UNIVERSITY OF TARTU
Faculty of Science and Technology
Institute of Computer Science
Computer Science Curriculum

Aleksei Kopõlov

ProLift: A Web Application to Discover Causal Treatment Rules From Business Process Event Logs

Master's Thesis (30 ECTS)

Supervisor(s): Marlon Dumas, PhD

Tartu 2022

Acknowledgements

I would like to express my deepest gratitude to my supervisor Professor Marlon Gerardo Dumas Menjivar who provided enormous support and guidance during the writing of this thesis.

I am also thankful to my family and friends for being supportive throughout my university studies.

Lastly, I'd like to express a special appreciation to my close friends Tarun Khajuria and Anna Vissart for their immense support and motivation.

ProLift: A Web Application to Discover Causal Treatment Rules From Business Process Event Logs

Abstract:

Causal process mining is a sub-field of process mining belonging to a family of techniques related to the field of business process management(BPM). The main goal of causal process mining is to utilize real process execution logs and causal machine learning techniques in order to discover, analyze and improve business processes. In this Master's Thesis, we provide a detailed overview of a web-based application and all of its components, developed by the thesis author. The main goal of the application is to utilize the latest discoveries in causal process mining techniques that are capable of discovering and quantifying cause-effect relations. The additional goal of the application is to provide end users with a responsive and user-friendly interface, which allows them to discover treatment rules from a business process event logs and to display these treatment rules in an easy-to-understand manner.

The Web application outlined in the thesis implements an approach to discover causal treatment rules proposed by Bozorghi et al. This approach uses uplift trees to discover rules that relate a treatment (e.g. giving a phone call to a user) with an increased probability that a positive outcome will be achieved (e.g. the customer will be satisfied with the service they receive).

Keywords:
Business process management, causal process mining, causal machine learning, uplift trees, process analytics tool.

CERCS: P170 - Computer Science, Numerical Analysis, Systems, Control.

ProLift: Veebirakendus põhjusliku ravi reeglite leidmiseks äriprotsesside sündmuste logidest

Lühikokkuvõte:
Casual Process Mining - ingl '(jooksva protsessi andmete kogumine)' on Process Mining'u alamharu, ning kuulub Business Process Management (BPM) - ingl '(Äriprotsessi-de juhtimine)' tööriistade hulka. Tema peamine eesmärk on koguda jooksvate protsesside andmeid logide näol selleks, et hiljem määratleda, analüüsida ja edendada äriprotsesse. Selles magistritöös on detailne ülevaade autori poolt arendatud veebirakenduse ja kõigi selle komponentide kohta. Antud veebirakenduse peamine eesmärk on määratleda ja hinnata põhjuste ja tagajärgede seoseid kasutades Casual Process Mining'u uusimaid avastatud võtteid. Veebirakenduse teine eesmärk on pakkuda reageeriv ja kasutajasõbralik liides, mida kasutades on võimalik avastada ja esitada äriprotsessi juhtumite logide analüüsidest leitud mutatsiooni soovitused kergesti arusaadaval viisil.

Magistritöös esitatud veebirakendus implementeerib Bozorghi et al. ettepandud meetodi protsesside muteerimiseks. See meetod kasutab uplift trees - ingl '(masinõpes kasutatav ennustav modelleerimise viis)' mudeleid, et avastada positiivseid tagajärgi (nt. kasutaja on rahul temale antud teeninduse üle) esiletoovate mutatsioonide (nt. kasutajale helistamine) kriteeriume.

Võtmesõnad:
Äriprotsesside juhtimine, põhjuslik protsess kaevandamine, põhjuslik masinõpe, protsessianalüütika tööriist.

CERCS: P170 - Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine (automaatjuhtimisteooria).

Contents

1 Introduction

As organizations became heavily digitized, the automation and optimization of business processes started taking on a more vital role in the date-to-date operations of most organizations. On one hand, business process automation brought in a plethora of benefits such as reduction of operational cost, reduction of order fulfillment time, and personalized client interaction. On the other hand, it made the competition for clients and their approval a more difficult playing field.

