of the publishers. These publishers are paid by Criteo to reserve an advertising space on their web page for the display of a banner. The figure 1.1 is a schematic of the whole process. As we can see, the system is actually much more complex and not so immediately understandable.

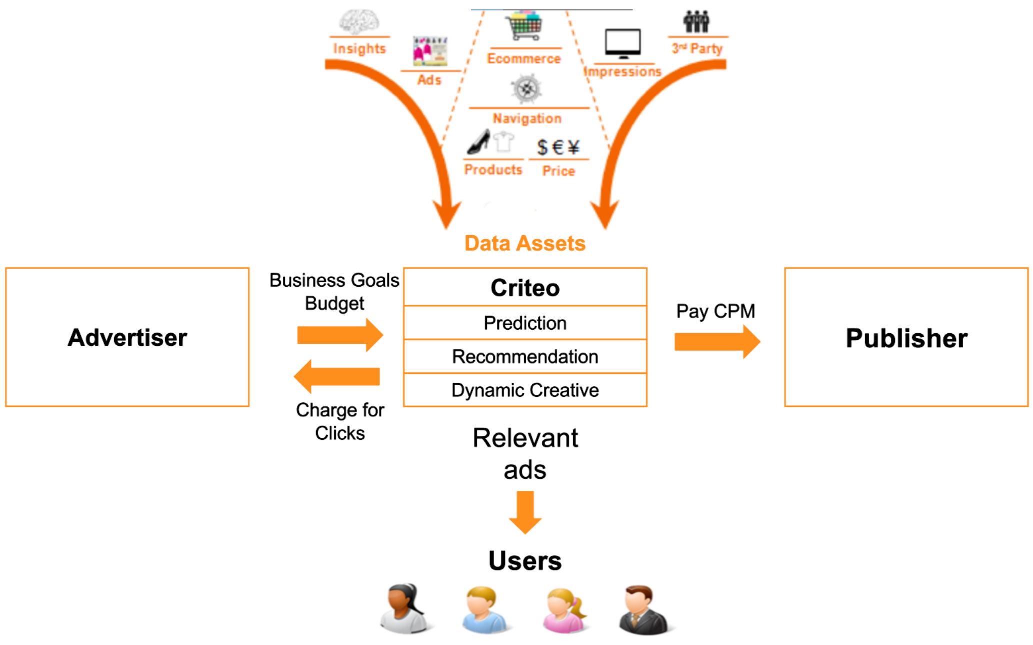


Figure 1.1: Advertisement engine[4]

Criteo manages more than one partner. This means that if there are multiple banners from different partners that are appropriate for the user, then the Criteo engine has to decide which one to display. The solution is an **Real-Time Bidding** (**RTB**). It is an automated system that makes Criteo bid on behalf of the partners to show their banner on the website reserved slot. This process is done in a few milliseconds and works based on a few parameters. Criteo tries to keep a network latency of **100ms** and places its servers where the other competitors are located (e.g. Google, Facebook, ...) due to population density.

This process is quite complex since there are many requirements to satisfy like showing the correct banner on basis of user preferences or fast response time. For this reason, this ads system can benefit of a recommendation system based on machine learning models. However, their configuration is not easy to set up if there is no knowledge of data owned. So, many different analyses have to be done to find anything that could be useful. There are tools that allow to show, for instance, how the users are split in the dataset and if there is some so-called *noise* that could break the model training.

In this case, Criteo collects a large amount of data, specifically events, that are time-based which means they have a timestamp and assume a specific meaning on basis of *which* and *when* these events happened. They range from internal events such as server logs to external events such as user interactions events. In particular, the latter is gathered by a service called **CSM** which means "Client-Side Metrics". When a banner is delivered and shown to an user, this service collects a lot of information from the user interaction done through the banner. This is possible by injecting some JS code which is in charge to listen specific events such CLICK or SCROLL that could happen on any position and on any element. For this reason, not only record the type of event triggered, but also its timestamp, position and other details useful for the analysis.

This service is useful for many aspects of the company. For instance, it helps to adjust the price of a banner when it's clicked or in the view-port, improve banner design or perform a better analysis.

1.2 Relook framework

The project is powered by **Relook**. it's a in-house data visualization framework that is developed internally by the planning and reporting team, and that is used to build dashboards and to provide self-service capabilities to management center, publisher management center and retail media. It is composed by different tools to make dashboards as web platform.

Initially, Criteo was using an external framework called **Looker**. However, Google acquired this tool and forced all its users to push their data on the Google Cloud Platform. So, it was not acceptable for the company for different reasons due to marketing competition and privacy, so they decided to make their own tool called indeed Relook.

It is developed with the goal to allow any user to create custom dashboard very easily. An example of dashboards in 1.2.

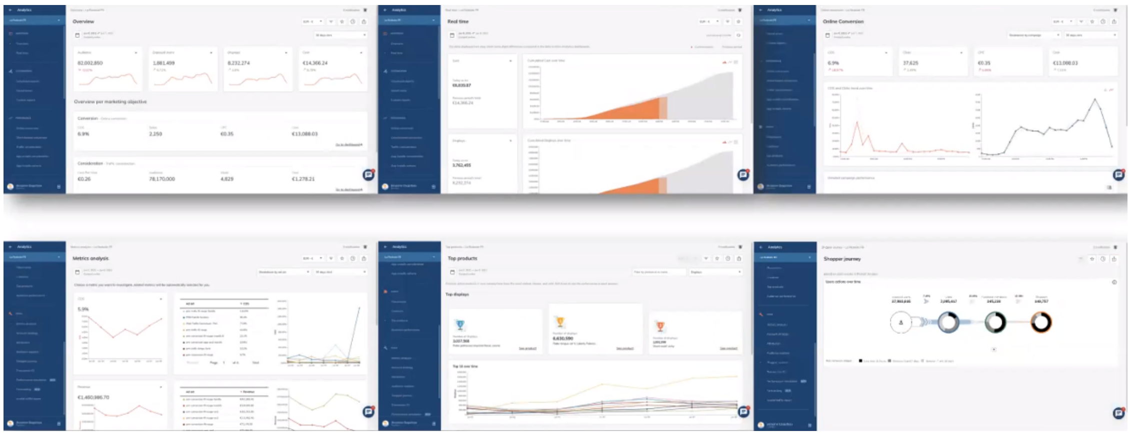


Figure 1.2: Relook dashboards example

Thanks to such tool, it is possible show different kind of data ready for relook such as page views, product impressions, clicks, cost or sale transactions. It is also possible to define how to visualize them, for example such as a bar chart or a table.

There are different features available that it may help the user to run a better analysis. it is possible to share the dashboard through a link with another user. In this way, sharing our own findings is easy and possible to do. Then, you can also save a view and load whenever you want. And finally, there is a way to enable a schedule. In other words, it is possible to ask the system to get a dashboard visualization in a specif interval of time without opening the web platform and then load it afterwards any time. The last but not the least, it also offers way to filter data to make the visualization more dynamic on basis of our preferences. So, they are features that are in common with all Relook dashboards.

However, the project developed and described on this thesis could not directly use anything of it since its internal mechanism requires a different data treatment in the back-end and a different logic behaviour in the front-end. So, it was needed to implement this powerful tool by using Relook as base and trying to integrate the application as much as possible in a way to use at the maximum some framework features available. Indeed, given the large amount of data and the complex operations needed to run on it, the application required a different internal architecture to make it usable without issues of any type like memory errors or timeout.

1.3 Event-Flow Visualization

The work made for the project is described in the next chapters and it is focused on the addition of many new tools, but in particular the creation of a new visualization platform able to show a flow of events through a timeline.
It's inspired by a Google Analytics tool called Path Analysis, which is able to show user navigation through the different pages of a website over time. Here is an example of the Google user interface available:

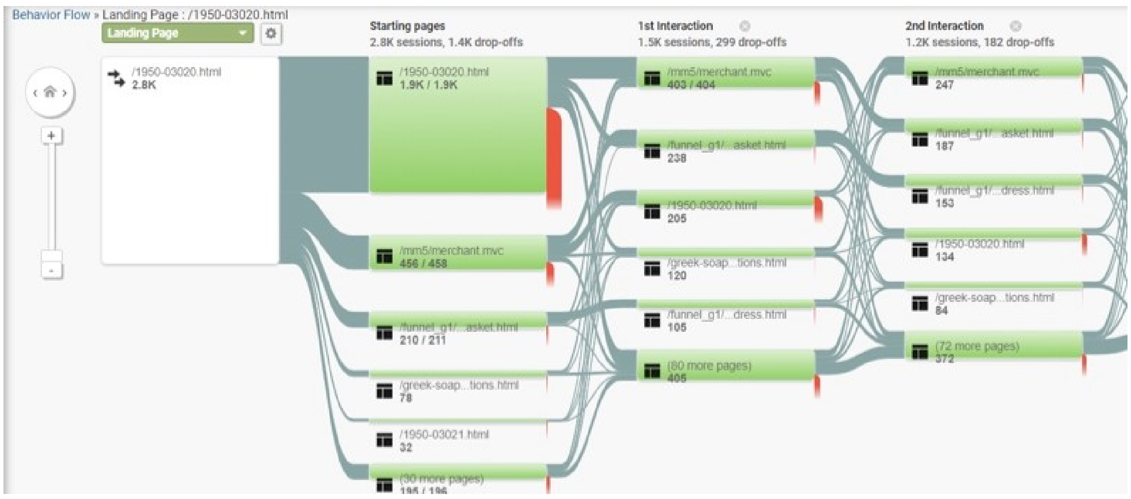

Behavior Flow » Landing Page : /1950-03020.html
Landing Page
/1950-03020.html 2.8K
Starting pages
2.8K sessions, 1.4K drop-offs
/1950-03020.html 1.9K / 1.9K
/mm5/merchant.mvc 456 / 458
/funnel_g1/...asket.html 210 / 211
/greek-soap...tions.html 78
/1950-03021.html 32
(30 more pages) 195 / 196
1st Interaction
1.5K sessions, 299 drop-offs
/mm5/merchant.mvc 403 / 404
/funnel_g1/...asket.html 238
/1950-03020.html 205
/greek-soap...tions.html 120
/funnel_g1/...dress.html 105
(80 more pages) 405
2nd Interaction
1.2K sessions, 182 drop-offs
/mm5/merchant.mvc 247
/funnel_g1/...asket.html 187
/funnel_g1/...dress.html 153
/1950-03020.html 134
/greek-soap...tions.html 84
(72 more pages) 372

Figure 1.3: Google Path Analysis[15]

Thanks to such a tool, it is possible to find the main actions that a user performs when visiting a website or to detect the presence of looping behavior. This temporal sequence can also allow to identify the next actions caused by a previous action[14].

However, the application is different from this tool, since it has to work not on the user navigation, but on other different events of any type. This means that it could create many new ways to analyze data since it is not limit to a specific case like user navigation. It is a more generic tool that can work on any temporal data, in other words data with a timestamp. The only limit is the structure of the raw data that has to respect few guidelines in order to be used with the application. So, the usage of any type of data with the application requires few preliminary steps that aggregates it in a new table structure and make it compatible with the application.

1.4 Thesis structure

Here is quick overview of the next chapters that helps to follow the thesis organization and so, the application development.

In the next chapter, **Development context**, is described the whole development process applied for each change on the code. So the different tools and protocols used and strictly followed to work on the project. There is also a description of the software methodologies available with a focus on the one utilized. Furthermore, there is a part which is useful to described the technologies that compose the application with some information on them.

In the chapter 3, **Project goal and overview**, there is the information to understand the goal of the project as well as its solution. It starts with a quick introduction of the current methods used to perform analysis and then it evolves with the new tool to develop. An overview is given as well in order to make the understanding easier in the next chapters.

The chapter 4, **Project development**, contains every significant detail that characterize the software development concerning different aspects of it. They are requirements gathering, prototyping, low-level architecture and testing. For each one of them, it is explained the approach used as well as a detail description which give a lot information of it.

The chapter 5, **Results**, is focused on the description of the final result achieved. It gives a large overview on the main features implemented and the different ways to use the application.

Finally, the chapter 6 is reserved for the **Conclusions** and for the next possible future changes to improve further the application as well.

Chapter 2

Development context

2.1 Development workflow

Before going into the development of the project, it is necessary to give an introduction to the internal development process, which is briefly described in the figure 2.1.

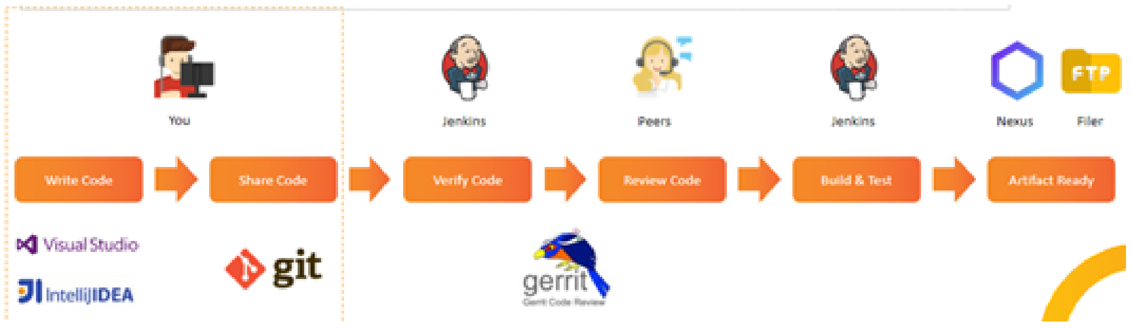
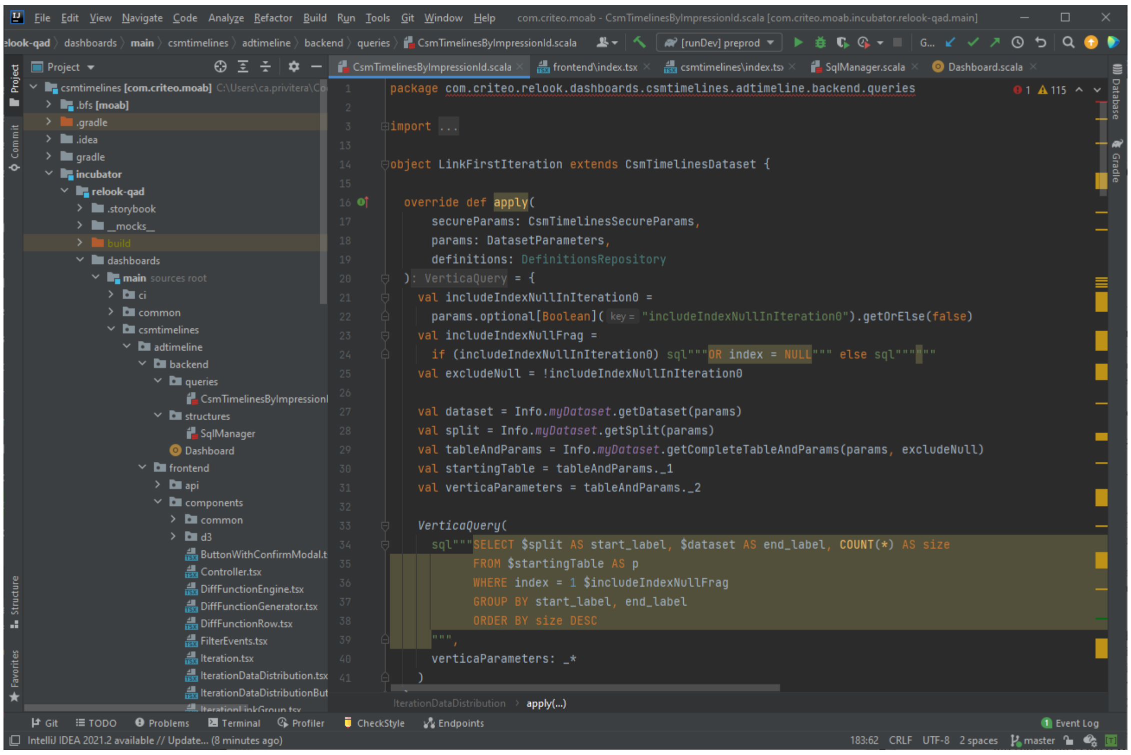


Figure 2.1: Criteo development workflow[7]

This process is quite long, but needed for each change made on the code. The reason for it is to make sure that every piece of code follows the best practice applied in the software development world. Indeed, it is not characterize only by writing and sharing code phase, but also by other phases that help to achieve the final result with less issues on the code.

If we want to analyze in detail the process, the first step, as mentioned before, is the task to write the code. This phase is usually done through any integrated development environment (IDE) that helps the developers to write codes or run tasks. Different of them are available in the market in form of free or premium. For instance, a light-weight IDE is Visual studio Code which supports different languages with support to a ton of plugins. In my case, I used the IDE called **Intellij**

IDEA, made by JetBrains[16], which offers powerful functions to the developer to automatize different aspects during the development. For instance, it imports automatically the modules needed while programming, so the developer does not need to look their position manually.



```scala
package com.criteo.relook.dashboards.csmtimelines.adtimeline.backend.queries

import ...

object LinkFirstIteration extends CsmTimelinesDataset {

  override def apply(
      secureParams: CsmTimelinesSecureParams,
      params: DatasetParameters,
      definitions: DefinitionsRepository
  ): VerticaQuery = {
    val includeIndexNullInIteration0 =
      params.optional[Boolean]( key = "includeIndexNullInIteration0").getOrElse(false)
    val includeIndexNullFrag =
      if (includeIndexNullInIteration0) sql"""OR index = NULL""" else sql""""""
    val excludeNull = !includeIndexNullInIteration0

    val dataset = Info.myDataset.getDataset(params)
    val split = Info.myDataset.getSplit(params)
    val tableAndParams = Info.myDataset.getCompleteTableAndParams(params, excludeNull)
    val startingTable = tableAndParams._1
    val verticaParameters = tableAndParams._2

    VerticaQuery(
      sql"""SELECT $split AS start_label, $dataset AS end_label, COUNT(*) AS size
            FROM $startingTable AS p
            WHERE index = 1 $includeIndexNullFrag
            GROUP BY start_label, end_label
            ORDER BY size DESC
      """,
      verticaParameters: _*
    )
```

Figure 2.2: Example of Intellij IDEA interface

It is compatible with different programming languages, in particular JavaScript and Scala which are the ones used in the project for front-end and back-end respectively. It also has a repository full of plugins useful for many tasks such as make the code prettier based on a specific configuration and so on. By default, it already has support for testing the code by showing useful information about it. It is also compatible with Gradle and its tasks which is the base of the project.

Next, the code must be shared with other developers, especially with those who are linked somehow with the project. The code is shared of course with git, one of the most common distributed version control system used nowadays. However, it is integrated with another tool called Gerrit, but it is more detailed in the next sections. Specifically, since I am working in a project based on Relook, the teams working on it are in charge to check any change made on it and make sure there is no something that could break other parts.

Once the code is pushed on the repository, besides the code review already slightly mentioned before, some automatic testing operations are started by Jenkins. This software totally enables the so called **CI/CD** which stands for Continuous Integration/Continuous Delivery. In other words, Jenkins runs many tasks on the code to perform some static syntax check, make sure the code has the correct format and runs some tests as well to known whether the change broke anything. Thanks to it, once the code is reviewed by another developer, then it is merge and integrated with the main branch as well as automatically built and delivered to pre or non-pre production environment. All these operations reduce at the minimum the effort required to the developers to make an update available externally.

2.2 Agile methodologies

Before writing code, it is necessary to establish the approach to use when it arises the opportunity to work on a new project. There are many different methodologies to develop a complex software[26]. Some of them are:

- Waterfall
- Spiral model
- Agile

Each one is characterized by specific characteristics that distinguish it from the others. The choice of one of them for the project was crucial per the final result to achieve.

A first possible method is the waterfall model. It is less iterative and flexible than the other approaches because breaks the project development into a linear sequence of steps, but it is considered one the most classic model that serve as basis for modern projects. Each step depends from the previous one, so it is not possible to run the single steps in parallel. For this reason, it is used in the manufacturing industries given that it is high expensive performing changes in the middle of development[1].

Another one is the incremental model. It consists in develop and test the product in an incremental way until it satisfies all its requirements, both functional and non-functional. In order to do that, the product must be composed in small components which can be built separately. If we want to put it in another way, it is the waterfall model made incremental[1].

V model is very similar to the waterfall, but follow a different order on the steps. Instead performing testing after each development phase, it runs all development steps in order and then back through all testing stages to verify and validate the project[1].

Given that the development required a certain level of flexibility as concerning changes and the addition of new features, agile methodology is the current one more suitable and actually used. This approach lends itself very well to teamwork, as it consists of discovering requirements during development. In fact, there is continuous collaboration, with the aim of being flexible when a change is required. Thus, the planning is not strict and tries to keep the product evolvable over time. In this way, there is a direct contact with the customer through faster deliveries and early feedback[26].

There are several variants of agile. The ones used are for example **Kanban** and **Scrum**. The latter is the approach used for this project, and it takes the form of having a subset of tasks that need to be completed in a short period of time, the **Sprint**. The software currently used for scheduling is called Jira[17]. Here is a diagram of it (figure 2.3).

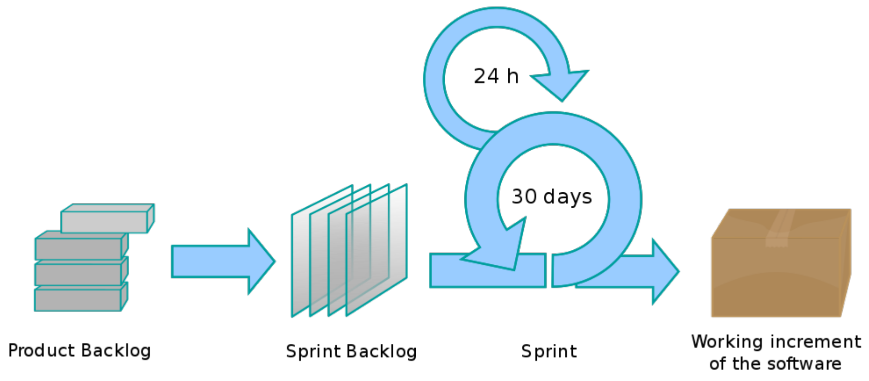


Figure 2.3: Scrum methodology

If we look closer, the **Product Backlog** is a set of tasks that need to get done. Instead, the **Sprint Backlog** is a subset of them with the goal of getting them done in about 14 days, the typical length of a Sprint in my case. Each day, a small **stand up meeting**, about 15 minutes, then needs to be scheduled to assess the progress of these tasks for all team members. After a Sprint, there is a delivery of the product along with a **Sprint Retrospective meeting** to note what was good and what was not.