Currently, when a large organization wants to roll out a sales campaign or implement changes in the client interaction processes, the decisions have to be consolidated either with personnel that has appropriate domain knowledge or with the data science and analytics department. More often than not, the decision falls on the latter group. This in turn results in an approval chain bottleneck. In order to alleviate this bottleneck, there has been a rise in software tools that focus on making some data analytics processes much simpler. It should be mentioned that those types of software are not designed to completely replace data science and analytics departments, but are merely there to alleviate some pressure.

The primary goal of our thesis was the development of an application that would adapt the latest methods and techniques in causal process mining. The main goals of the application are as followed.

First, the application must provide the end users with a user-friendly interface, where the end-user must be able to submit files for processing and provide the necessary configuration. Secondly, the application must be responsive and provide users with feedback regarding the status of the rule extraction process. Thirdly, the application must implement the latest techniques in causal process mining, as laid out in the following paper [CHL+20]. Lastly, the application must provide process mining results in a simple, human-readable, "if-then" format.

The thesis is organized as followed. In chapter 2 we provide a short introduction to the concept of BPM and uplift trees. In chapter 3 we provide an overview of our application architecture and its components. In chapter 4 we present a detailed walkthrough of our application user interface. In chapter 5 we explain the structure of our entities and their relationship to one another. In chapter 6 we describe our application's deployment process and live configuration. Lastly, in chapter 7 we summarize our results and give a proposition for future work.

2 Background

In this chapter, we will give a brief introduction to Business Process Management and Causal Machine Learning.

2.1 Business Process Management

Business Process Management (BPM) is a discipline that focuses on utilizing various methods to discover, model, analyze and measure different types of business processes [Pan12]. Even though BPM methods can vary from organization to organization, the general end goal of BPM is the improvement of business processes. The concept of improving business processes can take on multiple forms, the most common of which are: reduction of operational costs, reduction of execution time, reduction of error rates, and the gaining of competitive advantages [DRMR18].

2.1.1 Business Process

A business process is a set of events, activities, and decisions, the fulfillment of which leads to the completion of supportive, operational, or managerial tasks [DRMR18]. In simple terms, business process defines a list of steps needed to be taken in order to complete a certain task. The type of processes present in an organization can vary from organization to organization, however, typical examples of processes that can be found in most organizations include: [DRMR18]

- Order-to-cash: A process that is initiated when a customer submits a purchase order and ends when the product or service has been delivered to the customer.
- Issue-to-resolution: A process that starts when a customer submits a complaint related to a product or a service and ends when the customer and provider agree that the issue has been resolved.
- Application-to-approval: A process that starts when an individual or a company submits a formal request for receiving benefits or privileges to an organization and ends when the privileges or benefits in question have been granted or denied.

2.1.2 Process Mining

Process mining is an intersection of data science and BPM that focuses on utilizing event data and data processing techniques in order to acquire novel insights that can be then used in optimizing operational processes [vdA22]. Process mining techniques can be divided into two categories: backward-looking and forward-looking.

- Backward-looking - is a type of process mining technique that focuses on discovering bottlenecks in operational processes.
- Forward-looking - is a type of process mining technique that focuses on providing recommendations that would lower the failure rate of the currently running cases.

2.1.3 Causal Process Mining

Causal process mining is an emerging sub-field of process mining that seeks to develop methods to discover and quantify cause-effect relations by analyzing event logs [Dum]. The general difference between classical and causal process mining has to do with the type of process mining techniques that are being utilized. While classical processing mining techniques allow us to discover correlation relations in causal process mining, we are aiming to discover causal relations.

2.1.4 Event Logs

Event Data is data that has been extracted from organization information systems, used to support the processes that need to be analyzed [vdA22]. As an example, event data could be extracted from a Customer Relationship Management system (CRM), Enterprise Resource Planning system (ERP), or Supply Chain Management system(SCM).