2.3 MOAB

At Criteo, the development of any project between teams can be defined as "**trunk based + mono repository**". Git is configured so that a project is defined on a single branch called **trunk** (master). This in turn is composed of smaller branches that are only merged after code review and a testing phase run through Jenkins. Also, git uses the rebase strategy by default to maintain a clear history of all commits.

However, some projects need the builds of others to run. For this reason, trunks can be grouped as a single **virtual trunk** that is also the representation of a workspace in a local project. This is where **MOAB** comes out, which stands for "Mother Of All Builds." It is a build of a virtual trunk by compiling all repositories that depend on each other. This is usually done hourly and creates a MOAB ID, which is a tag on the virtual commit.

All this complex structure is managed locally by Gradle, which allows to partially automate this process and its management. In fact, it sets up the project and all its dependencies after a build command is executed. If there are any errors due to missing local software during the build, then it reports this to the user.

2.4 Code-review

Code quality is very important, especially when working on large complex projects. After all, maintaining high quality means that other people can understand the code more easily. This not only helps to make quick changes and add new features, but also allows to identify bugs before it is too late.

One way to maintain high quality is through the code review process applied here as well. Although Git is used for every project to share code changes, it lacks a code review feature. Therefore, this process is done at Criteo with the help of **Gerrit** tool[13]. It is a useful tool that works as another layer for Git (figure 2.4).

In fact, every change made will not be directly merge with the main base code, but it will be in a parallel repository that will not go to interfere with the original one.

Thanks to it, you are able to push your code to it. Another authorized developer must review and validate your code by assigning it a score. In the meantime, it will go through a testing phase consisting of general tests on syntax, type checking, build and best practice configuration made through Jenkins as mentioned at the beginning of this chapter. Only when these two processes

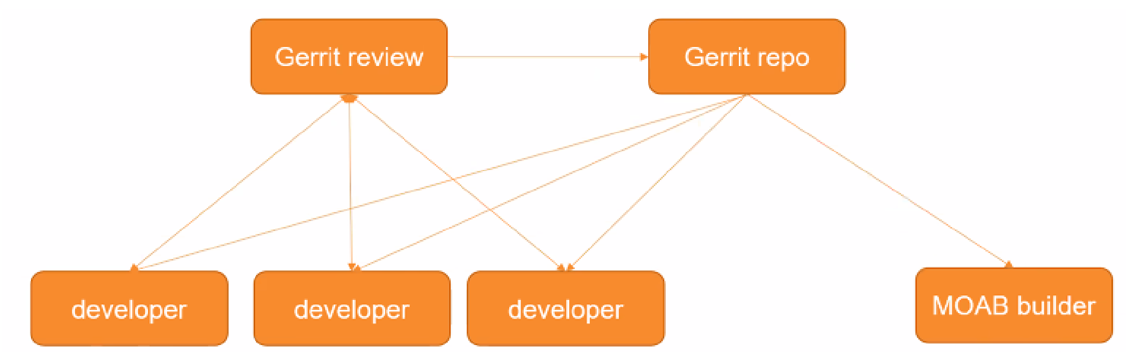


Figure 2.4: Gerrit structure

are completed with the maximum score, the code is merged into the real repository.

For instance, the data export was one of the task with the longest code review since it has a big impact with the others components of the company because it can break them. Any change on the code, small or bigger, had to be described clearly and reasoned. If there is a better or alternative way to achieve the result, then it must be done. The same can be said for the direct development of the application. The code review process is one of the most crucial aspects in a big company with many employees working inside, and it must be followed to keep the best practice in every project.

2.5 Technologies

2.5.1 DBMS

The timeline has to get data somehow to visualize them. They can be obtained through many internal DBMS at Criteo, each used for different purposes. The main ones are:

- Hive
- Presto
- Vertica

Hive is used as the main DBMS to store any kind of data ranging from user information to browser performance. However, this system is quite slow when it comes to queries. This is something that is not desirable when using the timeline, especially because some operations requires a certain complexity during the execution.

The alternative is to use **Vertica**, which is one of the fastest systems for executing queries. Thanks to this system, it is possible to get all the required results before any other DBMS and, in the case of the timeline, allows performing many more analyses in a time interval compared to the others. The reasons to opt for this system are various. Indeed, it stores data differently. If we consider the tradition RDBMS systems, they store data by row, but Vertica stores them by column and split them in different nodes.

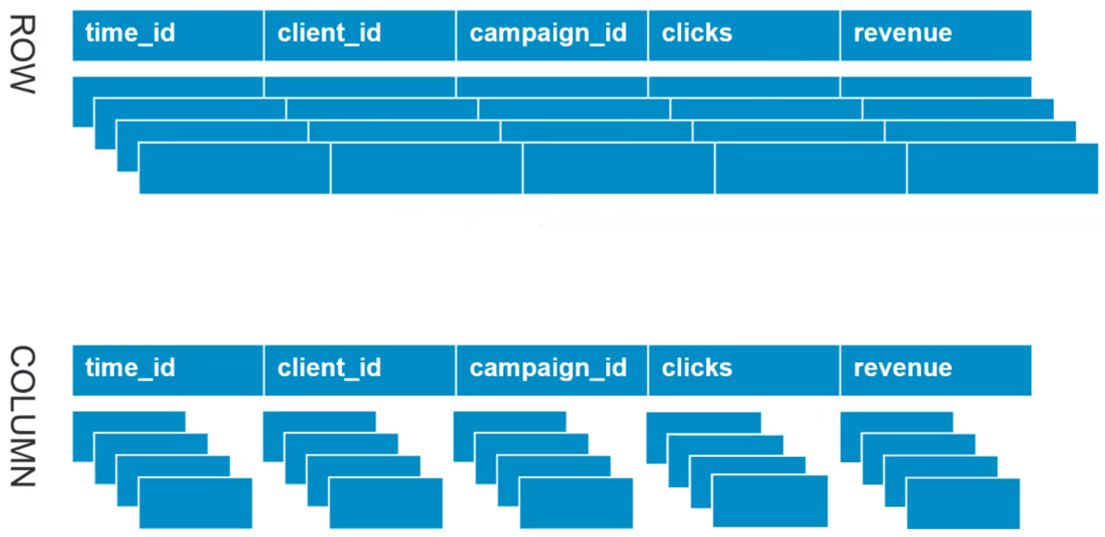


Figure 2.5: DBMS storage structures

In other words, this approach allows aggregating data much faster given that we can immediately have access to the whole column. This also allows to drastically reduce the number of I/O operations because the system does not need to read the other columns.

Furthermore, Vertica is also able to compress data with a powerful algorithm, especially when some values are repetitive by reducing the space up to 90% of original data.

Its architecture is a distributed MPP, which means it aims to paralyze query execution as much as possible through different nodes available in the cluster. Thanks to it, Vertica is compatible with all the main distributed platform such as Amazon Web Services and Azure. We can also configure some resource pools and projections which are basically indexes to further improve the performance[25].

All these features are taken into account for the project development and actually used for the data processing. In fact, the events to use for the application are recorded every time, 24 hours a day and 7 days a week. This causes the necessity to have enough space in the database storage to make sure it never runs out. However, the space for Vertica is not so large since it is used for specific applications like the one to develop. So, it was need to optimize data by removing the columns not actually necessary or useful for the analysis, aggregating the events and all the other suitable information, but also take advantage of Vertica algorithms to compress repeated values such as row identifiers.

2.5.2 Front-end

Given the complexity of Criteo's infrastructure, the project is powered by an internal framework called **Relook**. It contains many tools to build dashboards from scratch as web platform and, thanks to it, the project is ready to communicate with the back-end and be deployed as an instance on **Marathon**, the container orchestration system currently in use.
It is made with the goal to provide tools to make our own dashboards of any kind. The main ones already available for the usage are the following environments:

- RM: Demand side platform dashboards
- RM Internal: Internal dashboards
- RM SSP: Supply side platform dashboards

So, the idea is to use this framework as starting point for the application to make a new dashboard that could be considered as a new version 2.0. In fact, it is not just make a new environment where working on, but it consists in adding new features that give the opportunity to create new powerful tools such as the timeline viewer. These changes are added as a new layer on the framework that allows to break its limits which are described later on. Relook contains both front-end and back-end libraries. The front-end relies on the framework React used for building web apps based on the language **TypeScript**, now extended to be used as web components. These are custom HTML that can be used like any other HTML tag[19].

TypeScript is used to make JavaScript typed, so there is a file called *tsconfig.json* that contains various options, files to include and check. Thanks to it, it is possible to reduce the probability of errors due to different types of data. To identify files in TypeScript, some files have the extension *.ts* and not *.js*. Others are in *.tsx* because they are in TypeScript with **JSX**. The latter is a syntax extension to JavaScript and allows us to write HTML elements in JavaScript as React elements.

Using TypeScript as base language on the front-end side brings significant benefits[23]. Since its nature of being typed, it allows the **static type checking**. This is an important feature, especially for big and complex projects. It allows to catch errors due to data type already before running the application. However, the developer is not force to use it since it is optional by not expressing the data type or by assigning the type *any* to every variable or function parameter.

Figure 2.6: Libraries and languages used for the project

Another non-trivial benefit is the browser compatibility. The compiler already allows to use the new features available in the language update ES6, also known as ECMAScript 2015, and beyond and then will convert it in an older version, such as ES5. In this way, there are no worries about the compatibility since it is the compiler itself that allows the developer to use these functions to make them backwards compatible. So, older browsers would be able to execute the same application even if it does not support the new version of JS.

Furthermore, Intellij IDEA, the IDE currently used, has support for its static type checking, so it possible to take advantage of its code assistant. A lot of active hints are provided by IntelliSense as the development is on-going. For example, the editor can suggest the methods available and their expected parameters, a useful feature to speed up the development.

TypeScript is the base language for the library React running in the application [20]. This library is maintained by Facebook and its community that make it updated with features still until nowadays. It is heavily used inside the project in a way to reduce the lines of code required and make usage of the reusable code. It is composed by some aspects that characterize the library from the others available in the market and make you opt for it.

React is made to help the developer to build user interface and increase the feeling of the user experience. So, it allows to apply a top down approach which helps in case of complex interfaces. In technical terms, it allows to subdivide the user interface in simpler and reusable elements. They are encapsulated in so-called components that are characterized by three main parts:

- a behaviour that defines its internal mechanism
- a graphical design that can be defined through simple JS code or by HTML through JSX
- a state that represents the current DOM and internal representation

All these three elements together allow to make also very complex interfaces since each component can be debugged in isolation. In this way, the testing phase can benefit from these aspects by requiring less operations to do for testing a component.

What makes this library very helpful is its system of automatic update. This makes React perfect for data visualization and in particular for the model-view-controller (MVC) pattern. Thanks to it, the developer has to just worry about keeping data updated. The rest is done automatically by React which is in charge to update the visualization on basis of the behaviour defined in the respective component.

Another feature that it is worth mentioning is the virtual DOM. React makes use of it for its components. The idea behind it is that virtual DOM is the virtual UI representation kept into memory and then synchronized with the real DOM of the browser. What it is interesting is that any update on a component, and so on the virtual DOM, does not trigger an update on the whole real DOM, but only on the elements where the update happened. This is drastically different from the standard approach without virtual DOM, since any change, small or big, causes the update of the whole DOM.

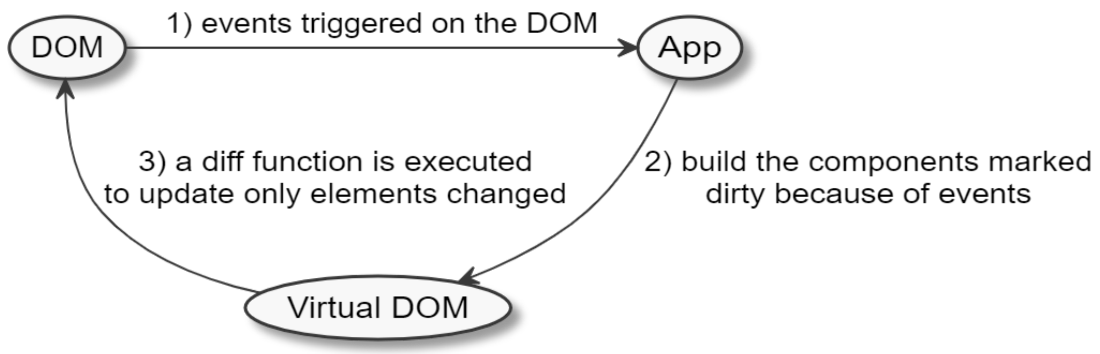

DOM
1) events triggered on the DOM
App
2) build the components marked dirty because of events
Virtual DOM
3) a diff function is executed to update only elements changed

Figure 2.7: Virtual DOM update process

The figure 2.7 shows the internal process applied. The actual DOM is the one in direct contact with the user interactions and sends every event triggered to the application. Then, the components affected by them are marked as dirty in the virtual DOM tree. Consequently, a comparison algorithm is run to compare the real DOM with the virtual one in order to make the changes needed.

The main visualization of the timeline and other elements is done with **D3JS**. It is a framework for web development that is able to describe the graphical aspect inside an SVG tag element, a container for displaying scalable vector graphics. There are many features that allow you to create any kind of visualization, like a timeline in this case. Nevertheless, it's not that easy to master and requires a lot of learning to use it correctly and efficiently. Thinking this kind of user interface with just HTML/CSS would have been much harder especially to create a flow.

There are other frameworks that are capable of creating different types of charts that can be used to create the timeline. In this case, it would be a Sankey diagram. For instance, Relook uses Highcharts for most charts, but it was not the right fit here. Indeed, these frameworks work at a very high level, so while it might speed up the creation of an initial prototype, it would not then be easy to add certain features that are not already available and suggested by the framework itself, since there is no complete control over it. This is not the case with D3JS, as it is at a lower level and allows much more customization. In fact, there are some basic powerful features to get the rendering you want, but also some higher level features like charts, maps or networks.

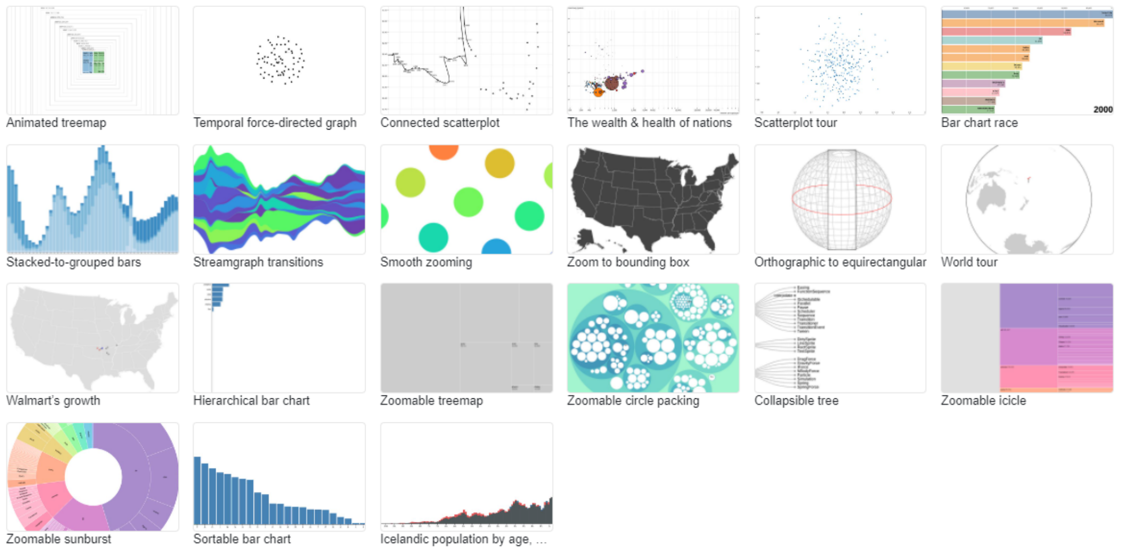


Figure 2.8: D3JS visualization examples [11]

However, D3JS working with React is a less common task than others. Both are trying to get access to DOM, which can conflict if not used correctly. In fact, React has its own way of managing DOM through the **Virtual DOM**. At the same time, D3JS wants total control over DOM. Fortunately, React allows this through a special clause called **Ref**[21]. Thanks to it, D3JS can access it to visualize its elements, but you have to be very careful when implementing it, because using it incorrectly could corrupt the operation during execution.

2.5.3 Back-end

As for the back-end, it is written in **Scala** and connects to the front-end locally as a proxy. By configuring a few environment variables like username and password to access to the DBMS, the project is ready to communicate with Vertica and run queries on it. A summary of Relook architecture is shown in the figure 2.9.

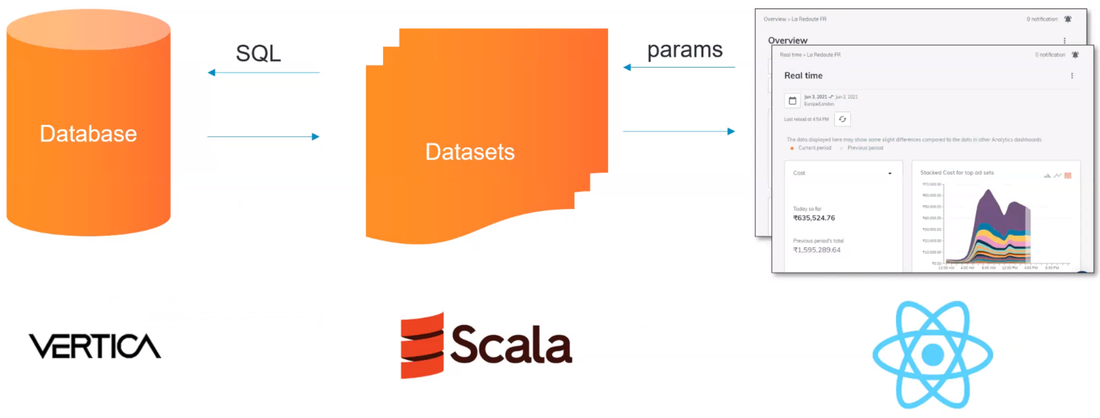


Figure 2.9: Relook architecture

As it is possible to see, the architecture shows that Relook is composed by there parts.

First, there is the database which Vertica and contains all the analytics data. So, data is pushed here through a data export process from the other main databases such as Hive.

Then, we have datasets which are the result obtained through parameterized queries in SQL. This is done on the back-end side which is in charge to receive the parameters from the dashboard to generate a query and, in turn, send it to Vertica in order to get the respectively computed data which are the so-called **datasets**. They are really the heart of the dashboard and the whole complexity of the queries relies on the datasets. Furthermore, the back-end is the gate for any endpoint for the external requests that come from the front-end or any other application. So, here is the point where the endpoints have to be created. Also, there is an implementation of security breach to make sure that any sensitive operation is not executed if the user is not authorized.

Finally, there is the front-end where the dashboards are displayed, made in React already described before. Here it is where the dashboard sends the parameters, such as filters for the data we want to show, and then it receives the datasets to consume and generate the data visualization. Each one of them, theoretically, is used for one element shown in the dashboard such as a table or a line chart. However, the same dataset can be used to show multiple different elements if its data structure allows.

A dashboard made in Relook can contain some filters. They can be of any type such as dates, type of currency, and so on. Then, they are sent to the back-end to recompute data and a new dashboard is visualized.

Chapter 3

Project goal and overview

3.1 How data is currently managed

Thanks to the data it is possible to improve any machine learning model and the results obtained every day. In this case, the data is used to improve the ad system of the Criteo engine, with the aim of delivering the most appropriate ads to the user viewing a website. This task is not easy to accomplish and, for this reason, machine learning is applied here as well. Thanks to that, the recommendation system can provide an ad with the relevant products based on the user preferences. The figure 3.1 shows the commonly used approach.

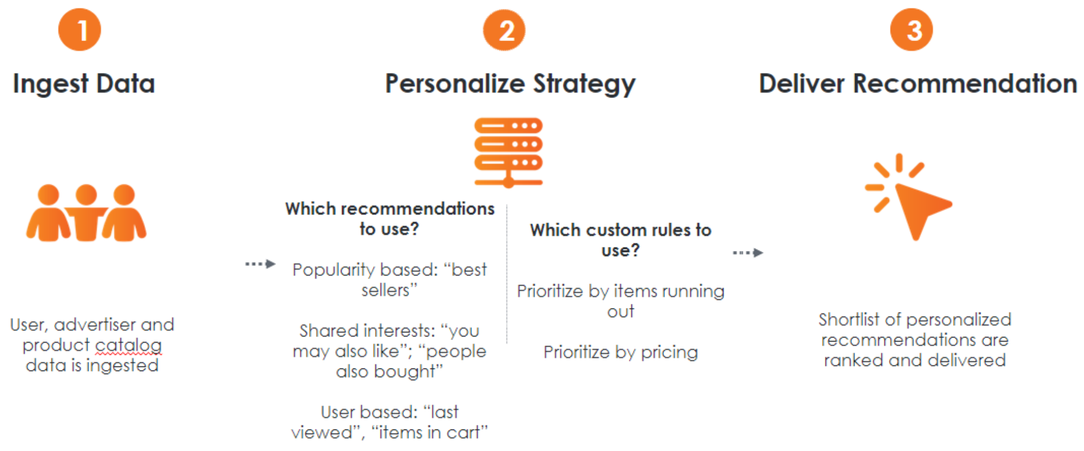

1
Ingest Data
User, advertiser and product catalog data is ingested
2
Personalize Strategy
Which recommendations to use?
Popularity based: "best sellers"
Shared interests: "you may also like"; "people also bought"
User based: "last viewed", "items in cart"
Which custom rules to use?
Prioritize by items running out
Prioritize by pricing
3
Deliver Recommendation
Shortlist of personalized recommendations are ranked and delivered

Figure 3.1: Criteo AI Engine[5]

As seen in figure 3.1, the recommendation system has several strategies available to deliver an ad. For example, they can provide products by popularity or the most recently viewed products by the user. This action needs to be performed in a few *ms*. Behind this, the machine learning model needs to understand the user's preferences by using data that exists in the database. To do this, the engine needs to track the user through cookies that are stored in the browser. By using them, the system can better understand what type of user is on the website. However, the system is not perfect and can sometimes give false readings due to the complexity of the process and the many variables involved. Furthermore, it takes time to be trained and reach a state that is quite reliable. So the model needs to be improved and properly configured every day, a task that is not easy to accomplish.

The Analytics team works hard to analyze the existing data to improve the ad system. For instance, they made great progress by providing a huge amount of information, such as heat maps of clicks performed by users on ads[8]. This was possible thanks to the different tools they use, such as Tableau which allows to show data in many ways through a set of charts available in the application. Thanks to this, they can optimize the overall ad design by providing other options for ad configuration.