Since event data often originates from multiple sources, we might run into issues where data is incomplete, duplicated, inconsistent, or generally of poor quality. In order to address such problem, there are a variety of techniques and approaches that are used to properly clean and structure data. After the data has been properly cleaned and structured, we are left with a data set that is commonly referred to as an event log.

Definition (event log) - An event log is a tuple $L = (E, \#, \prec)$ consisting of a set of events $E \subseteq U_{ev}$, a mapping $\# \in E \rightarrow U_{map}$, and a strict partial ordering $\prec \subseteq E \times E$ on events [vdA22]. In simple terms, an event log is a collection of events corresponding to the process that we are interested in analyzing.

2.2 Causal Machine Learning

Causal Machine Learning (Causal ML) is an umbrella term for machine learning methods that formalize the data-generation process as a structural causal model (SCM) [KLL$^+$22]. Causal ML is a large sub-field consisting of multiple methods and techniques used in modeling and measuring causality. Nevertheless, within the scope of our thesis, we will only focus on utilizing tree-based algorithms for evaluating causal inference.

2.2.1 Uplift Trees

Uplift trees are a type of causal machine learning models that are designed to predict the net effect of performing some action, such as sending a promotional email, on a certain outcome, such as a client buy a product [DMV18]. Based on literature research, it seems to be that currently most of the development in uplift trees originates from fields related to customer interaction, such as: sales, marketing and client support.

Uplift Definition [DMV18]. Let's assume that we have a customer base data set that we randomly divide into two groups, defined as treatment and control. As a formal definition, let X be a vector of controllable or uncontrollable variables, $X = \{x_1, ..., x_n\}$, and let Y be the target outcome, $Y \in \{0, 1\}$, with $Y = 1$ indicating a positive outcome and $Y = 0$ indicate a negative outcome. In addition, variable T denotes whether or not a customer is in the treatment group, $T = 1$, or in the control group, $T = 0$. Finally, P denotes a probability as estimated by a model. Uplift is then defined as:

$$U(x_i) := P(y_i = 1|x_i; t_i = 1) - P(y_i = 1|x_i; t_i = 0) \tag{1}$$

Tree-Based Uplift [CHL$^+$20] or uplift tree approach consists of a set of methods that use a tree-based algorithm where the splitting criterion is based on differences in uplift:

$$D_{gain} = D_{after_{split}}(P^T(Y) : P^C(Y)) - D_{before_{split}}(P^T(Y) : P^C(Y)) \tag{2}$$

Where D measures the divergence and $P^T(Y)$ and $P^C(Y)$ are the probability distributions of the outcome in the treatment and control groups. The three common ways to quantify divergence are: Kullback-Leibler(3), Euclidean Distance(4) and Chi(5).

$$KL(T : C) = \sum_{k=left,right} p_k \log \frac{p_k}{q_k} \tag{3}$$

$$ED(T : C) = \sum_{k=left,right} (p_k - q_k)^2 \tag{4}$$

$$\chi^2(T : C) = \sum_{k=left,right} \frac{(p_k - q_k)^2}{q_k} \tag{5}$$

Where p is the sample mean of the treatment group, q is the sample mean of the control group and k indicates the leaf in which p and q are computed [CHL$^+$a].

2.2.2 CausalML Library

In the scope of our application, for building uplift trees, we've deiced to utilize a modern Python library by the name of CausalML.

CausalML is a Python causal machine learning library, which was developed and is still actively maintained by the Uber development team [CHL+a, CHL+b]. The library itself consists of several uplift models and causal inference methods, designed on the basis of machine learning algorithms laid out in the following [CHL+20] research paper. Generally speaking, the library gives users a set of methods that can be used in estimating the causal impact of intervention I on outcome O for cases with observed features F. Currently, the library supports the following list of algorithms:

- Tree-based algorithms.
- Meta-learner algorithms.
- Instrumental variables algorithms.
- Neural-network-based algorithms.

In the scope of our application, we are planning to implement only the tree-based algorithms, since they allow us to extract rules that can be interpreted by business users.

3 Architecture

In this chapter, we will give an overview of the application architecture and its components. We will start by providing a general overview, then we will move on to the explanation of each architecture tier, the technologies used within that tier, and the reasons for their selection. Finally, we will finish this chapter by providing a quick summary.