3.2 Problematic

Although the team working on Big Data uses powerful tools to perform analysis, they would like to have a tool to analyze time-based data through a timeline, something that is not ease to achieve with the current software, and so they need a dedicated application just for that.

However, they do not want to show the user navigation, but they want to visualize the following collected events:

- banner pointer events
- banner viewability

Concerning the **banner viewability events**, they are more technical and server-side. So, in order to make the explanation more clear, I just focus on the other type.
The first type of events called **banner pointer events** are events collected by the service CSM, already mentioned in section 1.1.2. In particular, when a user goes to a publisher website, CSM injects some JS code on the banner shown. These scripts have the task to listen the user interactions on the banner and send the collected events, in combination with their timestamps, to an application that will log the raw data in the Criteo databases.

These events range from the simplest ones, such as *TOUCH_START* or *CONTAINER_SCROLL*, to the most complex ones, such as *DISCARD_PRIVACY* or

SURVEY_ANSWER_CLICK. An example is shown in the following image which could not perfectly reflect the actual behaviour of a banner, but it is good way to introduce the goal of the internship.

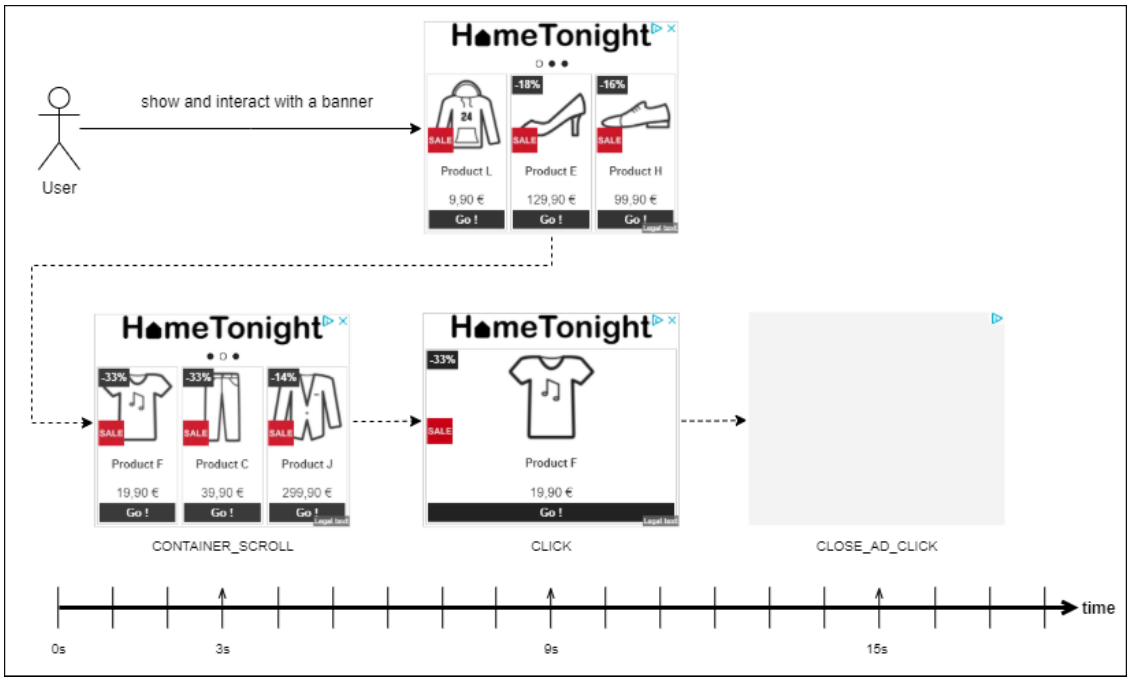


Figure 3.2: Example of banner pointer events collected by CSM

In this example, the user is interacting with the banner, and for this reason CSM collected some events over time. As it is possible to see, the banner changed state according to *which* and *when* the events were triggered. Then, all these events together compose a sequence, or in other words the interaction history of a user and it can be shown through a timeline.

However, this is just a history for a user over many others. In fact, not any other user would interact with a banner at the same way. They could trigger other different events and/or in other moments, hence generate different timelines. So, the goal is to create an application able to gather all these histories and show them all in once through a single timeline. In this way, we can get a summary of all sequences obtained. Thanks to it, it could be possible to visualize the general trend of users while interacting with a banner, get information about the most repeated events and much more.

3.3 Solution

In order to explain the internal mechanism of the timeline to create, let's suppose to have two users, *user1* and *user2*, and they trigger some basic events while interacting with a banner.

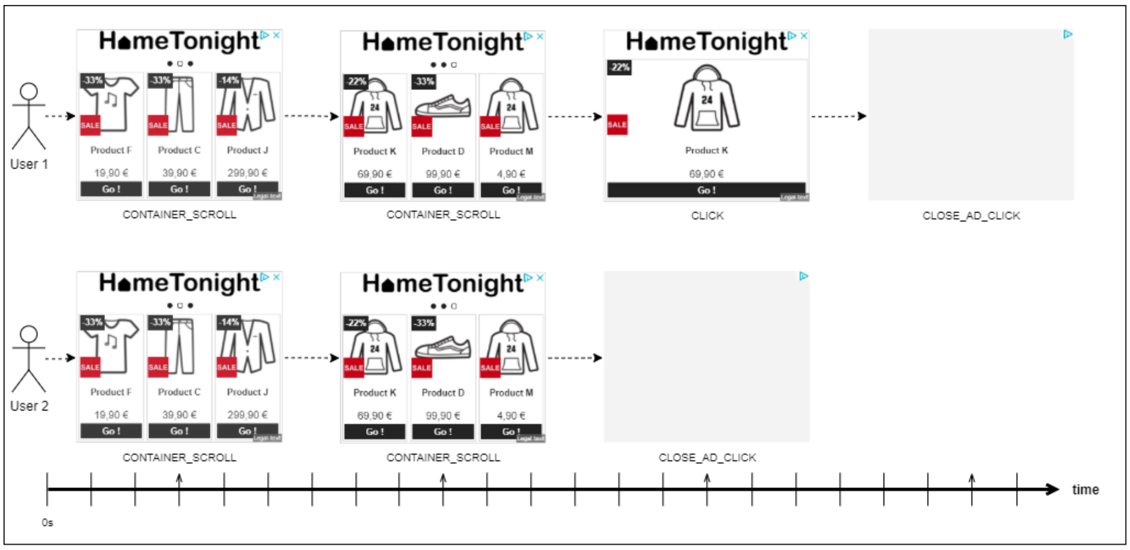


Figure 3.3: example two user interaction histories

As it is possible to notice, these two users have their histories composed by events collected through CSM. Therefore, the final goal of the internship is to make a timeline able to merge all these histories of interactions to get a summary of what happened. This is achievable only by recreating these histories so that the application has immediate access to all of them. Here is the expected result for the given example:

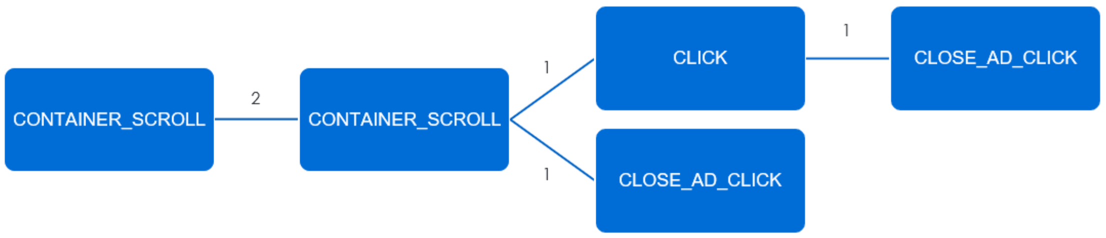


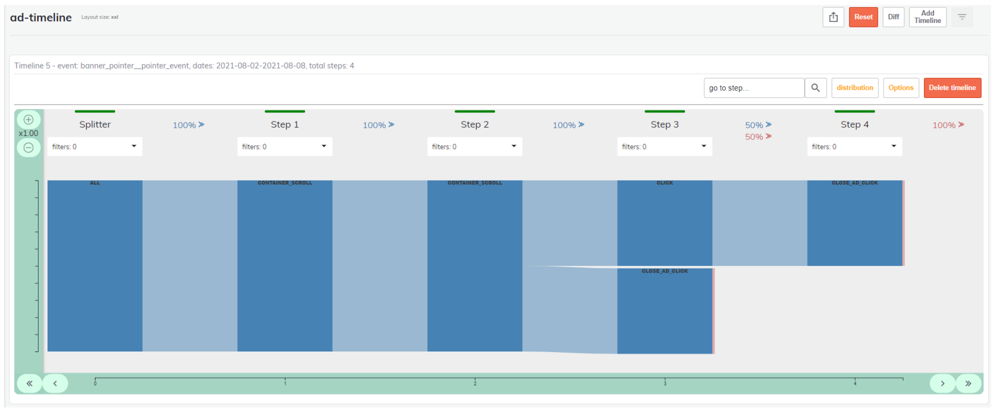
Figure 3.4: final timeline of user interaction histories

Indeed, we can see at the beginning we have the *CONTAINER_SCROLL* event for all users, then another one, and then they trigger different events, *CLICK* and *CLOSE_AD_CLICK*.

However, this is just an example to make clear the goal of the project. Furthermore, it is not just to merge histories, but it is much more to improve the effectiveness of analysis and give an incredible wide set of options.

3.4 Overview

Before talking about data setup and architecture, it is better to show a screenshot of the application built in order to have a clearer idea of the low level structure and how the internal logic works. In particular, this example exactly show the result of figure 3.4:



Timeline 5 - event: banner_pointer__pointer_event, dates: 2021-08-02-2021-08-08, total steps: 4

Figure 3.5: final timeline interface

Before going into details, it is needed to explain some terminologies used in the final version. Let's consider banner pointer events to show as reference, namely events triggered when a user interacts with the banners.

As it is possible to see in the figure 3.5, a timeline is composed by different columns, one called *splitter* while the others *steps*.

The first one does not describe any event, but shows how the initial data source is split a priori. In this example, there is no split and the whole data is gather under the name *ALL*, but it can be split in other way, by *host platform* or by *date* for instance.

All steps that go from 1 to N show the events that happened in the position X of the user history. For example, all events in step 1 are the first events triggered by the users, while events in step 2 are the second events triggered and so on. The single event has a certain height which indicates the number of users with respect to the total.

Finally, all these steps are connected by some *links*. Each one goes from an event in step X-1 to another one in step X. These links show how many users moved from an event to another one. It is important to highlight that the links could overlap each other to connect two steps. This is due to the order of events inside a step which are sorted in this case by height in decreasing order.

There are other visual elements in the web page such as controls to scroll a timeline, zoom, search a specific step, go to the first or last steps and so on. There are also buttons to reach more advanced features in order to change the analysis outcomes. Indeed, each timeline has a button called *options* to change its configuration. Here we can change the type of events we want to analyze, or add some filters to the data. It also contains user interface options and more.

Another feature is the data distribution. Through this button, a modal is opened by showing information about the data distribution of the current timeline. Here there are two charts, one shows the original data distribution, while the other shows the modified data after applying the timeline filters.

A button with the share icon is at the top of the navbar. Through that button we can share the view or import another one. Then it is also possible to compare two timelines through the Diff function, this one available at the top of the navbar as well. However, They will be examined in a specific section.

Although all these features, the application required some prototypes before achieving this state. Here is, one of the first prototype:

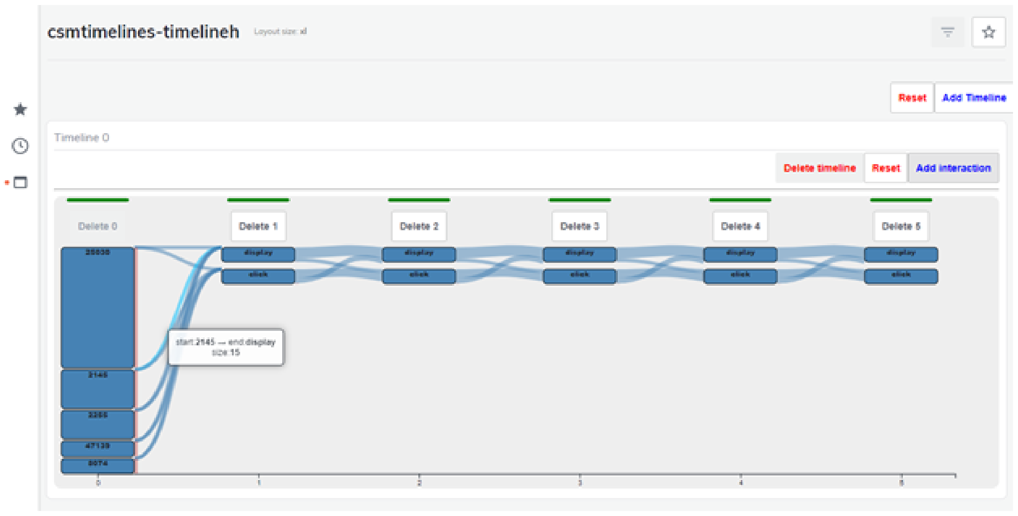


Figure 3.6: Timeline prototype

If we compare it with the first prototype version, shown in the figure 3.6, it is immediately visible that many features have been added such as some controls, Diff function, data distribution and several options. Besides that, we also have a visible UI improvement and some optimizations in the logic. The first version was also a little bit cumbersome to use, while the new one automate basic operations without the user intervention.

Chapter 4

Application development

4.1 Development phases

After several months of work, the timeline is able to analyze different types of **events** that are actually used by the company, and in particular by the analytics team. Indeed, the development took around 6 month to achieve the expected application. Here is a Gantt chart of all main steps made to reach the final result of the development:

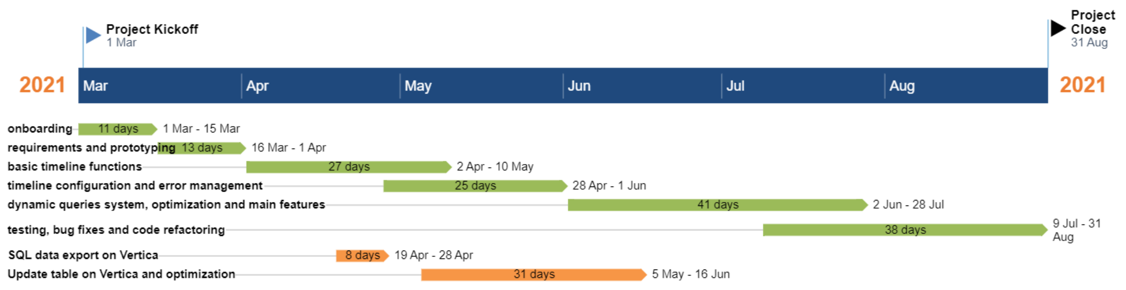


Figure 4.1: Gantt project chart

The process of development was quite linear, except for some tasks that were needed to be done in parallel. Indeed, the tasks needed to directly develop the timeline are shown in green, instead, the tasks in orange to prepare data and use them with the timeline.

If we would like to go step by step and detail the green tasks, first there was an on-boarding phase, required to know more about the company and get a better knowledge about the tools used there. After that, it starts the real process of development which is composed by a phase of requirements and prototyping. Next there is a coding phase by starting adding core features up to additional functions to improve the user experience. Finally, there it is a final task which is focused on testing and bug fix, important to deliver the final product stable and solid as much as possible.

It was also needed, in a certain point, to work in parallel on the data preparation. In fact, because of technical reasons such as speed execution and data structures, it was needed to work on the original data source to aggregate and export them in Vertica with a different structure.

Given the complexity of data and the large amount of tests for any operation, the process to export data took a while. Specifically, this is due to security, memory and optimization. In fact, if on the one hand the development of the timeline was made in a safe area of the project which cannot cause any particular damage to the other components of the company, on the other hand the data export could have a big impact on the storage and may totally fill it, or it could conflict with other export jobs and make them crash.

Furthermore, deciding a wrong data structure to export can cause some delays because the new table must be reset and update the data export code with subsequently new tests and general checks.

4.2 Requirements gathering

The development started with different prototypes and some ideas. It was important to understand how Criteo collects data and what its meaning is. Only then it was possible to design the timeline and know how it should work. However, it is not enough to start working on it as the people directly interested are the Analytics team. Therefore, it was necessary to conduct a requirements gathering phase with them by scheduling a meeting.

In general, a tool used to analyze data must be able to:

- show information
- share findings with others
- allow different configurations
- save the view
- highlight aspects of the analysis
- allow a comparison between two analyses to highlight their differences

A prototype was needed to show the basic aspect and the features to build. In this context, it may be useful to show a mock-up of the home page. From the figure 4.2, there is a mock-up user interface made that is very similar to the final product.

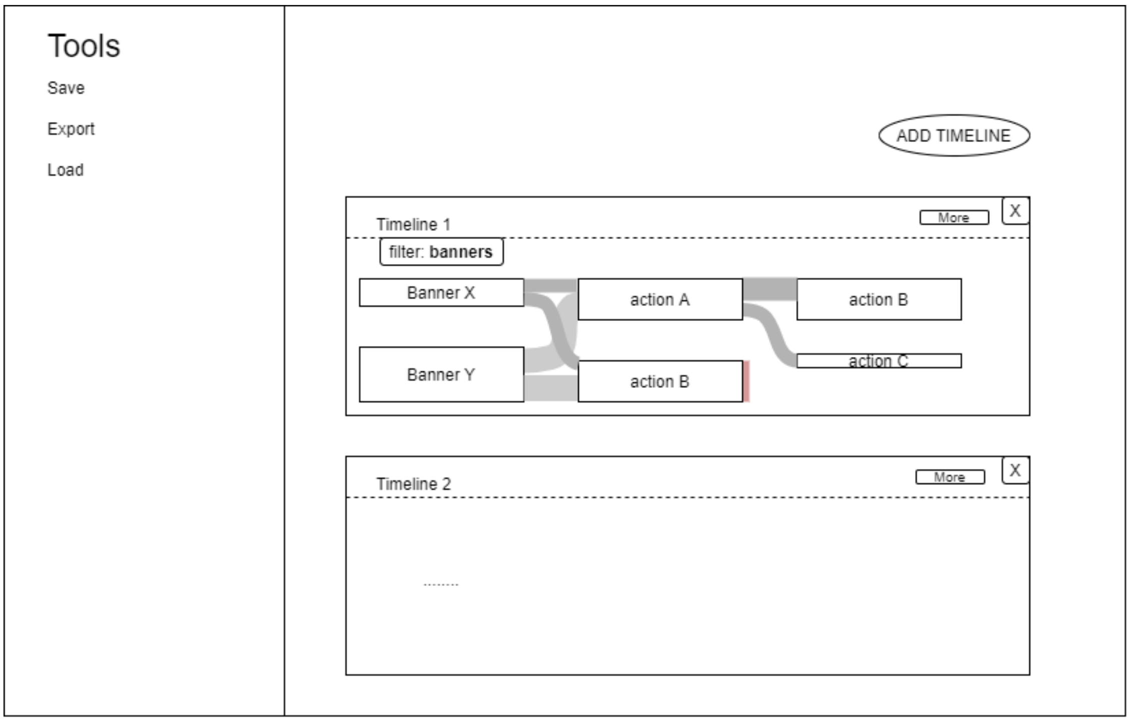


Figure 4.2: Timeline web page

The mock-up does not necessary need to be exactly the same of the final product, but it helps a lot to develop the general structure of the application. The developer, in this way, has an idea on where place the functions needed and how the user interface should look like. In fact, the mock-up must be associated to a set of requirements to describe the final product.

The best way to perform a deep analysis is to add different features with the goal to achieve the final result we would like to. These features belong to the functional requirements, or in other words a set of features that the application has to support. These functions range from the basic ones to the most complex ones and it can be a sort of dependency to each other on basis whether a function can be present only if another function is implemented.

These functional requirements are listed in the table 4.1 through two columns, the first one assigns an id to the feature which helps to define uniquely, while the second one is a quick description of the feature itself.

ID	Description
FR1	add/remove timelines from the view
FR2	any timeline can be configured to show a type of event
FR3	any timeline can be scrolled and zoomed manually
FR4	any step of any timeline can be filtered by event
FR5	any timeline can have a global event filter that works on every its step
FR6	any type of events obtained by the timeline can be filter by date
FR7	any timeline can have the auto-zooming enabled or disabled
FR8	any timeline can show the small events bigger, but the scale is lost
FR9	any timeline can ignore one or more events and replace them with IGNORED_EVENTS
FR10	any timeline can select the initial splitter of the dataset or not split at all
FR11	the ignored events can be excluded from the workflow and not shown at all
FR12	duplicated events can be removed from the original sequence before showing a timeline
FR13	any timeline can show the first and the final events of all sequences
FR14	any timeline can be filtered by other types defined for the specific type of events under analysis
FR15	any timeline can be configured to obtain sequence with a minimum and maximum length after applying every option chosen
FR16	any timeline can be configured to obtain sequence with a minimum and maximum length before applying the options ignore/exclude events and collapse events
FR17	any timeline can include data without history in the splitter
FR18	show data distribution with only the filters
FR19	show data distribution with every options chosen
FR20	save current view in local storage at any change that requires an update on the data on any timeline configuration
FR21	export view through a JSON
FR22	import view through a JSON
FR23	run a diff function to compare two timelines step by step connection through the percentage variation

Table 4.1: Functional requirements

Each function has a specific goal which covers a specific part of the analysis concept. Not only these features must be implemented, but they also need to get along with other non-functional requirements listed here:

ID	Name	Description
NFR1	Compatibility	application must run on any device with an updated browser, so at least Desktop PC and Laptop
NFR2	Performance	each step must be run in a few seconds
NFR3	Usability	application must be usable from any user in less than 15 minutes of explanation
NFR4	Fault tolerance	everything has to work even in case of error and signal the user through notification
NFR5	Extensibility	it is possible to add new components without any particular problems
NFR6	Availability	application is not accessible only when there is no internet connection, in all the other cases it must work
NFR7	Flexibility	application must be developed in order to guarantee a fast changing in case a functional requirement changes

Table 4.2: Nonfunctional requirements

The application should be accessible on any desktop PC or laptop, regardless of the operating system. For this reason, the application must run as a web application to reduce issues related to cost, compatibility, and maintenance.

Proposing software that can run on almost any device means that if one device breaks, another can be used to run it. It needs to be fast to run analysis smoothly while maintaining availability and fault tolerance. It also needs to be easy to extend and allow quick changes on existing functionality.

4.3 Actors

Once defined and gathered the functional and non-functional requirements, it is important to defines other aspect of the project before starting its development. These aspects has to deal with the entities involved by the application, their interfaces and the context of the whole system. In particular, the actors and their description are listed as follow in order to have an idea of the main entity who have to interact with the application. The table 4.3 defines the actors of the system.

Actors	**Description**
Criteo employee	He/She uses the application to perform some analysis on data available
Vertica	It is queried to get data when a user uses the application and show the dashboard

Table 4.3: Actors of the application

Once the actors are defined, here is the context diagram which is an alternative representation of interaction between actors and system.

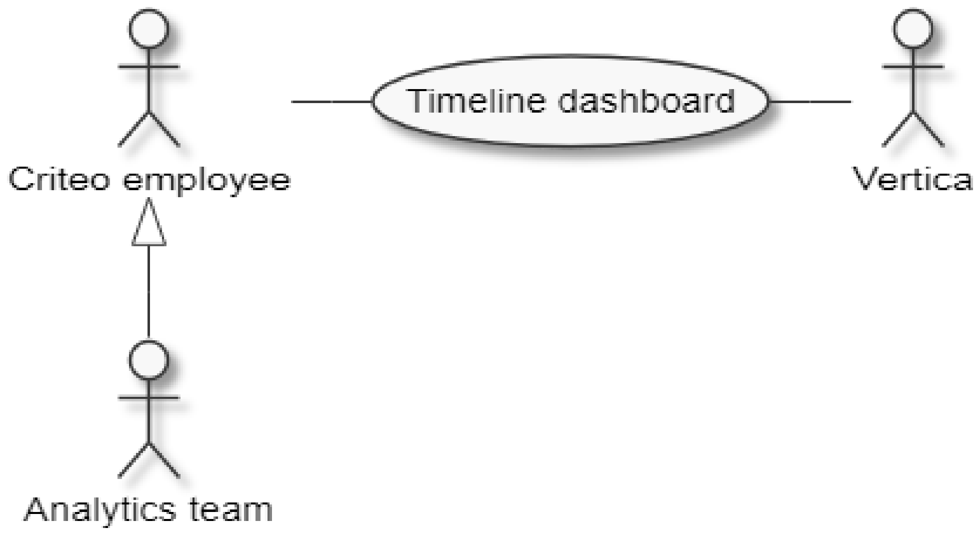


Figure 4.3: Context diagram

An actor is not necessary a real person/user who can interact with the software, but it can also be a system which has to be present in order to use the application. So, if the application has to interact with something or someone else, then it is considered an entity.

Indeed, the system has two main actors. The first one is the Criteo employee and not any user. This is because the application is just available for internal use at the moment. There the idea to make it available for the standard user, but it is not the current goal of the project. The employee gets the full access to the application to perform any kind of analyze available. Here he/she can use select the events and choose every feature added into the application.

Next, the other main actor is not a user, but a system. Specifically, it is Vertica, the DBMS used to get the data source to analyze. It may be consider internal to the system of the project, but, since the application is a tool for analysis of any source, we can consider the DBMS external, provided that it respects the data tool structure. Through this system, the application is in charge to query it as soon as the user choose the timeline configuration.

4.4 Interfaces

Next, after defining the actors of the system, it is necessary to establish how these actors can interact with the application.

These means are described by two different types:

- logical interface
- physical interface

As their name says, the logical interface describes how the entity can interact logically with the application. So, any technology or interface is considered. Instead, the physical interface describe how this interaction works physically, so any hardware element that is involved.

The table 4.4 is describing the interfaces needed to make the application usable by the entities described before.

Actors	**Logic interface**	**Physical interface**
Criteo Employee	GUI and company VPN	Keyboard, mouse, monitor
Vertica	API	Network

Table 4.4: Interfaces

The first entity, the Criteo employee, can use the application only through the GUI, the graphical user interface of the web page, and the internal VPN of the company. In fact, since the project is just for internal use, the website is available only by through the VPN. Thanks to it, it is not needed any type of authentication since the user is already authenticated by the VPN itself. Instead, the physical interface is any hardware that allows to interact with the web page.

The second entity, Vertica, can be contacted by the rest API when the application has to query it in order to get the data needed. This is possible by using the network because the DBMS has its entry point which, in turn, manages its servers and nodes to paralyze the query execution.

4.5 Use cases

A further diagram that may help to capture the requirements of the project and make clear its behavior is the use-case diagram model. It consists in representing the possible interactions between actors and system during the execution. However, these use cases do not describe the internal mechanism, but describe the conditions, before and after the interaction with a quick description of it. In particular,they are shown and described as follow:

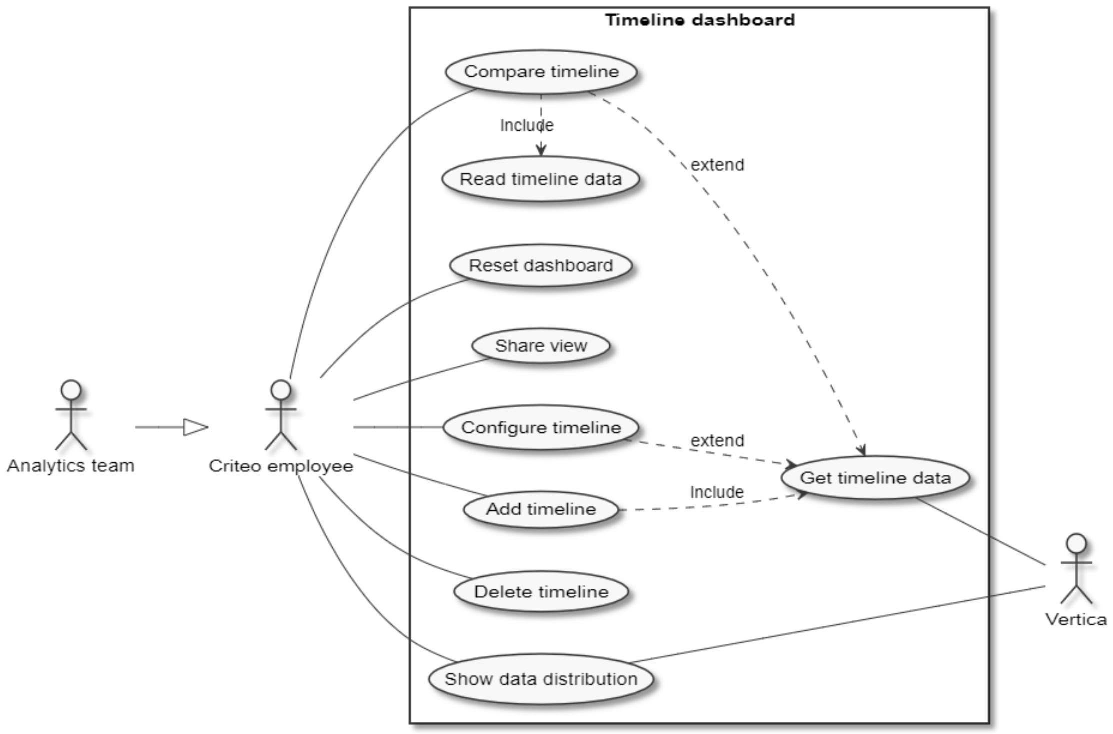


Figure 4.4: Use cases

Now here it is a quick description for each use case.

Name	Add timeline
ID	UC1
pre-conditions	user got access through VPN to the application and N timelines are shown (with N >= 0)
post-conditions	N+1 timelines are shown
nominal scenario	1. user click button to add a timeline 2. a new timeline is added 3. the timeline get data from Vertica 4. now the user can interact with it
non-nominal scenario	1. User click button to add a timeline, but Vertica is overloaded and/or no connection 2. a new timeline is added 3. the timeline cannot get data from Vertica and notify the user

Table 4.5: Use case 1 - add timeline

Name	Delete timeline
ID	UC2
pre-conditions	user got access through VPN to the application and N timelines are shown (with N>=1)
post-conditions	N-1 timelines are shown
nominal scenario	1. user click button to delete a specific timeline 2. that timeline is deleted
non-nominal scenario	

Table 4.6: Use case 2 - delete timeline

Name	Reset dashboard
ID	UC3
pre-conditions	user got access through VPN to the application and N timelines are shown (with N>=0)
post-conditions	0 timelines are shown
nominal scenario	1. user click button to reset the dashboard 2. every timeline is deleted
non-nominal scenario	

Table 4.7: Use case 3 - reset dashboard

Name	Share view
ID	UC4
pre-conditions	user got access through VPN to the application and N timelines are shown (with N>=0)
post-conditions	a JSON is exported and imported in another page
nominal scenario	1. user click button to share the view 2. a JSON of the view is exported 3. the JSON is imported in another or the same page 4. the new dashboard is loaded
non-nominal scenario	1. user click button to share the view 2. a JSON of the view is exported 3. the JSON is imported in another page but with another version of the application which does not support the JSON structure passed 4. An error happens and the user is notified

Table 4.8: Use case 4 - share view

Name	Configure timeline
ID	UC5
pre-conditions	user got access through VPN to the application and N timelines are shown (with N>=1)
post-conditions	the timeline chosen gets a new configuration
nominal scenario	1. user click button to configure a specific timeline 2. user update configuration 3. the new timeline is loaded and contacts Vertica if the options that requires an updated were changed
non-nominal scenario	1. user click button to configure a specific timeline 2. user update configuration 3. the new timeline is loaded and contacts Vertica if the options that requires an updated were changed, but Vertica is overloaded and/or no connection 4. the timeline cannot get data from Vertica and notify the user

Table 4.9: Use case 5 - configure timeline

Name	Compare timeline
ID	UC6
pre-conditions	user got access through VPN to the application and N timelines are shown (with N>=1)
post-conditions	the timelines are compared step by step through the percentage variation
nominal scenario	1. user click button to compare two timelines 2. user select two timelines 3. a comparison step by step is obtained by reading timelines data or by contacting Vertica if data was not loaded yet
non-nominal scenario	1. user click button to compare two timelines 2. user select two timelines 3. a comparison step by step is obtained by reading timelines data or by contacting Vertica if data was not loaded yet, but Vertica is overloaded and/or no connection 4. the comparison cannot get data and notify the user

Table 4.10: Use case 6 - compare timeline

Name	Show data distribution
ID	UC7
pre-conditions	user got access through VPN to the application and N timelines are shown (with N>=1)
post-conditions	the data distribution of a timeline is shown
nominal scenario	1. user click button to compute data distribution of a timeline 2. a modal is shown with two charts, one shows the original data distribution while the other one shows the modified data distribution by application every timeline option
non-nominal scenario	1. user click button to compute data distribution of a timeline 2. a modal is shown with two charts, one shows the original data distribution while the other one shows the modified data distribution by application every timeline option, but Vertica is overloaded and/or no connection 3. the comparison cannot get data and notify the user

Table 4.11: Use case 7 - show data distribution

Next, a mapping between functional requirements and use cases is shown in order to understand how they are connected to each other.

	UC1	UC2	UC3	UC4	UC5	UC6	UC7
FR1	X	X	X				
FR2					X		
FR3	X						
FR4	X						
FR5					X		
FR6					X		
FR7					X		
FR8					X		
FR9					X		
FR10					X		
FR11					X		
FR12					X		
FR13					X		
FR14					X		
FR15					X		
FR16					X		
FR17					X		
FR18							X
FR19							X
FR20					X		
FR21				X			
FR22				X			
FR23						X	

Table 4.12: Mapping use cases and functional requirements

4.6 Data export

The application would have never worked without the data needed. These data are gathered by some loggers such as the service CSM, already described in previous chapters, and then stored on the main database system which is Hive. However, this system is incredibly slow, especially when the number of rows to query is very high. So, it was needed to export data through some jobs on a new system called Vertica. It is a powerful database management system with a lot of features and one of the fastest in the world.

Although all these benefits, moving data directly to Vertica is not exactly a good idea. In fact, the original data is raw which means there is a good deal of information it is not desired. Beside that, it is preferable to aggregate and modify them in order to get already computed data and avoiding in this way to repeat the same process each time a timeline is displayed.

It was also needed to consider if it is better to give a certain level of freedom to those who want to perform analysis. In fact, a high degree of aggregated data can cause a drastic reduction on the set of options to configure a timeline, but at the same time faster execution. So, it was needed to find a good trade-off between functionality and performance.

Since it was required to work on two specific event sources which are banner_pointer_events and banner_viewability, then it was needed to export data from their original tables stored in Hive and shown in figure 4.5.
The first table, *enginejoins.bid_display_click_visit*, contains the keys *arbitrage_id* and *impression_id* that identify the user and its banner impression. This table is used as reference for the keys since it is the one that contains them all.
The table *glup_parquet.banner_view* contains the *banner_viewability* events stored in the column *event_type*. There is another column called *tracking_method* which is a further column used for filtering the type of tracking you want to consider in the dataset. For this reason this column is exported as well. It also contains the needed timestamp to compute the event sequences.
Finally, *glup_parquet.banner_pointer_event* is the table which contains the events *banner_pointer_events*, more specifically in the column *pointer_event*. As for the previous table, here is the timestamp needed.

All these tables are characterized not only by the keys, but also by *day*, *hour* and *host_platform*. The first two columns define the day and the hour of the database partition that recorded the corresponding event. While the latter defines the location where this event was recorded such as Europe or USA.

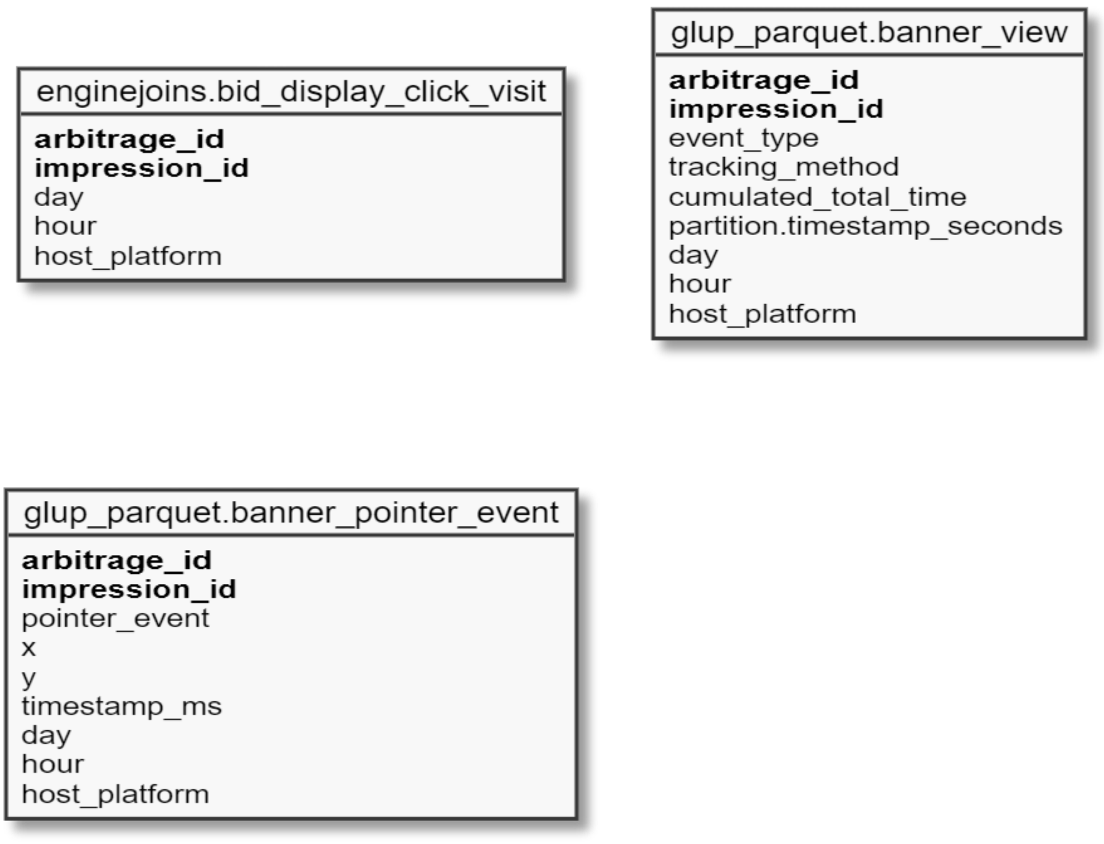


Figure 4.5: Starting data tables

After different designs and tests with the aggregated data, the final table used by the project has been made. This new table has to contain the events required and their filters needed to restrict the range of data to consider. The figure 4.6 shows the final result achieved to get the new data structure compatible with the application built.

Here is a quick explanation for these columns:

- The columns in **bold** identify uniquely a specific user interacting with a banner
- *day* and *hour* define when these events happened
- the column *index* specifies in which step the event happened in the user history
- *host_platform* indicates the location of user such as Europe or United States
- *banner_viewability___event_type* and *banner_pointer___pointer_event* are two different type of events. The first ones are **banner viewability**, while the second ones are **banner pointer events** already mentioned before.

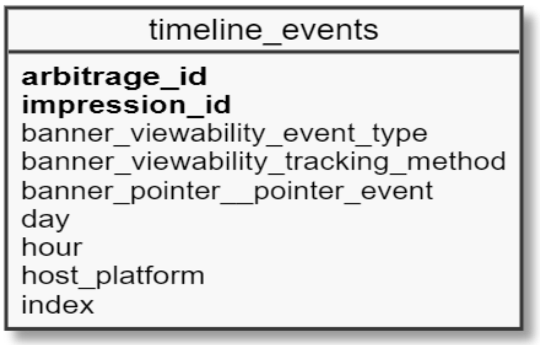

timeline_events
arbitrage_id
impression_id
banner_viewability_event_type
banner_viewability_tracking_method
banner_pointer__pointer_event
day
hour
host_platform
index

Figure 4.6: Final exported data table

Thanks to this structure, it is possible to generate a timeline showing one of the types of events available. Each step is represented by the index which is the order of actions. The application built is using by definition the columns *day* and *host_platform* as main filters for all events, and so for both *banner_viewability___event_type* and *banner_pointer___pointer_event*. The filter that can only work for a specific type of event is declared with name that follows this structure *TypeEvent_NameFilter*. In this case *banner_viewability* events are characterize by a further filter called *banner_viewability_tracking_method*.

The data export consists in some jobs running to work on the raw data from Hive to Vertica. In order to create and run these jobs, it was needed to work on a project made by the BigDataFlow team for this scope on the following files:

```
bigdataflow
└─ sql
   ├─ ...
   └─ creator
      ├─ ...
      ├─ csm_timelines_by_impression_id.sql
      ├─ csm_timelines_by_impression_id.sql.schema
      └─ csm_timelines_by_impression_id.conf
```

The *.sql* file contains all the queries needed to describe the aggregation process. It starts by getting the raw data and ends by aggregating them, subsequently producing the final table. There is also information such as the retention of data which describe the interval of time of data we want. For instance, the last month.

The *.schema* file is used to export data on Vertica by describing the final schema of the table and a projection which is used to speed up the query execution. Without it, the table would be exported only on the other systems, excluding Vertica. It is useful not only for that, but also for optimization thanks to the projection. It is a crucial aspect that goes to affect the final performance.

The *.conf* file contains much information of configuration and not only. There is also a section for testing the queries used for aggregation. Thanks to it, we can make sure we get exactly what we want. If there are some queries not working correctly in some particular cases, the computed result will be different from the expected one and the job will be never run because of a failed test.

However, there are other tests running which are format and syntax test. All of them are run when the code is pushed on Gerrit. Further, in order to avoid some errors during the jobs' execution, it is possible to run a *dry run* which is a way to run temporary all jobs and make sure they are working. Once everything is done, we can show the jobs on the BigDataFlow page.

The process to export data on Vertica was not just a change on the code, but many changes in a way to achieve the final result expected. The need to aggregate more data and change a bit the final structure sometimes has caused, not only some delays, but also some issues. The system that exports data through a job has some strict rules to follow when you want to update the destination table structure. For instance, it is not possible to add a new column and rename another one already existing there at the same time. This rare case happened to me since I needed to add columns and rename others.

So, there are two solutions for this problem: split the update in two in a way that one adds new columns and the other renames, or totally delete the job and the table created to start from scratch. The second option is the easiest one, but it takes much more time since we need to recreate the job and the table to fill with new data again.

Instead, the first option just needs you to make two different updates. However, when you add new columns, they are filled only for the new data added, while the old data already exported will have NULL as value. In my case, NULL was a wrong value and I needed to recompute the whole dataset. This was possible through the **backfill** operation available on the BigDataFlow platform which allows to re-run the export job on the whole data again instead of executing only on the new added data.

Here it is possible to show all jobs, pause them and run some useful operations, such as *backfill* used to restart the jobs again and recompute all data.

4.7 Architecture

Front-end

The application has the general architecture described by the figure 4.7, while the figure 4.8 is the zoom on the front-end side.

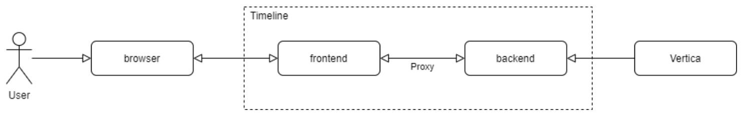


Figure 4.7: Timeline architecture

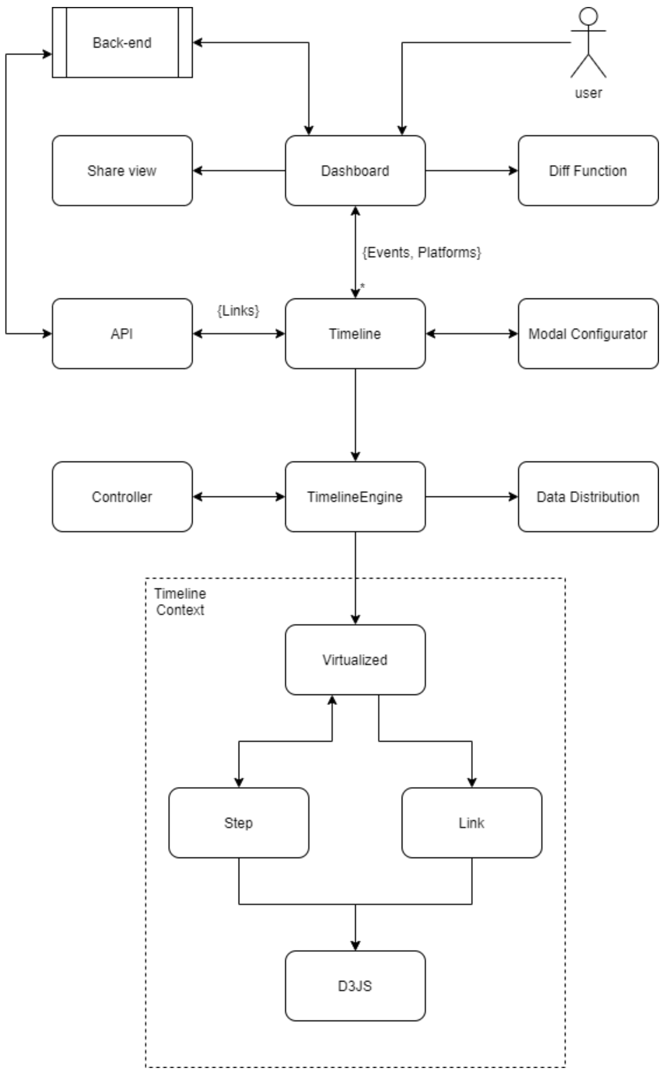


Figure 4.8: Front-end architecture

Thanks to an architecture like this one which corresponds to a MVC architecture, it is possible to show data through the timeline by keeping them updated as needed. Furthermore, it is perfect to manage optimizations in an easier way.
This architecture is just a simplified version of the real architecture, but it is enough to describe the main features available. Let's go step by step to explain the full working behaviour.