3.1 Architecture Overview

As with many web applications, ProLift was designed using a well-established software architecture known as the Three-Tier Architecture. The main concept behind the Three-Tier Architecture is the separation of application into three logical and physical tiers known as: the presentation tier, the application tier, and the data tier.

Before moving on to the explanation of each tier and the technologies used within that tier, we have provided a figure [1], illustrating ProLift architecture from a high-level perspective.

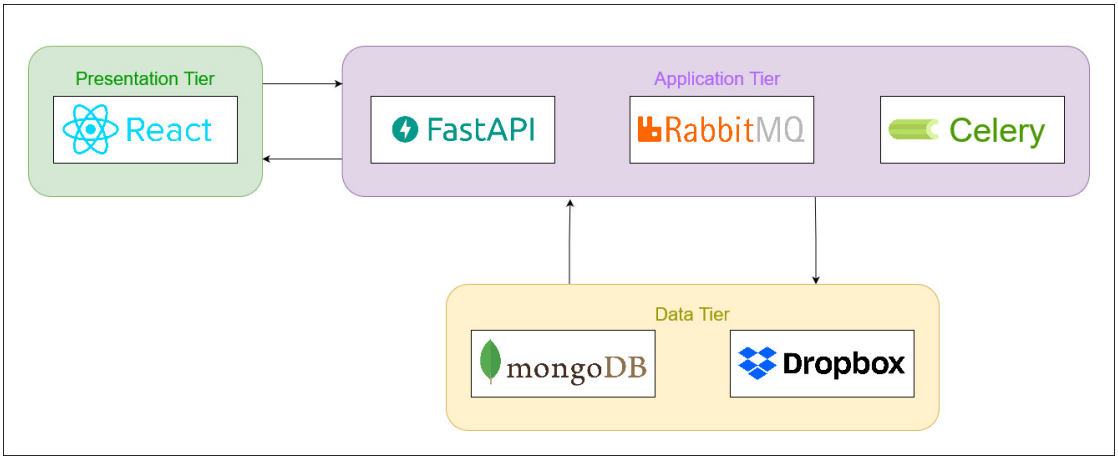

Figure 1. ProLift architecture from a high-level perspective.

3.2 Presentation Tier

The presentation tier or also known as the user interface tier is the main point of interaction between the end users and the application. Though the structure of the presentation tier can vary depending on the requirements, the general purpose of this tier stays the same for most of the time, and that is to:

- Display information to the end user.
- Collect information from the end user.

In the scope of our application, the presentation tier has been implemented through a web interface with the help of a very popular JavaScript library known as React.

React also known as ReactJS or React.js, is an open-source, JavaScript library for building user interfaces, developed by Meta (formerly known as Facebook) and is now maintained by both Meta employees and React community members [Incc, Incd].

Since the designing and implementation process of an intuitive and user-friendly interface can be a fairly arduous one, we have decided to dedicate a separate chapter [4] where we will be going into greater details.

However, one important point should be addressed before we move on, and that is the reason for choosing React. It is to no surprise that there exist multiple libraries and frameworks that are designed to build powerful front-end applications such as: Angular, Vue, and the aforementioned React, and even though all of them are designed to solve a similar problem the way they address it, is different. Angular, in comparison to React and Vue, is a much more powerful and complex tool, and considering that we don't have any interest in building an enterprise-level user interface, there was no reason to choose Angular over React or Vue. This meant that in the end, the choice came between Vue and React, and even though Vue is a much simpler library, React has a much larger community and a much richer package ecosystem, which meant that in the case of React we were less likely to encounter unknown issues and more likely to find packages of pre-built components, that would speed up our developing process.

3.3 Application Tier

The application tier or also known as the logic/middle tier is the core of the application. In most applications, this tier is the most critical and complex, as it is responsible for executing the business logic of the whole application. Generally speaking, the main purpose of this tier is to:

- Process incoming data or commands.
- Make logical decisions and execute processes.
- Provide a way for moving data between the presentation tier and the data tier

In the scope of our application, the application tier is implemented with the help of 3 separate technologies: FastAPI, RabbitMQ, and Celery.