Everything starts with the **Dashboard** component which is responsible to show the main page with a list of timelines. Here, the user is able to add or remove timelines, reset a page or run special functions such the **Diff function**, used to make a comparison between timelines, or **share a view**. The dashboard also contacts the back-end to know which datasets, events, host platforms and other filters are available in the data source.

Each **timeline** gets the information needed by the Dashboard. This information is the datasets we can query, the events we can filters, the host platforms available and others. The timeline can be configured through a **modal** which contains different options to filter data and approaches to show them. It is also in charge to call the **API** component which in turn contacts the back-end to get data to show a single timeline. Indeed, given the large amount of data and considering the execution speed is one of the main requirements, the timeline is made to work in on-demand. In other words, it does not ask the back-end to run only one query to generate the whole timeline which can take also several minutes, but it runs a query for each step. It means smaller and faster queries are executed, calculated in few seconds, only when needed and, in this way, the memory required is reduced. However, this complicated more the fetch requests logic, but it was needed to avoid the **out-of-memory** error and the **timeout** fetch in the browser which is typically 3 minutes.

Next, there is the **timeline engine** which the core to generate the actual timeline SVG representation. Here is the **controller** and the feature to show the current **data distribution** of total number of users with total number of steps done.

As said before, the timeline is composed by **steps** and **links** made by a framework called **D3JS**. Both are instantiated through the **Virtualized** component. It is an important component of optimization. Thanks to it, each step and link is instantiate only when it is visible and in the view-port. In this way, the application requires memory for only what is visible and runs the queries only for them.

The library D3JS is powerful enough to build any SVG element, but it is not easy to master. In fact, there are some concepts to keep in mind when this tool must be used. In my case, the links to connect the different steps were one of the most complex base elements to generate. Indeed, initially, its generation was not so great and a little bugged since there were issue such as links in the wrong position with strange shapes, not generated or never deleted when there was a new timeline and so on. Furthermore, not only there were issues with visual elements, but also with the internal logic. At the beginning, there was the idea to generate a whole timeline with just one HTTP request. However, this was not possible for two reasons: running only one query takes much more time higher than timeout of the browser since it needs to read the whole data source, and generating every SVG element even if most of them were not visible would cause out of memory errors. So, the best solution was the on-demand to stream only the data actually needed.

Back-end

Now, let's focus on the back-end side architecture. In order to allow a large amount of customization in the queries, it was needed a system to generate new dynamic queries on basis some parameters passed by the timeline. The architecture that follows this requirement is shown in the next figure. The first component, called

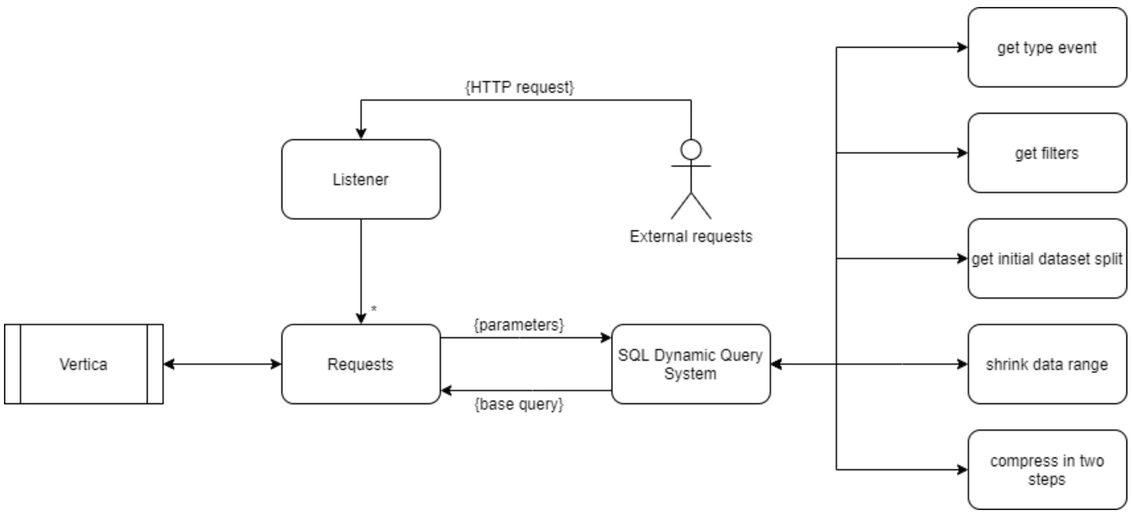


Figure 4.9: Back-end architecture

listener, is in charge to listen every **HTTP** request coming outside as for example a dashboard being used by a user.
This listener has to pass the request and all the parameters contained in the Query string to the corresponding request handler.

These requests, used to generate the timeline and all its different visualizations, are:

- get splitter
- get step N
- get total steps
- get events
- get filters
- get data distribution

As their name says, they are used to generate every aspect that ranges from the steps shown on the timeline to the events and filters that it is possible to analyze. There is also a handler to show the data distribution of the current timeline visualized.

However, these handlers have to send complex and dynamic queries to Vertica with the objective to get the result from the data source. This is done through the **SQL dynamic Query System** which is in charge to generate different SQL codes used by the handlers on basis the parameters passed and the type of request. This component contacts different functions with different goals. Some of them are used to generate the **WHERE** condition to filter the whole original dataset, others have to work with nested queries given that some final results are obtained through a higher complexity. In fact, in some situations, it is needed to change the starting table to filter data and generate some new columns useful for the representation of the timeline.

The component to generate dynamic queries in the back-end side needed to be tested it carefully through every possible combination of parameters to be sure that there are not queries generated in the wrong way. However, broken queries sometimes happened and, especially after adding new options that generates more complex ones of them. So, this situation forced in a certain point to change approach and make this component more modular in order to simplify the testing and its understanding.

The back-end, based on the language Scala, is developed to allow the communication among front-end and Vertica, the current DBMS used. In particular, it has the task to generate the query SQL code on basis of the parameters passed by the front-end and return the data computed by Vertica, so it is not a simple passage of data. On basis of the endpoint type, it may or not require some parameters to generate the appropriate query. The following table contains the set of properties available, then there is a description for each endpoint.

Name	**Type**	**Description**
dataset	string	the type of events to analyze
firstDay	date	the initial date to filter the dataset
lastDay	date	the last date to filter the dataset
iterationId	number	index of the step to obtain
split	string	the type of splitter to get
includeIndexNullInIteration0	boolean	show or not sequences with 0 length in the splitter
preFilter_{PROPERTY}[]	string[]	it is a filter of a specific property of a kind of events
collapseEvents	boolean	enable or disable the collapse of duplicated events
ignoreEvents[]	string[]	list of events to ignore
hideIgnoreEvents	boolean	hide or not the events to ignore
limitIteration0	number	minimum sequences length to consider
limitIteration1	number	maximum sequences length to consider
iterationLimitOrder	string	apply sequences length limit "before" or "after" ignoring, hiding and collapsing the events
showLastEventsInIteration2	boolean	show or not final events in step 2
preFilter	string	name of filter property to obtain its unique possible values
startingFromOriginalData	boolean	obtain or not data distribution without ignoring, hiding and collapsing the events

Table 4.13: List of parameters accepted by the back-end

All these parameters are sent by the front-end on basis of the configuration options, set by the user. Furthermore, the endpoints are made to combine this parameters to generate different queries combinations. As it is possible to infer, the number of combination is very high, so it was needed a deep testing on the back-end side to make sure that there are not broken combinations that make the DBMS crash, or worse generate a wrong data result.

In order to completely meet the requirements, the following endpoints are available and reachable from the outside (all of them have */dashboards/ad-timeline/api/datasets/csm_timelines_* as prefix) through the *GET* method. These endpoints are described by the file *Dashboard.scala* which is in charge to create these endpoints and listen for any request on them.

Endpoint	**Parameters**	**Default**
max	*includeIndexNullInIteration0* *dataset* *firstDay** *lastDay** *preFilter_{PROPERTY}[]* *collapseEvents* *ignoreEvents[]* *hideIgnoreEvents* *iterationLimitOrder* *limitIteration0* *limitIteration1*	false pointer events false [] false before 0 0
link_first	includes *max* endpoint params *split*	 ALL
link_next	includes *max*endpoint params *iterationId** *showLastEventsInIteration2*	 false
events	dataset	pointer events
prefilter	*preFilter*	first filter
interation_data_distribution	if *startingFromOriginalData* includes *max*endpoint params if *startingFromOriginalData* *dataset* *preFilter_{PROPERTY}[]* *firstDay** *lastDay**	false true pointer events

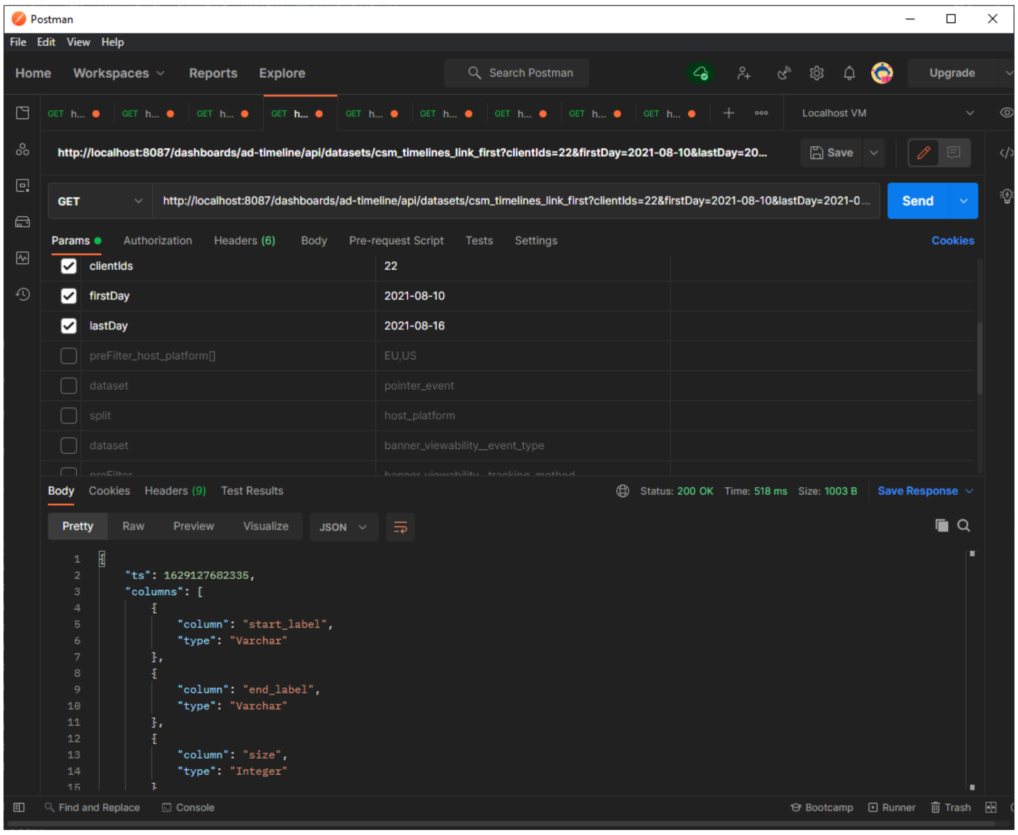
Table 4.14: List of parameters accepted by the back-end

4.8 Testing

Testing an application is one of the most important phases during the development. If there is bug, minor or critical, this can be caught only through a testing phase. There are two ways to test a software which are listed as follow:

- Manual testing
- Automatic testing

The first solution, **manual testing**, is the typical approach that the developer applies while he/she is coding its application. As the name says, it consists in just executing and *"playing"* with the software after every change made on the code. For *"playing"* is meant to say to interact with every element of the application in different ways by trying to find edge cases that can help to catch bugs. From this, we can infer that it is not feasible if the application is complex and too big in order to be tested in every its part. This is why there are usually teams engaged just to test the application and let the developers to focus more on coding and debugging to correct bugs found on the testing phase. It is good to mention also other tools, beside Intellij IDEA, that help to test a product. A first tool is **Postman** which allows to test the endpoints exposed on the back-end. Through this application it is possible, not only to specify the endpoint to contact, but also a ton of options of any type such URL parameters in case of GET requests, the body to send if POST requests and also authentication configuration.


Postman
File Edit View Help
Home Workspaces Reports Explore
Search Postman
Upgrade
Localhost VM
Save
Send
Params Authorization Headers (6) Body Pre-request Script Tests Settings
Cookies
Body Cookies Headers (9) Test Results
Save Response
Pretty Raw Preview Visualize JSON
Find and Replace Console
Bootcamp Runner Trash

http://localhost:8087/dashboards/ad-timeline/api/datasets/csm_timelines_link_first?clientIds=22&firstDay=2021-08-10&lastDay=20...

GET http://localhost:8087/dashboards/ad-timeline/api/datasets/csm_timelines_link_first?clientIds=22&firstDay=2021-08-10&lastDay=2021-0...

✓	clientIds	22
✓	firstDay	2021-08-10
✓	lastDay	2021-08-16
☐	preFilter_host_platform[]	EU,US
☐	dataset	pointer_event
☐	split	host_platform
☐	dataset	banner_viewability__event_type

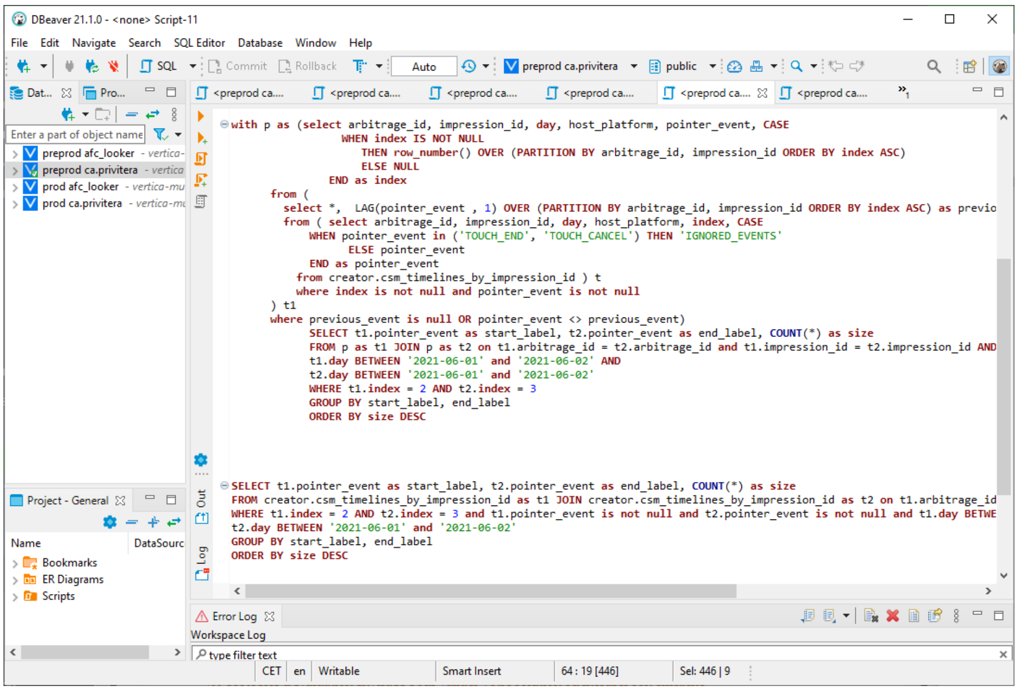
Status: 200 OK Time: 518 ms Size: 1003 B

```
{
    "ts": 1629127682335,
    "columns": [
        {
            "column": "start_label",
            "type": "Varchar"
        },
        {
            "column": "end_label",
            "type": "Varchar"
        },
        {
            "column": "size",
            "type": "Integer"
        }
```

Figure 4.10: PostMan interface

This tool comes handy for the dynamic SQL system running in the back-end since its complexity in debugging. In fact, many variables are involved when a query is generated and the number of combinations that it is possible to make is quite large. So, thanks to this software, it is possible to make multiple requests and save their parameters to use later on. Then, it is just needed to compare the results obtained and try to analyze them to be sure that the queries generated are correctly working.

However, testing the back-end with the only usage of Postman is not so comfortable because if there is something wrong that breaks the execution, then we are force to stop the application, change the code and run it again. So, A second tool is **DBeaver CE** which is good to write SQL queries and send them directly to the DBMS.


DBeaver 21.1.0 - <none> Script-11
File Edit Navigate Search SQL Editor Database Window Help
SQL Commit Rollback Auto preprod ca.privitera public
<preprod ca....
Dat... Pro...
Enter a part of object name
preprod afc_looker - vertica-
preprod ca.privitera - vertica
prod afc_looker - vertica-mu
prod ca.privitera - vertica-m
Project - General
Name DataSourc
Bookmarks
ER Diagrams
Scripts
Out
Log
Error Log
Workspace Log
type filter text
CET en Writable Smart Insert 64 : 19 [446] Sel: 446 | 9

```
with p as (select arbitrage_id, impression_id, day, host_platform, pointer_event, CASE
                 WHEN index IS NOT NULL
                    THEN row_number() OVER (PARTITION BY arbitrage_id, impression_id ORDER BY index ASC)
                    ELSE NULL
               END as index
      from (
        select *,  LAG(pointer_event , 1) OVER (PARTITION BY arbitrage_id, impression_id ORDER BY index ASC) as previo
        from ( select arbitrage_id, impression_id, day, host_platform, index, CASE
            WHEN pointer_event in ('TOUCH_END', 'TOUCH_CANCEL') THEN 'IGNORED_EVENTS'
                  ELSE pointer_event
            END as pointer_event
          from creator.csm_timelines_by_impression_id ) t
          where index is not null and pointer_event is not null
      ) t1
      where previous_event is null OR pointer_event <> previous_event)
            SELECT t1.pointer_event as start_label, t2.pointer_event as end_label, COUNT(*) as size
            FROM p as t1 JOIN p as t2 on t1.arbitrage_id = t2.arbitrage_id and t1.impression_id = t2.impression_id AND
            t1.day BETWEEN '2021-06-01' and '2021-06-02' AND
            t2.day BETWEEN '2021-06-01' and '2021-06-02'
            WHERE t1.index = 2 AND t2.index = 3
            GROUP BY start_label, end_label
            ORDER BY size DESC


SELECT t1.pointer_event as start_label, t2.pointer_event as end_label, COUNT(*) as size
FROM creator.csm_timelines_by_impression_id as t1 JOIN creator.csm_timelines_by_impression_id as t2 on t1.arbitrage_id
WHERE t1.index = 2 AND t2.index = 3 and t1.pointer_event is not null and t2.pointer_event is not null and t1.day BETWE
t2.day BETWEEN '2021-06-01' and '2021-06-02'
GROUP BY start_label, end_label
ORDER BY size DESC
```

Figure 4.11: DBeaver interface

Of course, it requires a configuration to authenticate the user and allow the usage. This tool was needed to test the complex queries generated by the back-end. In fact, on basis of filters set by the user in the application built, the back-end can generate very complex SQL queries which contains a large amount of parameters and complex keywords that make the testing a big issue. Thanks to this tool it is possible to copy the queries generated by the back-end and test them in a easier and direct way without making any changes on the application. For instance, the queries shown on the screenshot 4.11 are quite long and are some example of possible queries generated.

Manual testing is important as much as **automatic testing**. In fact, automatic testing is a way to re-test the application over time while adding or removing features. For example, let's suppose to create different components, and we test them while we develop them. However, it can happen to add a new feature that can be broken. The operation to fix can take some time, but in the end it is patched. However, this fix can break another feature that it was tested manually before, and it is very likely we will not catch this issue anymore. The automatic testing is very useful in this case because we can re-run all tests automatically for each change made on the code. As consequence, we can consider manual testing error-prone and highly time-consuming. Instead, automatic testing allows us to save time and tells us exactly what went wrong. Furthermore, it can serve as documentation for anyone wants to use the project.

Regarding the back-end, the technology used is called **ScalaTest**. It is one of the most common tools used to test Scala code and not only. In fact, it is also compatible with other languages such Java and ScalaJS, included other tools for testing like Mockito to create stub functions [22]. I used the tool with the **FlatSpec** which is a testing style format. There are many of them on basis the needs, but this one has test structure more readable expressed like "*X should Y*" [12]. These tests have to be asserted in order to define whether a test passed or failed. **Matchers** is an element used in ScalaTest for doing exactly this by providing a way to check numbers, strings, objects and so on [18].

Regarding front-end side, it is used **jest**, a JavaScript testing framework, with a particular tool called **testing-library** to test frameworks like React in this project. This solution allows to simulate a usage of the application as a user would do it by interacting with the DOM nodes.
In order to do that, there are different ways to query these elements. One way is to get a base component, such as a button, through a **regex** which is a regular expression to obtain elements that satisfy it. Another way is to assign an identifier to the element in the code through the keyword *data-testid*. Once the wanted element is obtained, it is possible to get their parameter values such as their *text*, or trigger some events through the module **fireEvent** which can be for example a click on the button. The last but not the least, another tool used is **Storybook** which consists on adding some stories for e2e testing and UI isolation development. Each story is actually the representation and the description of one or more UI components which allows to interact with them in isolation and perform some testing described by the developer.

Both tools, front and back-end, were mainly used to test the most common and important components in the project, especially the core which stands as base for other elements. The idea was to cover an area as large as possible. These tests do not cover basic functions/objects that do not need further investigation. Also, the functions directly provided D3JS and React were skipped as well since they are already tested by their owners. Indeed, the effort was mainly conveyed on what actually a test was needed in order to achieve a certain level of confidence.

Chapter 5

Results

5.1 Optimization

Despite the complexity of the timeline itself, the attention was much more localized in the optimization aspect. Given the large amount of data and the big limits that a browser imposes, the optimization was crucial for the result to achieve. It is not only about improving the user experience, but also avoid crashes and other weird issues that could impact the usage of the tool. Furthermore, the solutions implemented were made to address indeed the nonfunctional requirements involved.

In this section are analyzed the biggest execution problems met during the development. So, it is described the source of issue, which implemented solution fixes it and finally few comparison examples. In particular, the aspects that are analyzed shortly are those that have to deal with performance and the request timeout. It is also described the drawbacks of these solutions that could affect other aspects of the project, such as the costs.

The first part is more focused on a specific aspect of the first issue, Out-Of-Memory. Here it is depicted the tool used to keep under control the performance in real-time and the solution proposed to make sure to limit the cause of this issue. In fact, many nonfunctional requirements were involved on this issue.

Instead, in the second part, it is outlined the approach used to fix the problem of the request timeout. Specifically, the timeout limit imposed by a browser when a HTTP request is send to a server. Since a timeline requires a lot of computation, it was need to optimize the requests and keep everyone of them below the limit in order to be sure the execution never stops because of it. However, it can still happen, although the solution implemented, for instance when there is a bad connection or when the DBMS Vertica is overloaded. That is why it is implemented a backup solution to restore the execution.

5.1.1 Out-Of-Memory

As it is possible to infer from the frameworks used such as D3JS, the application is quite complex to execute, especially because of its interface. In fact, the performance are not affected only by the number of elements shown a represented into the DOM of the browser, but also by their type that characterize them.

In particular, the most complex elements shown in the page are the SVG tags used to generate the timeline. The most frequent vector elements are:

- rect
- path
- line

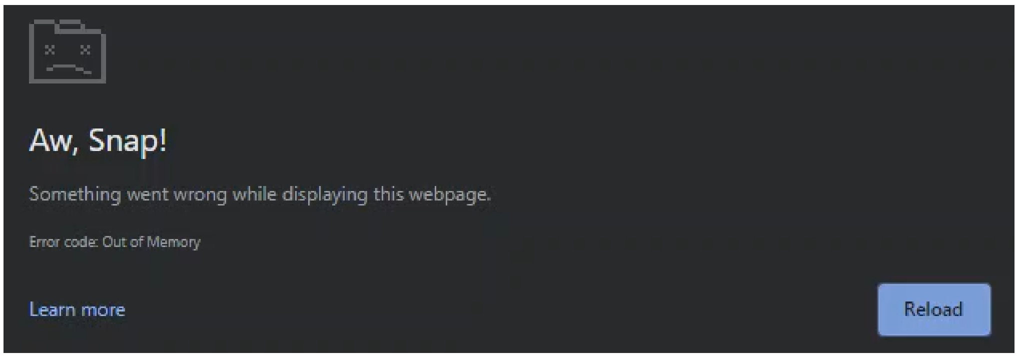
The rect tag is used to build the event blocks, the path tag to build connections between the events and finally the line is used to mainly represents the number of users who do not have events in the next step.

All of them have in common that they are **vector** tags. In other words, their representation is not based on a table of pixels like an image which is in this case a **raster** element, but they are based on set of rules to compute and draw the desired element.

The raster element have the problem to have a worse quality when they are shown bigger than their resolution. The benefit of these vectors elements is that they get the maximum visualization quality possible since they are generated when they are displayed. However, its representation is expensive in terms of computation cost. In fact, they are typically computed and drawn every time there is a change on the size of the element or when they are moved and rotated. Instead, the raster elements do not need any computation since its representation is always the same.

As it is possible to infer, D3JS makes a great use of SVG tags, so the timeline is very complex from this point of view. Any element added reduces the performance of the browser and so the final user experience. In fact, it was needed to consider every possible case of timeline generation to make sure the application never gets at the point to reach the limit of the memory available on the machine, and so the Out-Of-Memory error. So, the solution was the virtualized component made, already described before, that address this kind of issues.

The tools used to keep under control this aspect are the **developer tools** proposed by the browser Chrome and made by the Google company[2]. Thanks to

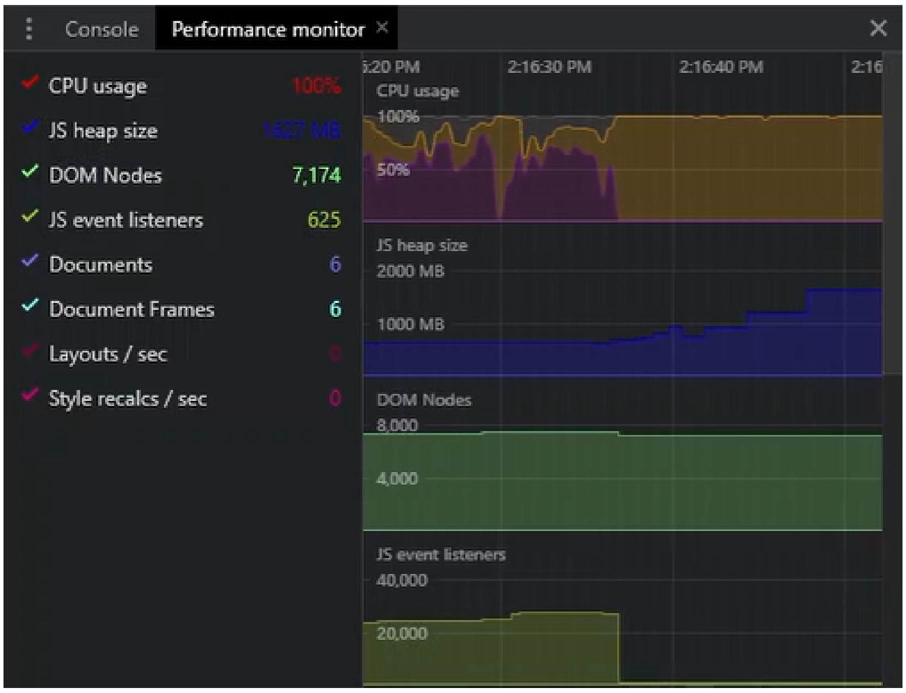


Aw, Snap!

Something went wrong while displaying this webpage.

Error code: Out of Memory

Figure 5.1: Out of memory error caused when there is no virtualized component



✓ CPU usage	100%
✓ JS heap size	1627 MB
✓ DOM Nodes	7,174
✓ JS event listeners	625
✓ Documents	6
✓ Document Frames	6
✓ Layouts / sec	0
✓ Style recalcs / sec	0

Figure 5.2: Performance without the virtualized component

such a tool, it is possible to keep track of many parameters such as CPU usage and JS heap size. If we take a look at the figure 5.2, it shows the execution performance with the virtualized component disabled in a particular edge case which is a very long timeline to generate.

As it is possible to notice, any action on this page brings the CPU to its full usage at around 100% and the JS heap size is increasing to load everything shown by the page up to reach a point to run out the memory. It must be considered that the project is running in pre-production/debug mode, so it is expected already to not have nice performance. However, the final result should still bring the same error, since it is inevitable.

In the figure 5.3, it is shown the performance with the virtualized component enabled. As it is clearly shown, the CPU and the memory can benefit of this solution. In fact, the CPU keeps a low usage while it is used almost constant, and the same can be said for the memory which is kept very low.

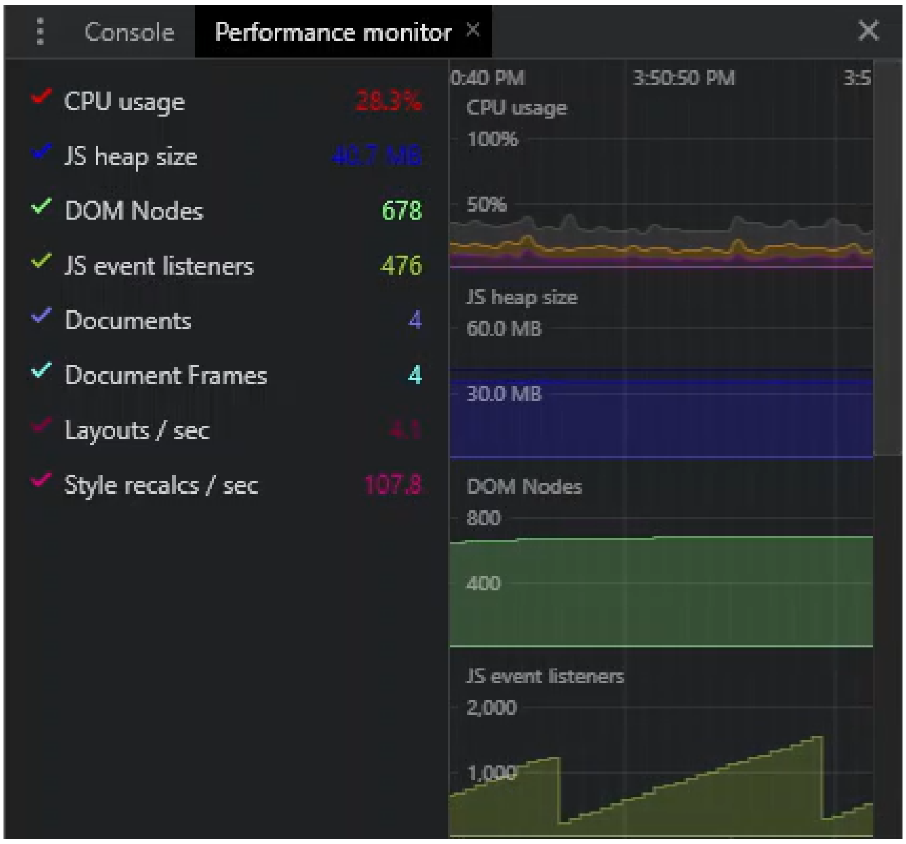


Figure 5.3: Performance with the virtualized component

This virtualized component was built with the aim to address this issue. Thanks to its mechanism, the browser loads only visible vector elements, while the others are not loaded at all and it is kept only little information needed to generate them.

5.1.2 Request timeout

Another issue that it is worth to talk about is one of the limit imposed by a browser, the so-called request timeout. Any browser has set a maximum duration for any HTTP call that comes from the application. This limit can be low or high and it was an issue for the execution that was needed to be addressed. For instance, the duration set for the browser Chrome is 3 minutes which is not so high when big data and computation are in play.

Usually the vendors of browsers set this limit for different reasons. A typical one is due to performance. The idea is to force developers to achieve and keep quite well the performance. Unfortunately, this limit cannot be set by the application running itself, but it must be set by the user usually by navigating the advanced settings of the browser to increase the limit of a request. Something that is not feasible for the standard user.

This limit had a big impact on the application execution since it was made to analyze a large amount of data, so it was expected to get long times before to obtain the final result computed. However, this forced to change approach on the query management. Instead to generate a timeline with just one query, this one is generated through multiple smaller queries. A comparison of these two approaches is shown in the figure 5.4.

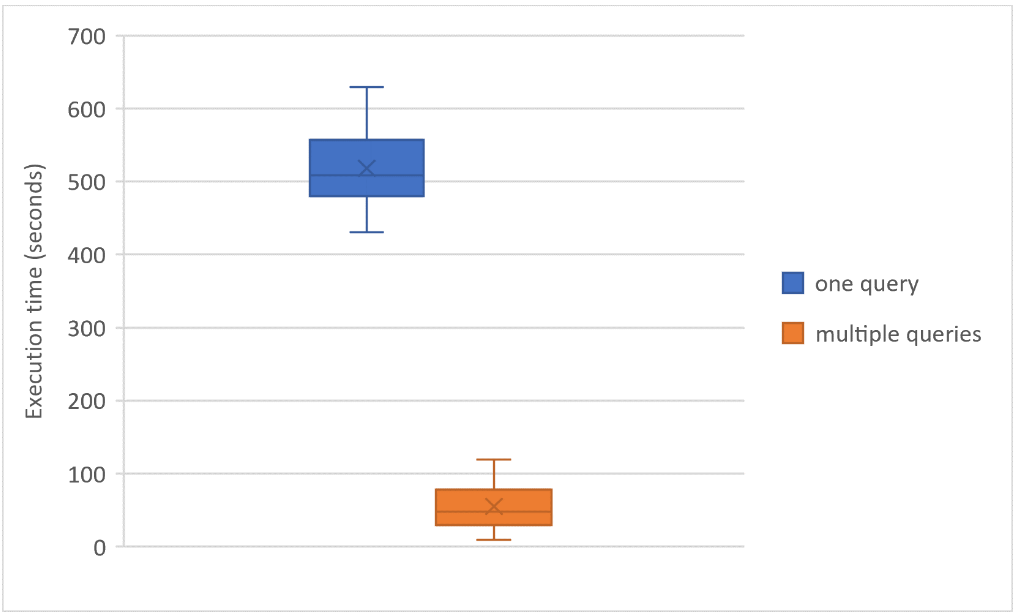


Figure 5.4: Comparison between one query and multiple queries to generate a timeline

These metrics are got many times through different timeline configurations. The variability of the values obtained is quite high, but it is something due to the overload on Vertica and its caching system. So, it was needed to run these tests more than one to get a certain level of confidence.

Initially there was the idea to use just one query to ease the API request management, In fact, through one query, it is possible to analyze the full timeline with just one initial wait, long but just one. However it was too much the waiting for the browser, so long that the request was aborted by the browser itself because it was overcoming the limit imposed. Beside this, any error connection that user has, it would cause automatically an abort of the request as well. So, in order to test it, it was need to increase the browser limit or to use the tool DBeaver to directly compute the query result with Vertica.

The other approach was to run multiple queries, but much lighter because they target just smaller parts to compute on the single timeline. This approach was possible thanks to the already implemented virtualized component. This element shows only the steps visible in the view-port and other few more as buffering process to make the usage smoother. This allowed to enable a sort of on-demand mechanism. Technically speaking, when a step of the timeline is loaded, then the API system is contact to run a query related to that step. In this way, Vertica needs just to read data filtered by step as well, which decreases a lot the quantity of rows to compute.

5.2 Features

After a deep analysis on the requirements and architecture, every feature of the application is represented in this section. Not only there is an introduction on them, but there are also some examples and reasons behind their mechanisms.

In the figure 3.5, it is shown the dashboard with an example of timeline generated. Through the title on the top of its box shows, not only the type of events under analysis, but also in which interval of dates data are obtained. There is also represented the *total number of steps* which is a number to indicate how many steps there are to finish all sequences. For example, this number could be quite big, especially if we want to perform a complete analysis. This was actually caused initially by the banner animations that were recorded as user events, but now no longer present thanks to a patch. However, this was an opportunity to address this kind of situations in the application through some features described afterwards.

It is also important to remind how it is possible to interact with it. There are some basic UI buttons to zoom in or zoom out a timeline, to go to the next/last or previous/first step. It i also possible to scroll it by dragging through a pointer. Every step can be filtered to hide some events that we do not care about, and we do not want to show.

5.2.1 On-Demand

A first feature that it is important to discuss is the on-demand mechanism implemented. It is the base for the optimizations described before, so it allows to to resolve the out-of-memory error and the request timeout.

This mechanism works for each timeline added in the page and it follows the following steps:

1. a timeline is added in the page
2. each step in the view-port is correctly loaded by calling its respective API call
3. while the API call is loading, the step shows a blinking yellow bar at the top
4. when its API call returns a response
   (a) if the step is visible, the step shows a fixed green bar at the top and its visualization is correctly loaded
   (b) if the step is not visible, then it is visualized with a green bar only when it is visible again
5. if there is any error, then the bar is set to red and the user can refresh the API call again or refresh the whole page since the page state is store locally

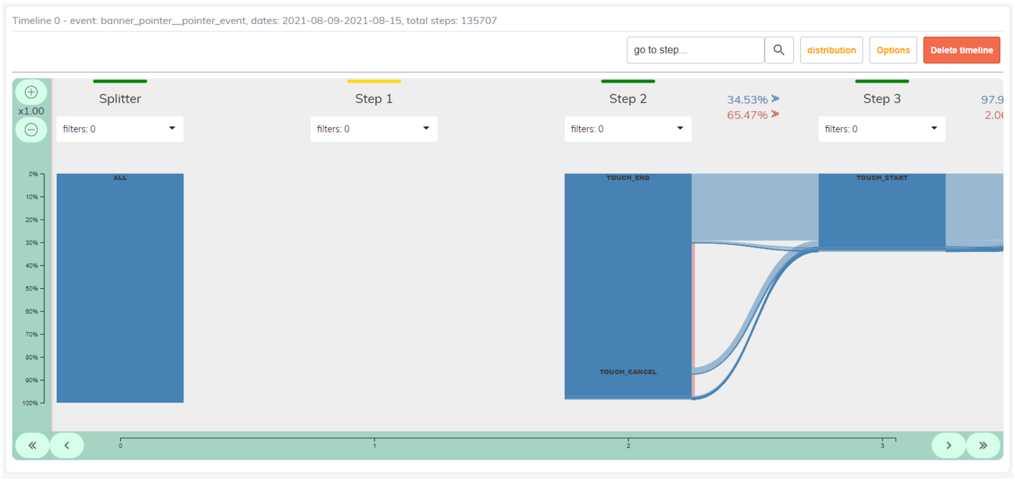
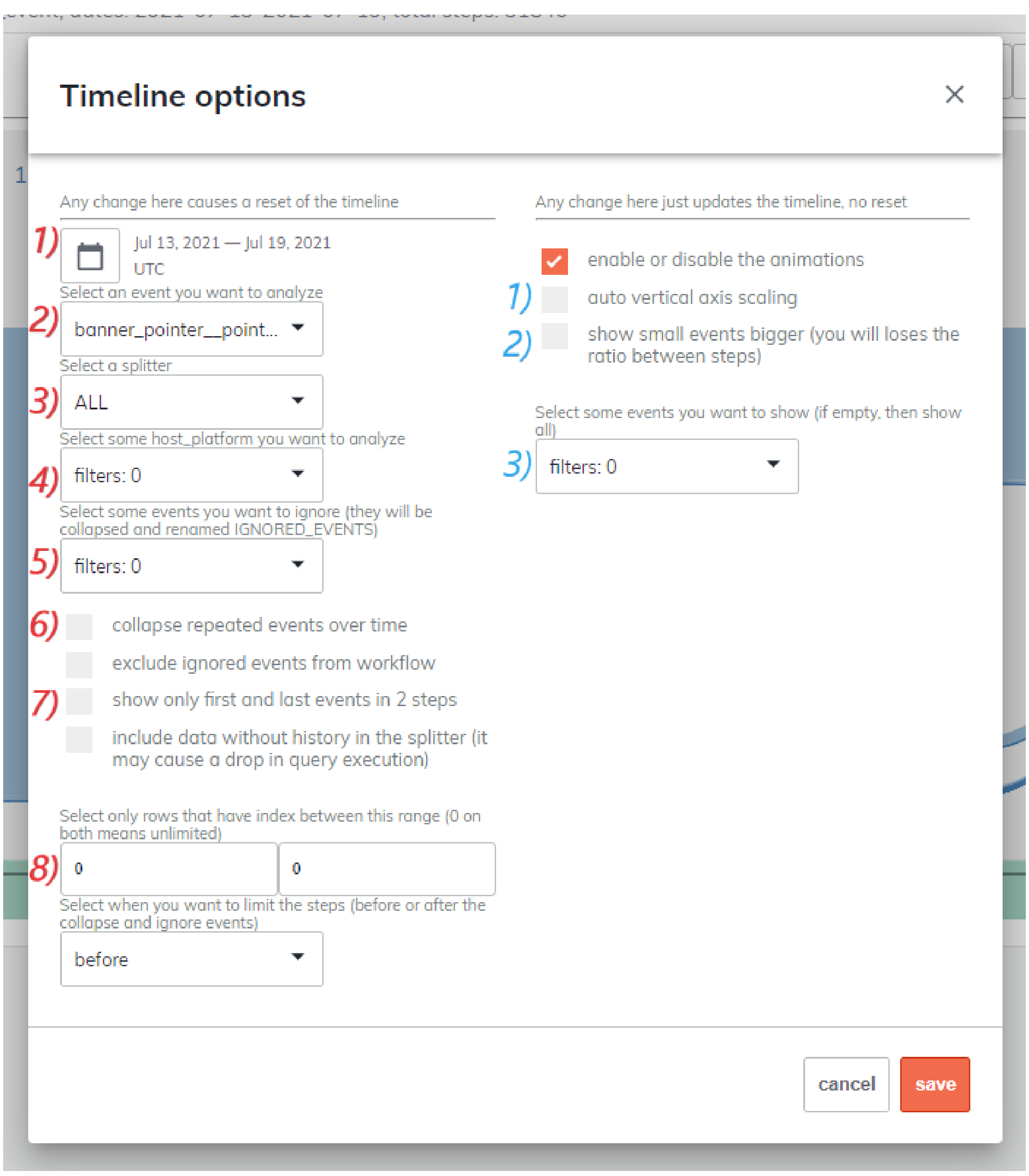

Timeline 0 - event: banner_pointer__pointer_event, dates: 2021-08-09-2021-08-15, total steps: 135707

Figure 5.5: Loading steps example

Everything describe in the process is possible thanks to the virtualized component which is in charge to run this task. Furthermore, the user is kept informed through this bar color drawn at the top of each step as shown on the figure 5.5.

5.2.2 Configuration

A timeline can be added or removed, but it can be also configured through a modal as shown here:



Timeline options

Any change here causes a reset of the timeline

1) Jul 13, 2021 — Jul 19, 2021 UTC

Select an event you want to analyze

2) banner_pointer__point...

Select a splitter

3) ALL

Select some host_platform you want to analyze

4) filters: 0

Select some events you want to ignore (they will be collapsed and renamed IGNORED_EVENTS)

5) filters: 0

6) collapse repeated events over time

exclude ignored events from workflow

7) show only first and last events in 2 steps

include data without history in the splitter (it may cause a drop in query execution)

Select only rows that have index between this range (0 on both means unlimited)

8) 0 | 0

Select when you want to limit the steps (before or after the collapse and ignore events)

before

Any change here just updates the timeline, no reset

✓ enable or disable the animations

1) auto vertical axis scaling

2) show small events bigger (you will loses the ratio between steps)

Select some events you want to show (if empty, then show all)

3) filters: 0

Figure 5.6: Timeline options

A quick description is given on the following list:

- options on the left that **require** a refresh of the timeline:
    1. date filter
    2. event type selector
    3. select a splitter
    4. other filters (such as host_platform)
    5. ignore events and hide them
    6. collapse repeated events over time
    7. show only first and last events in two steps
    8. restrict range of data
- options on the right that do **not require** any refresh:
    1. auto vertical axis scaling
    2. show small events bigger
    3. graphical filters

Here there are many options to perform a more detailed analysis and give the user a way to custom the visualization and the data obtained.