FastAPI is an open-source, Python-based, web framework for building application programming interfaces or APIs for short [Rama, Ramb].

Within the scope of our application, this framework will be used in the construction of what is commonly referred to as a RESTful Integration Service. RESTful - meaning that the service uses representational state transfer (or REST for short) as a means of communication between different applications, and Integration - meaning that the service's primary goal is to facilitate communication between several decoupled components. In our case, the RESTful integration service will function as the main entry point of all requests coming in from the user interface.

It should be mentioned that FastAPI is not the only framework on the market and that there are a plethora of frameworks that could have easily substituted FastAPI, such as: NodeJS, Django, Spring Boot, Flask, etc. We've chosen FastAPI for two reasons: one, we didn't want to use too many programming languages, which discarded all none Python-based frameworks, and two, we wanted to build a simple yet fast REST service, which made FastAPI the best option in comparison to Flask or Django.

One of our nonfunctional requirements was to make sure that the application could handle peaks in demand without leading to a server crash, and in a way that the performance degrades gracefully in case of heavy traffic. This was made possible by implementing an asynchronous interaction style using message queues and a master-worker architecture. In our application, the role of the master is preformed by RabbitMQ and the role of the worker by Celery.

Celery is a Python-based, open-source task queue software [SKa, SKb], the main goal of which is to asynchronously execute processes in the background.

Within our application, we have a set of functionalities that are computationally intensive, such as: processing a large chunk of data or training a model. Such processes can not be executed directly in the integration service as they will end up blocking it, which is where Celery comes into play. Celery allows our application to asynchronously execute computationally intensive processes into a different thread or even a different machine.

Celery is not the only task queue software, but it is the most reliable and most commonly used in Python-based applications.

RabbitMQ is an open-source message queuing software also known as a message-broker [Incf], the general purpose of which is to enable different applications, and services to communicate with one another and to exchange information.

In the scope of our application, RabbitMQ fulfills two important roles. First and foremost of all, RabbitMQ facilitates asynchronous communication between our RESTful service and our Celery workers. Secondly, RabbitMQ allows us to distribute tasks

between multiple Celery workers and to easily scale our application if there will be ever a need for that.

Below, we have provided a figure 2 illustrating the process of communication within the application tier.

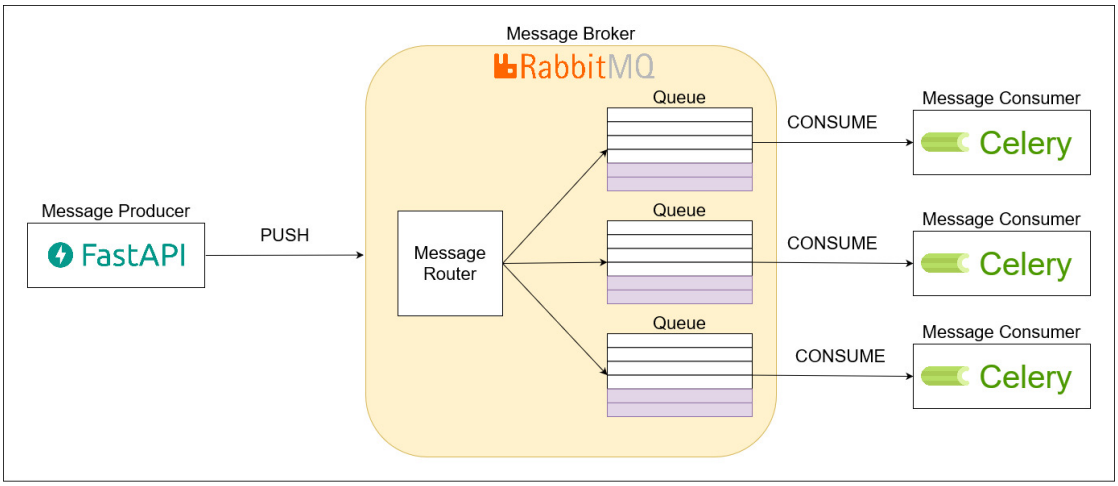

Figure 2. Communication within the application tier.