These options can be combined in different ways on basis of the user needs. This wide range of possibilities was practicable thanks to the final data structure imposed on Vertica. If we want to go more in details, the options on the right do not require any refresh because they do not affect the data source to obtain, while the options of the left require an update by running again the API requests to get a new timeline since they affect instead the data source.

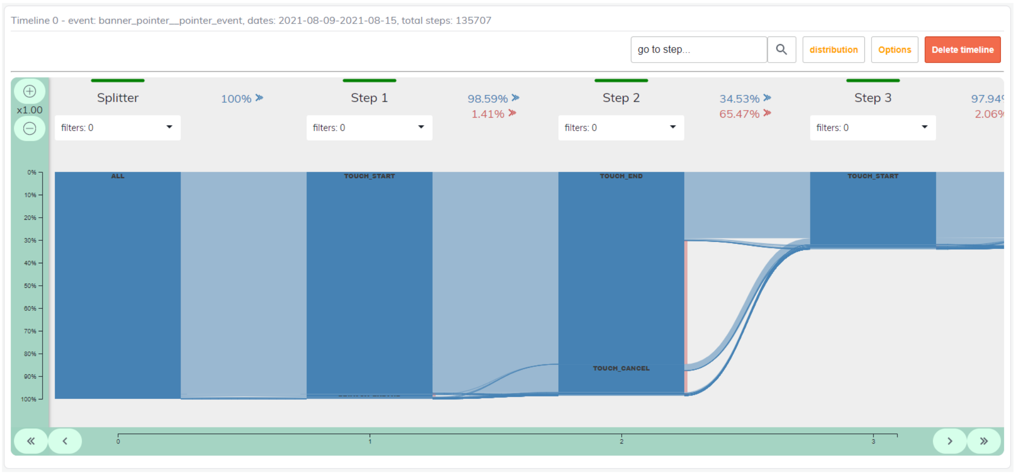
In order to configure the timeline, it is needed to reach this modal which contains the whole set of configuration. It is just needed to click the *options* button, shown at the top-right of the modal, to open the modal. Once it is open, it is possible to make changes, enable or disable options, and it also possible to cancel the last changes made by clicking the button *cancel*. After selecting the options desired, it is just needed to click *save* in order to go back to the timeline and, on basis of the options selected, there will be a refresh or not.

In the next sections, option by option is analyzed as well as the possible situations where they can be greatly effective. Let's go step by step to describe them and understand when they could be useful.

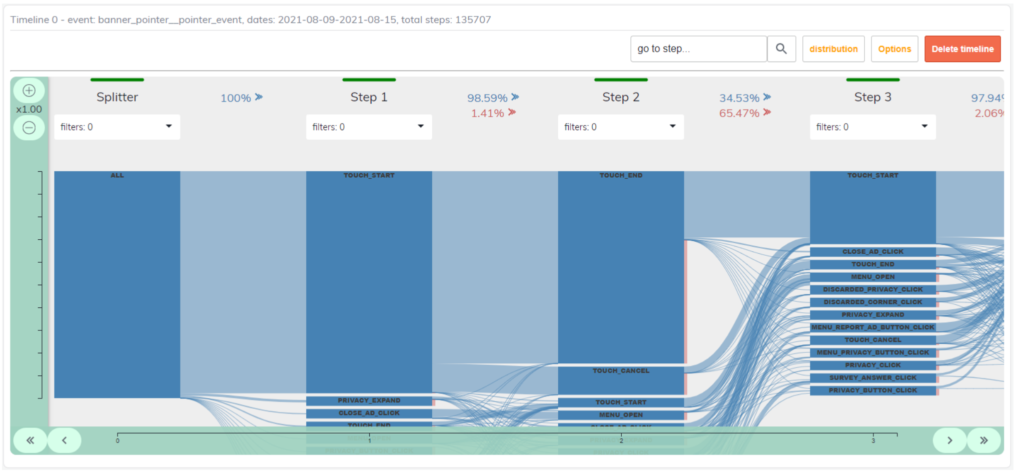
5.2.3 UI options

The options on the right shown on the modal in figure 5.6 are less important than the others since they affect only the interface, so they are used just to improve the data visualization. For this reason, they do not need a refresh on the API requests. In brief, they are capable to perform an automatic zoom in while scrolling, make the small events bigger and show/hide some events.

The figure 5.7 shows a comparison with these options. In the image *a*, the timeline does not have any interface option enabled. Instead, the image *b* has the option *show small events bigger* enabled.


Timeline 0 - event: banner_pointer__pointer_event, dates: 2021-08-09-2021-08-15, total steps: 135707

(a) Timeline with no UI option enabled


Timeline 0 - event: banner_pointer__pointer_event, dates: 2021-08-09-2021-08-15, total steps: 135707

(b) Timeline with small events bigger

Figure 5.7: Comparison between timeline with no changes and with small events bigger

As it is visible, without this option active, the events not frequent are not so visible because of the need to keep correct the scale shown on the left of the timeline. The alternative is to zoom in through the two button on the top-left of the box, but it is not comfortable in some cases. So, through this option checked, the small events have a minimum size, but we lose the scale reference, that is why the scale is hidden on the left. This option could be useful when there is not interest on keeping the scale since the events are sort by size on each step.

Beside that option, it is also possible to filter the events in a specific step as shown in figure 5.8. This can help a lot when we want to focus more the attention a specific events and leave out the others. In case there is not interest

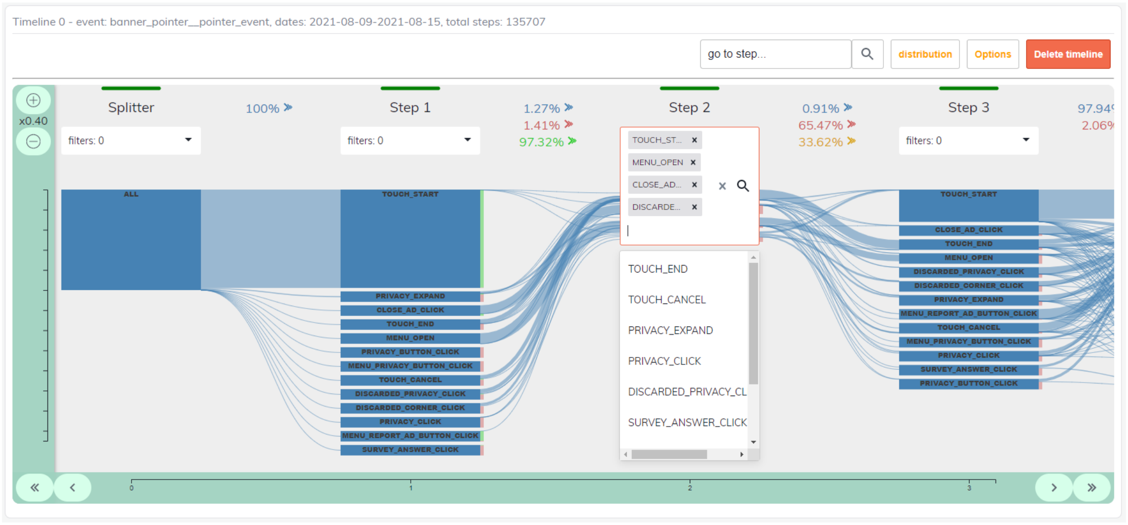
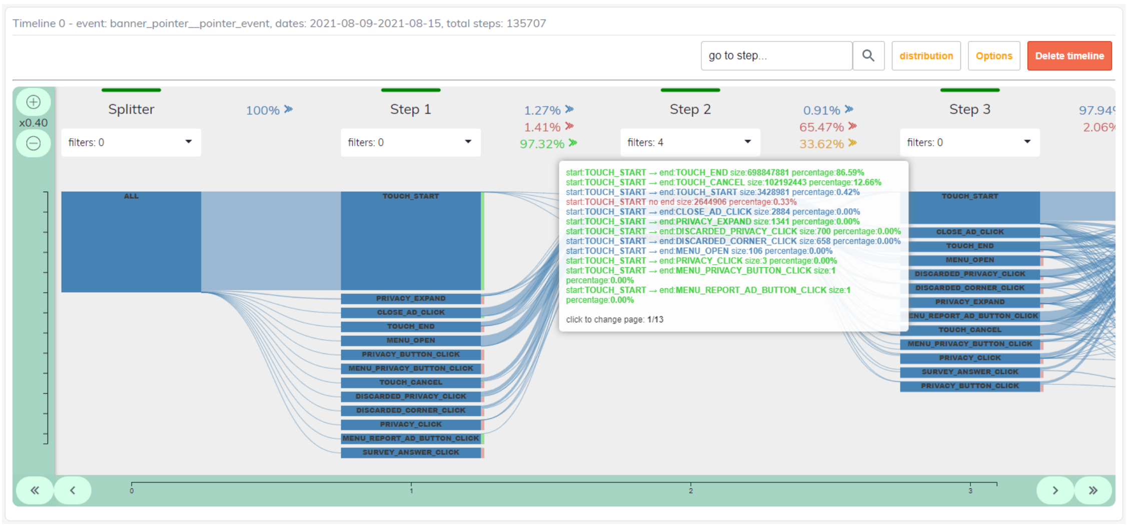

Timeline 0 - event: banner_pointer__pointer_event, dates: 2021-08-09-2021-08-15, total steps: 135707

Figure 5.8: Filter events in a specific step

at all on the visualization of specific events on every step, then it is possible to filter them through a global filter in the configuration modal of the timeline. We can also combine global filter and step filters on basis of our needs. Some, lines gets the color green in order to say that they are connected with some hidden events.

Next, each timeline already shows much information on what it is visualized. Although this, we can get much more information by moving the mouse over some elements. They are:

- top of a step
- top between two steps
- a event block
- a line which connects two events



Timeline 0 - event: banner_pointer__pointer_event, dates: 2021-08-09-2021-08-15, total steps: 135707

```
start:TOUCH_START → end:TOUCH_END size:698847881 percentage:86.59%
start:TOUCH_START → end:TOUCH_CANCEL size:102192443 percentage:12.66%
start:TOUCH_START → end:TOUCH_START size:3428981 percentage:0.42%
start:TOUCH_START no end size:2644906 percentage:0.33%
start:TOUCH_START → end:CLOSE_AD_CLICK size:2884 percentage:0.00%
start:TOUCH_START → end:PRIVACY_EXPAND size:1341 percentage:0.00%
start:TOUCH_START → end:DISCARDED_PRIVACY_CLICK size:700 percentage:0.00%
start:TOUCH_START → end:DISCARDED_CORNER_CLICK size:658 percentage:0.00%
start:TOUCH_START → end:MENU_OPEN size:106 percentage:0.00%
start:TOUCH_START → end:PRIVACY_CLICK size:3 percentage:0.00%
start:TOUCH_START → end:MENU_PRIVACY_BUTTON_CLICK size:1
percentage:0.00%
start:TOUCH_START → end:MENU_REPORT_AD_BUTTON_CLICK size:1
percentage:0.00%

click to change page: 1/13
```

Figure 5.9: Tooltip and information shown

In the example 5.9, it is shown a tooltip appearing when the mouse is moved over the top between two steps. This tooltip gives more information such as how many users moved from an event to another, its percentage, which connections are hidden and the number of events not connected in the next step.

5.2.4 Splitter, ignore and collapse events

The options shown on the left of the configuration modal directly work with the back-end to filter data and generate new different timelines. This is why a refresh is needed when at least one of these options is changed. A part of them (options 1, 2, 4 in section 5.2.2) are mainly used to filter data by *date* or other parameters, and to decide the type of event to analyze.

Through the option 1, we can get the whole sequences that have their timestamp in the range defined. Instead, with the option 2, we can select the sequences we want to analyze, and so banner pointer events and banner viewability. Finally, the option 4 allows to restrict the sequences to consider by filtering them by other parameters available such us *host_platform*.

Then, there is another option (number 3 in section 5.2.2) used to split the dataset at the beginning. For example, the we call split a priori the dataset by a specific parameter available into the interface. The figure 5.10 is splitting the dataset by host_platform, or in other words, by location such as Europe, Asia and Unite States.

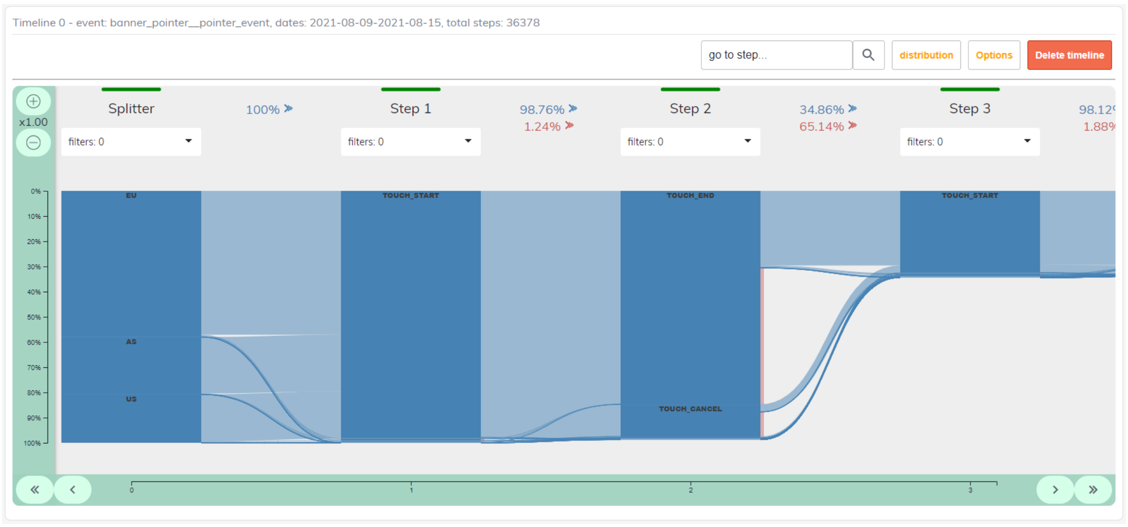


Figure 5.10: Dataset split by host_platform

Next, all the other options are used to reduce the total number of steps and make the timeline more readable. An example is the function **ignore events**. As the name says, this option allows to choose some events and ignore them by excluding them or replacing their name with **IGNORE_EVENTS**. In some situations, it is very useful since it can happen that there are different events in sequence, but not really useful. For instance, if there is a sequence of events like *TOUCH - SCROLL - SCROLL - OPEN* and we want to ignore *TOUCH* and *SCROLL*, then the new sequence would be *IGNORE_EVENTS - OPEN*.

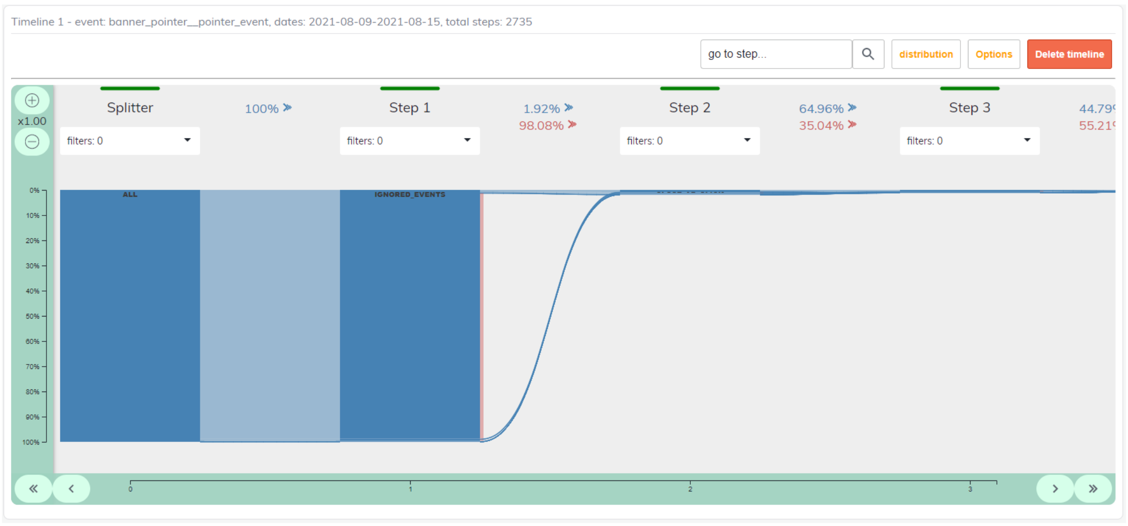
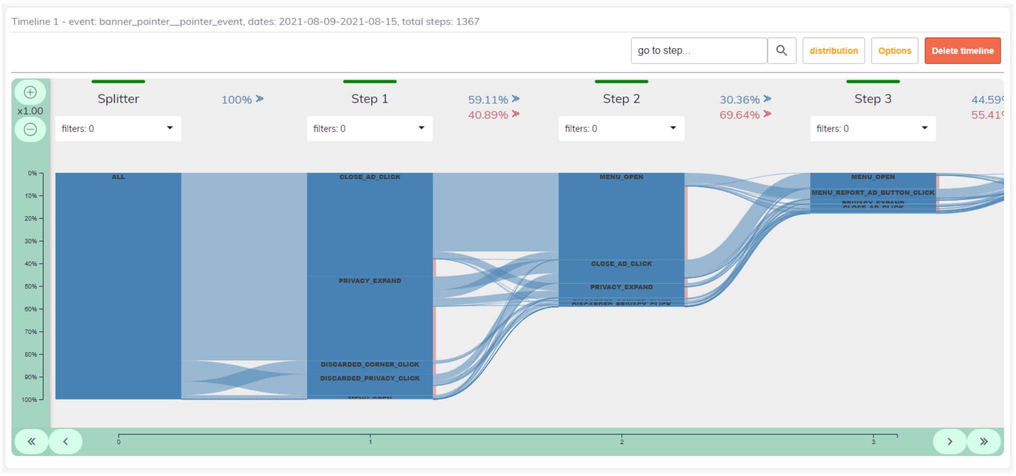
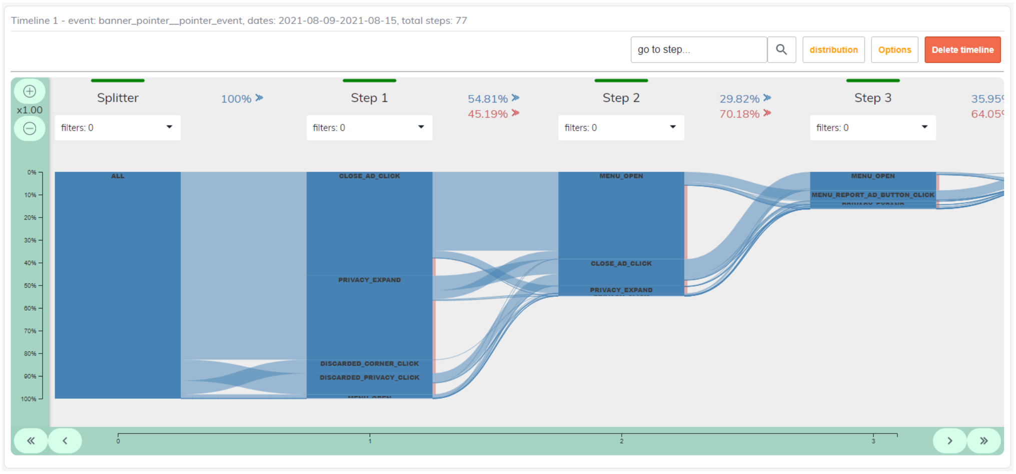


Figure 5.11: Example of TOUCH events ignored

This can drastically reduce the length of the timeline, especially if we enable the option to exclude the label IGNORE_EVENTS from the workflow in combination with the option **collapse repeated events over time** which has the goal to collapse multiple equal events in sequence as one single event. Here is a comparison of the usage of these two options:


Timeline 1 - event: banner_pointer__pointer_event, dates: 2021-08-09-2021-08-15, total steps: 1367

(a) Ignore events and excluded from the workflow


Timeline 1 - event: banner_pointer__pointer_event, dates: 2021-08-09-2021-08-15, total steps: 77

(b) Ignore, exclude from the workflow and collapse events

Figure 5.12: Comparison between ignore events and ignore with collapse events

As it is possible to notice on the title of the timelines, the total number of steps decreased to less than one hundred (originally there were thousands steps without any configuration).

5.2.5 Show only first and last events in two steps

In case, we do not care what the users did in the middle of steps, then it may be useful the option **show only first and last events in two steps**. Thanks to it, the timeline will be reduced to just 2 steps in which the step 1 shows the first events triggered by the users and the step 2 the last events.

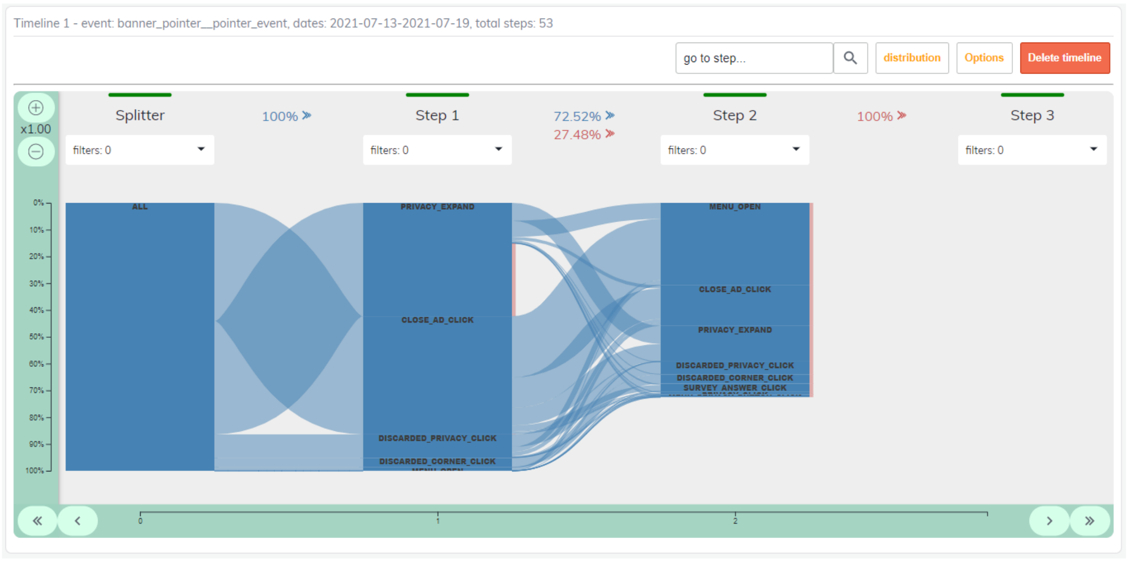


Figure 5.13: Show only first and last events in two steps

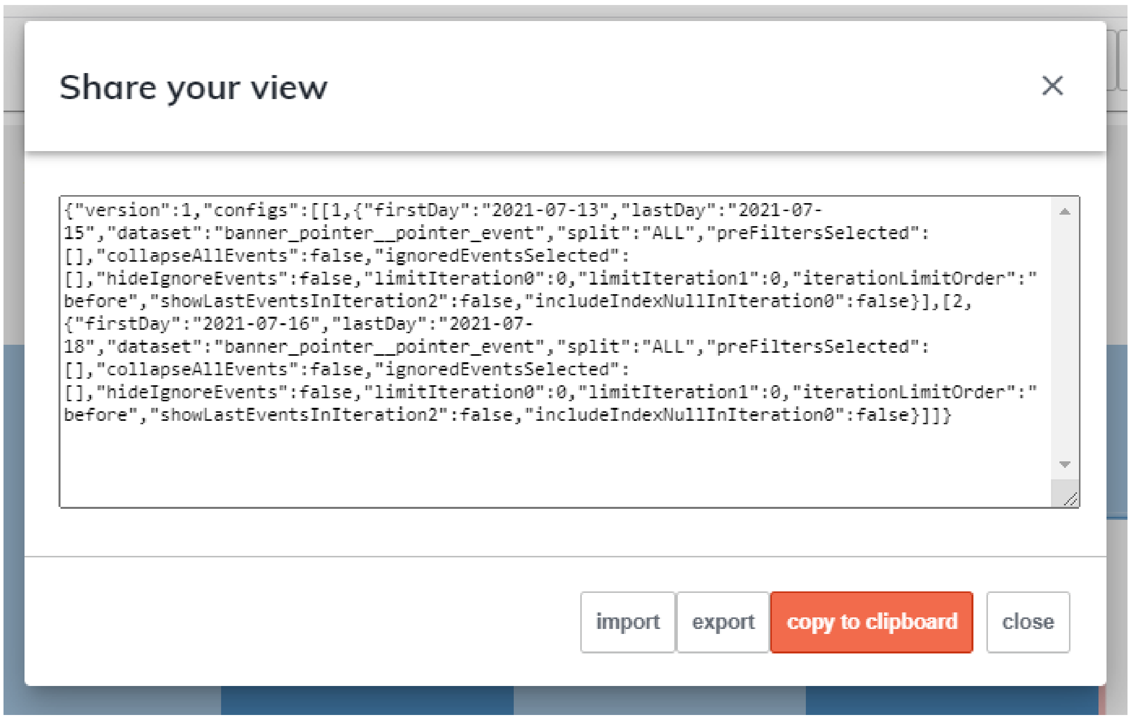
In this example (figure 5.13), it is possible to notice that the links could be overlapped. As mentioned before, the order of events in each step could make different links overlap each other to reach the destination. For instance, the link which connects *ALL* to *CLOSE_AD_CLICK* is overlapped by the link *ALL* to *PRIVACY_EXPAND*. In any case, we just needed to move the mouse over a link to make it change the color and highlight it.

This kind of configuration can help when we want to focus more the attention only on the first and final events triggered in the sequences. However, the quantity of information lost is a lot, and could affect in some case the outcomes of a data analysis.

Although all these drawbacks, this option can be used with the Diff function, used to compare two different timelines. Since this function depends on the timeline configuration, we can directly compare the initial and the final events if we enable the corresponding option. The Diff function is detailed in the next sections with more information about it.

5.2.6 Save and share the view

However, share our findings with others is important as much as a good analysis. So, there is a feature to export or import a JSON configuration of the current view. Initially, there was the idea to share it through a dynamic URL, but it could have a limit length[24] and that was a problem since a configuration can get very long.



```
{"version":1,"configs":[[1,{"firstDay":"2021-07-13","lastDay":"2021-07-
15","dataset":"banner_pointer__pointer_event","split":"ALL","preFiltersSelected":
[],"collapseAllEvents":false,"ignoredEventsSelected":
[],"hideIgnoreEvents":false,"limitIteration0":0,"limitIteration1":0,"iterationLimitOrder":"
before","showLastEventsInIteration2":false,"includeIndexNullInIteration0":false}],[2,
{"firstDay":"2021-07-16","lastDay":"2021-07-
18","dataset":"banner_pointer__pointer_event","split":"ALL","preFiltersSelected":
[],"collapseAllEvents":false,"ignoredEventsSelected":
[],"hideIgnoreEvents":false,"limitIteration0":0,"limitIteration1":0,"iterationLimitOrder":"
before","showLastEventsInIteration2":false,"includeIndexNullInIteration0":false}]]}
```

Figure 5.14: Sharing feature

This modal can be opened through the share icon shown at the top of the navbar. Here, we can import a configuration or click the export button to generate the JSON of the current view. Then it is possible to select the text or click directly the button "copy to clipboard" to copy the JSON into the clipboard of the operating system.

This feature was possible thanks to the local storage function. In fact, the application is made to store the the view into the local storage automatically at every change of any timeline.

The local storage has a certain importance when an analysis takes much time. For instance, if we must generate multiple timelines in different interval of times or with different filters, then it could take a lot. So, we can stop there and then reach the page in another moment to get the same old view. This is possible by using the local storage of the browser which is permanent if the user does not reset it.

5.2.7 Range selector and data distribution

As it is explained before, the timeline has to merge different sequences together in order to show a summary of all of them. This means, that we can get a variable timeline length on basis of the length of each sequence. However, it may not be clear how actually users are distributed in the dataset, so the number of sequences with respect to their length is visible through the charts of the data distribution. An example is shown in figure 5.15.

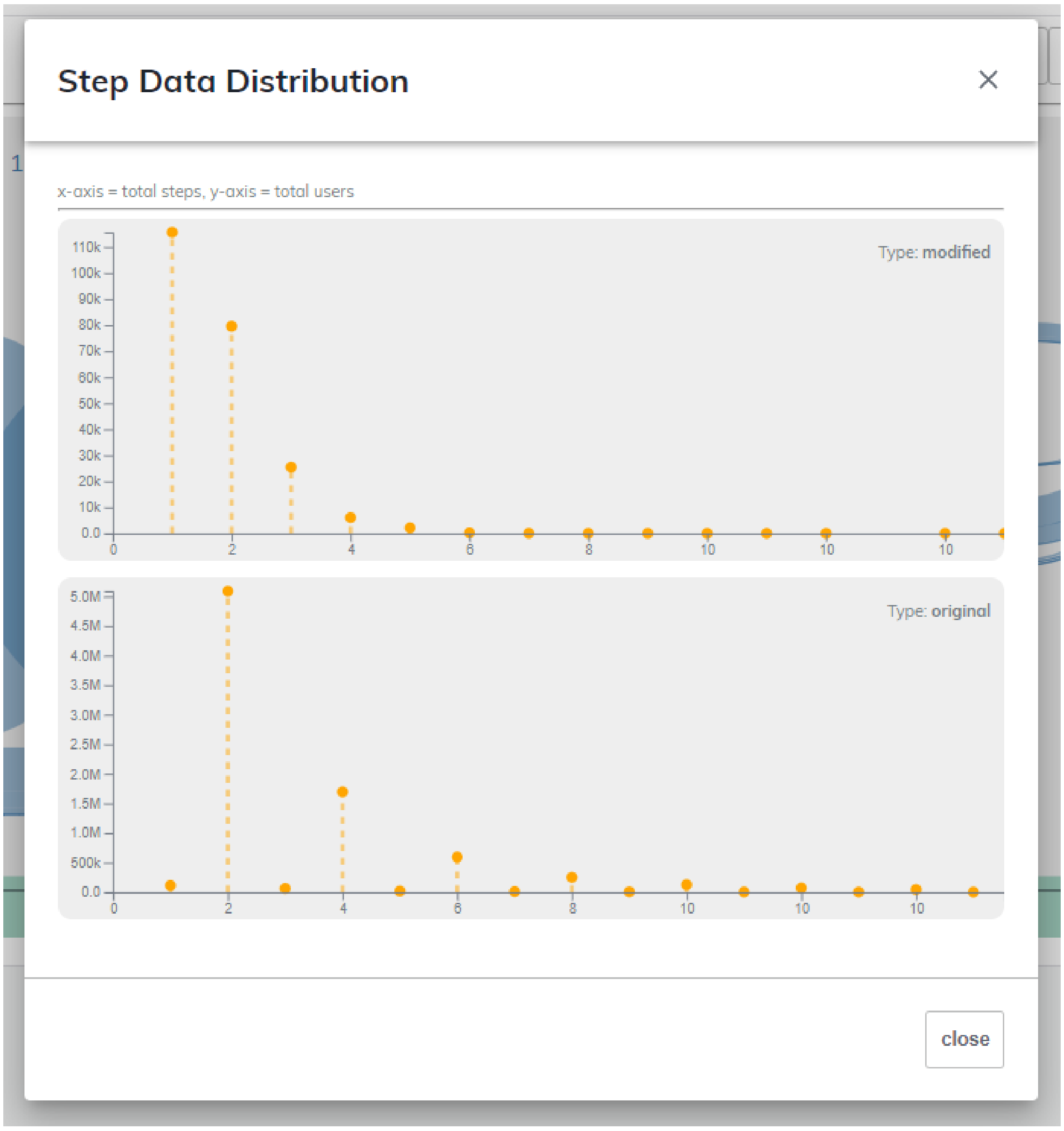


Figure 5.15: Example of data distribution

On the x-axis there is the total number of steps, instead on the y-axis the total number of users who did that number of steps. If we consider, for instance, the point x=5 and y=100, then it means that 100 users finished triggering new events after 5 steps.

As it is possible to see, there are two charts showed through this modal, one is called *original* while the other one is called *modified.* they are two and not just one because it is possible to consider only users who did a specific number of total steps defined in an interval of the option **restrict range of data**. This is can be done *before* or *after* the filtering operations on basis of our needs. For example, we may be interested users who did in total a number of steps between 1 and 5.

So, the *original* chart shows the data distribution of the original dataset after applying just the base filters which are interval of dates and type of event, while the one with the name *modified* shows the new data distribution after applying all the timeline options. The original chart is useful to remember how the data were distributed initially and work on basis on them.

5.2.8 Diff function

Another useful function is the **diff function** which allows comparing two timelines configured in the current page. It is very powerful and useful especially in the analytics context. Indeed, it is possible to use the timeline for comparing an A/B testing.

This kind of test allows to test a change on a specific platform or service. For instance, let's suppose a change on a banner design is made. Before releasing this change all over the world, it is possible to release it partially, or better release it to a 50% of population. In this, way we can gather information about the new change and old change in the same interval of time. Next, we just need to compare the results through some tools.

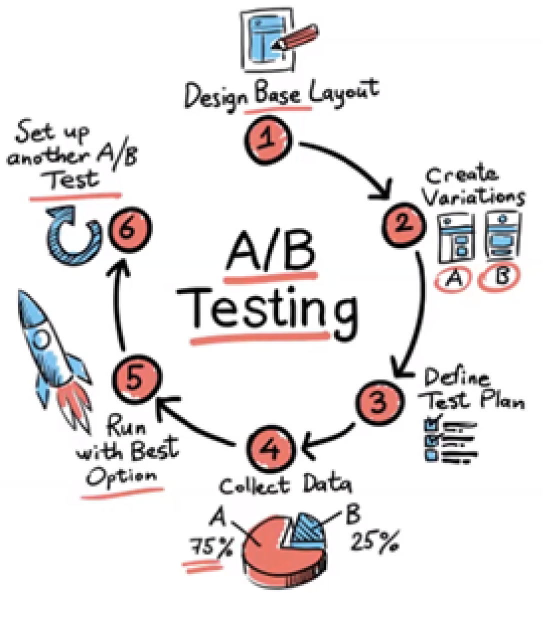
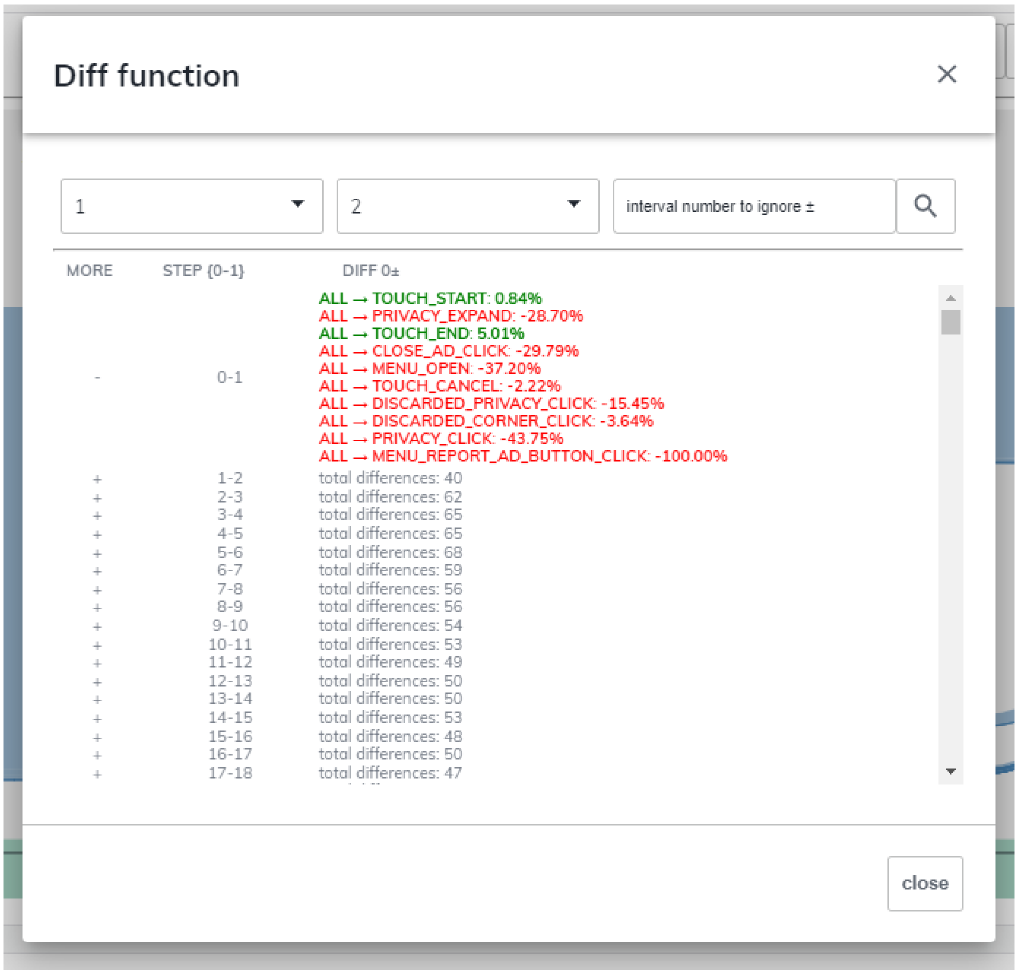


Figure 5.16: A/B testing[3]

Here is an example of comparison through the application made:



Diff function

1 | 2 | interval number to ignore ±

MORE	STEP {0-1}	DIFF 0±
-	0-1	ALL → TOUCH_START: 0.84% ALL → PRIVACY_EXPAND: -28.70% ALL → TOUCH_END: 5.01% ALL → CLOSE_AD_CLICK: -29.79% ALL → MENU_OPEN: -37.20% ALL → TOUCH_CANCEL: -2.22% ALL → DISCARDED_PRIVACY_CLICK: -15.45% ALL → DISCARDED_CORNER_CLICK: -3.64% ALL → PRIVACY_CLICK: -43.75% ALL → MENU_REPORT_AD_BUTTON_CLICK: -100.00%
+	1-2	total differences: 40
+	2-3	total differences: 62
+	3-4	total differences: 65
+	4-5	total differences: 65
+	5-6	total differences: 68
+	6-7	total differences: 59
+	7-8	total differences: 56
+	8-9	total differences: 56
+	9-10	total differences: 54
+	10-11	total differences: 53
+	11-12	total differences: 49
+	12-13	total differences: 50
+	13-14	total differences: 50
+	14-15	total differences: 53
+	15-16	total differences: 48
+	16-17	total differences: 50
+	17-18	total differences: 47

close

Figure 5.17: Diff function

This comparison is done step by step and computing the percentage variation expressed by the formula:

$$(\frac{x_f - x_i}{x_i} * 100)\%$$

Through this formula, it is possible to know if there was an increase or decrease in a specific transition of events between two steps. There is also a way to filter this differences and show only those that satisfy the parameters. Another good thing is that this function depends on the configuration of the timelines. So, it means we can compare, for example, only the final events by setting on the option to show the final events in step 2.

Chapter 6

Conclusions

The timeline has been developed with the goal to give the maximum flexibility on the type of analysis to any user. A lot of different features allow managing almost every aspect of data shown giving a way to include or exclude part of it. Not less important, much information is visualized on user interface to give many more details and perform a better analysis. In conclusion, I would like to propose some improvements and new additions for the application.

6.1 What's next

The current state of the application contains the most important features to be usable. However, it can further improve with new functions and use cases that may extend its limit.

For example, it could be useful having a new filter that allows to filter by banner. In fact, the application is available to show timelines of any type, but everyone includes all the banners. It means, that in case we would like to try to analyze a specific banner, this is not possible. However, it should not be a problem because it is usually needed to perform a general analysis on all the banners since they have the same configuration in common.

Actually, this change was in program, but it was not possible because there was a on-going internal change on the banner management. Add this feature before this change would cause another change to adapt the old one with the new data structure. Anyway, the software was developed with the goal to be modular. So, adding a new filter requires very few changes. In particular, it is needed to add a new column which describe the banner id on the data export process, and code a bit the back-end/front-end to know the presence of a new filter. No other modification is needed, the rest is done automatically which means there is no need to add new endpoints, select boxes or new API functions.

Another element that could be relevant is to make this new powerful tool to the standard user. Currently, the customers of the company have to access to all base tools of relook to make their dashboards, but they do not have access to this application since it is available only for internal use. Indeed, there are different reason on it. First, this software is quite advanced to use to gather information, so only expert of the sector could use all its functions. Secondly, it is released in pre-production, in other words it is an area that only internal employees can access. So, the next step is to move it in production and make it available to all external customers.

Glossary

bare-metal A bare-metal server is a computer server that hosts one tenant, or consumer, only. The term is used for distinguishing between servers that can host multiple tenants and which utilize virtualisation and cloud hosting. Such servers are used by a single consumer and are not shared between consumers. 2

DBMS A Database Management System (DBMS) is software designed to store, retrieve, define, and manage data in a database. 14, 15

DOM The Document Object Model is a cross-platform and language-independent interface that treats an XML or HTML document as a tree structure wherein each node is an object representing a part of the document. The DOM represents a document with a logical tree. 20, 58

git Git is software for tracking changes in any set of files, usually used for coordinating work among programmers collaboratively developing source code during software development. Its goals include speed, data integrity, and support for distributed, non-linear workflows. 10, 13

Gradle Gradle is a build automation tool for multi-language software development. It controls the development process in the tasks of compilation and packaging to testing, deployment, and publishing. Supported languages include Java, C/C++, JavaScript. 10, 13

Jenkins The leading open source automation server, Jenkins provides hundreds of plugins to support building, deploying and automating any project. 11, 13

MPP An MPP Database (short for massively parallel processing) is a storage structure designed to handle multiple operations simultaneously by several processing units. In this type of data warehouse architecture, each processing unit works independently with its own operating system and dedicated memory. 15

MVC The Model-View-Controller (MVC) is an architectural pattern that separates an application into three main logical components: the model, the view, and the controller. Each of these components are built to handle specific development aspects of an application. 51

Query string A query string is a part of a uniform resource locator (URL) that assigns values to specified parameters. A query string commonly includes fields added to a base URL by a Web browser or other client application, for example as part of an HTML form. 52

RDBMS An RDBMS is a type of database management system (DBMS) that stores data in a row-based table structure which connects related data elements. An RDBMS includes functions that maintain the security, accuracy, integrity and consistency of the data. This is different than the file storage used in a DBMS. 15

React React is an open-source, front end, JavaScript library for building user interfaces or UI components. It is maintained by Facebook and a community of individual developers and companies. React can be used as a base in the development of single-page or mobile applications. 16, 20

Bibliography

[1] Dr. Alex Ferworn A.K.M Zahidul Islam. «A Comparison between Agile and Traditional Software Development Methodologies». In: *Global Journal of Computer Science and Technology* (2020). ISSN: 0975-4172. DOI: 10.34257/GJCSTCVOL20IS2PG7. URL: https://computerresearch.org/index.php/computer/article/view/1987.

[2] *Chrome - developer tools*. URL: https://developer.chrome.com/docs/devtools/.

[3] *Criteo - A/B testing.*

[4] *Criteo - ads system.*

[5] *Criteo - AI Engine.*

[6] *Criteo - company*. URL: https://www.criteo.com/fr/company/.

[7] *Criteo - Development Workflow.*

[8] *Criteo - heatmaps on clicks.*

[9] *Criteo - new infrastructure*. URL: https://criteo.com/news/press-releases/2020/10/criteo-launches-latest-green-energy-based-data-center-in-japan/.

[10] *Criteo - offices*. URL: https://www.criteo.com/contact-us/find-us/.

[11] *D3JS visualization examples*. URL: https://observablehq.com/@d3/gallery.

[12] *FlatSpec*. URL: https://www.scalatest.org/user_guide/selecting_a_style.

[13] *Gerrit - code review*. URL: https://www.gerritcodereview.com/.

[14] *Google - Path Analysis*. URL: https://support.google.com/analytics/answer/9317498?hl=en.

[15] *Google - Path Analysis Example*. URL: https://www.practicalecommerce.com/wp-content/uploads/2019/05/ClickPaths_1.jpg.

[16] *JetBrains - Intellij IDEA*. URL: https://www.jetbrains.com/idea/.

[17] *Jira - software*. URL: `https://www.atlassian.com/software/jira`.

[18] *Matchers*. URL: `https://www.scalatest.org/user_guide/using_matchers`.

[19] *Mozilla - webcomponents*. URL: `https://developer.mozilla.org/en-US/docs/Web/Web_Components`.

[20] *React*. URL: `https://reactjs.org/`.

[21] *React - Ref*. URL: `https://reactjs.org/docs/refs-and-the-dom.html`.

[22] *ScalaTest*. URL: `https://www.scalatest.org/`.

[23] *TypeScript benefits*. URL: `https://medium.com/swlh/the-major-benefits-of-using-typescript-aa8553f5e2ed`.

[24] *URL length limit on the browser*. URL: `https://www.geeksforgeeks.org/maximum-length-of-a-url-in-different-browsers/`.

[25] *Vertica - storage approach*. URL: `https://www.vertica.com/secrets-behind-verticas-performance/`.

[26] David Young. «Software Development Methodologies». In: *White paper* (Aug. 2013).