3.4 Data Tier

The data tier, or also known as the database tier, is responsible for storing and managing all the necessary data, either in the form of a database or in the form of a file repository.

In the scope of our application, the data tier has been implemented through 2 technologies: MongoDB and Dropbox.

MongoDB is an open source, non-relational database management system (also commonly referred to as a NoSQL database) that uses flexible documents instead of tables and rows to process and store various forms of data [Edu].

In our application, MongoDB fulfills the role of the central database, which means that it stores nearly all the necessary data and metadata which have been either generated by the application tier or provided to the application tire.

The process of choosing a proper database more often than not relies on the core requirements of the application itself. In our case, we decided to go with MongoDB, a non-relational document-oriented database. The reasons for choosing MongoDB stem

primarily from the structure of our data or to be more specifically the absence of it. Since we were developing a prototype, we were not sure about the structure of the data that we would end up storing. This meant that during the development process, our data sets could have undergone multiple changes and reworks (which it has). In situations like this, it is always advisable to work with a NoSQL database as they are much more agile than for example a traditional relational database.

Dropbox is a file hosting service [Incb]. Since our application has to process submitted files, we need to have a place to store them, that is where Dropbox comes into play.

There are multiple different file hosting services and most of them are pretty much interchangeable, which is why the choice often comes to personal preferences and the price of service.

3.5 Architecture Summery

Before moving on to the next chapter, we will provide a table 1 with a quick summary of all technologies that have been implemented in our application and their purpose within the scope of our application.

Table 1. List of used technologies and their role.

Technology	**Role**	**Purpose**
React	User Interface	Main point of interaction with the end user
FastAPI	REST API Service	Facilitate communication between user interface and the rest of the application
RabbitMQ	Message Broker	Facilitate asynchronous communication between FastAPI and Celery worker
Celery	Asynchronous Worker	Perform computationally intensive operations
MongoDB	Main Database	Store and proved all the necessary data to the application
Dropbox	File Repository	Store files

4 User Interface and Flow

One of the main goals of our application is to provide a simple yet initiative and responsive User Interface(UI). In this chapter, we will provide an overview of our application UI. We will start by describing the steps that we took in the process of designing our UI, then we will move on to describe the structure of our UI from the developer's perspective, and lastly, we provide a walk-through of our UI from the end user's perspective.

4.1 UI Design

The process of developing a UI can be fairly expensive and arduous. Moreover, considering that UI is the point from which all the end users will interact with our application, a lackluster implementation can result in poor acceptance and adaptation from potential clients. This is why it is in our own interest to make the UI as intuitive and as user-friendly as it is possible.

Considering the importance of a good UI, it is no surprise that nearly all large tech companies have designated teams, whose primary goal is to focus on designing and developing high-quality UIs. Even though we may not have access to the same amount of resources as a large tech company, it would be very unreasonable for us not to adopt at least some general best practices in interface designing.

In the process of developing our interface, we have decided to implement 2 best practices that are primarily used to give developers a good general overview of the interface before the beginning of the development process, therefore mitigating some unnecessary reworks. The 2 best practices in question are the construction of a User Flow and a Wireframe.

4.1.1 User Flow

User Flow is a type of diagram, that is designed to give an overview of what an end user has to do in order to complete a certain task. Such diagrams can help us with multiple problems: from understanding as to how our interface should be structured and what information we should provide to our end users, to discovering redundant or unnecessary steps within our end user journey.

Below, we have provided 3 user flow diagrams (Figure 3, 4 and 5). Figure 3 illustrates a set of steps necessary to create a Project entity, figure 4 illustrates a set of steps necessary to create a Model Configuration entity and lastly figure 5 illustrates a set of steps necessary to extract treatment rule outcomes. In chapter 5 we will go into greater details as to what exactly is a Project entity, Model Configuration entity, and their structure. However, for now, when referring to a Project entity, think of it as an entity that is used to bind a file (provided by the end-user) with the configurations necessary to process that file, and when referring to a Model Configuration entity, think of it as

an entity that is used to bind data (extracted from the file) with a model (that is used to extract treatment rules outcome).

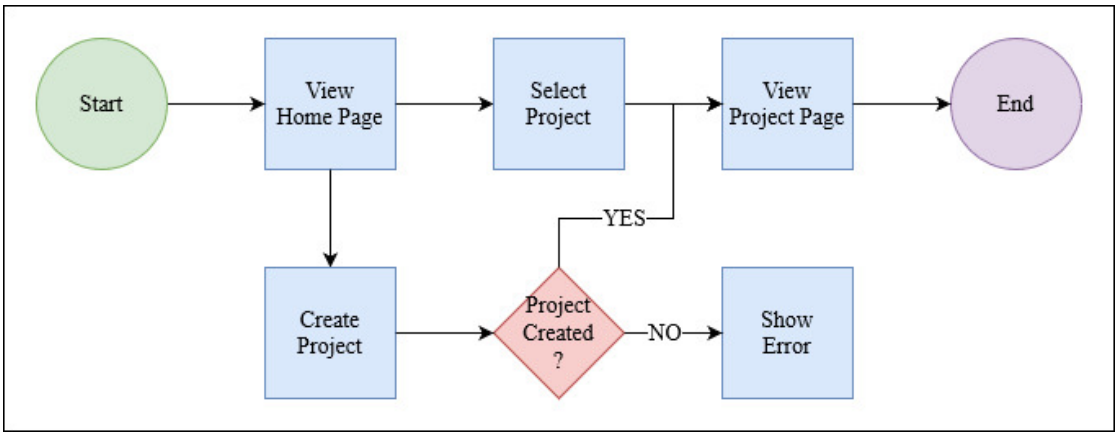

Figure 3. Project entity creation flow.

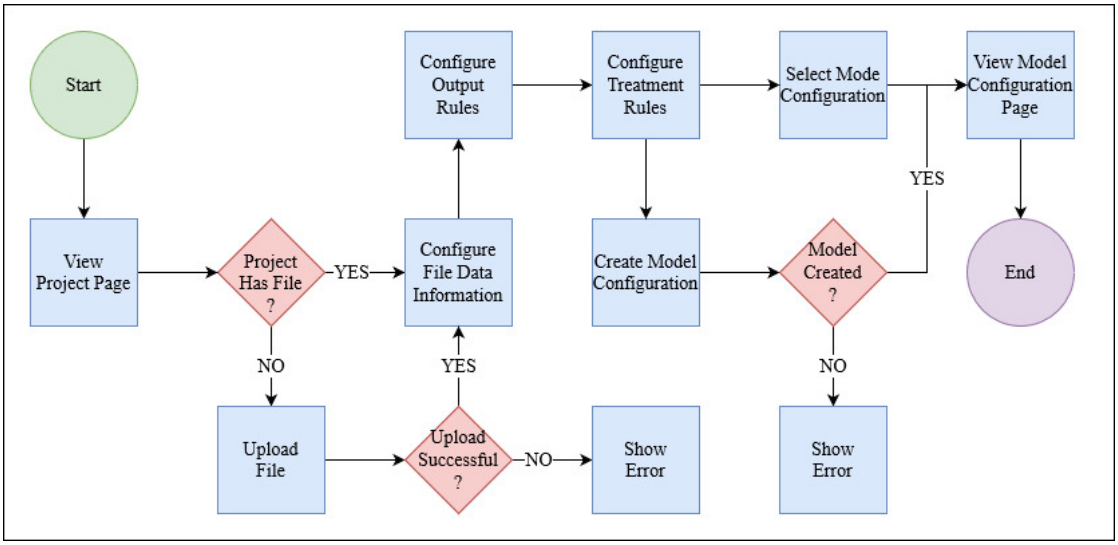

Figure 4. Model Configuration entity creation flow.

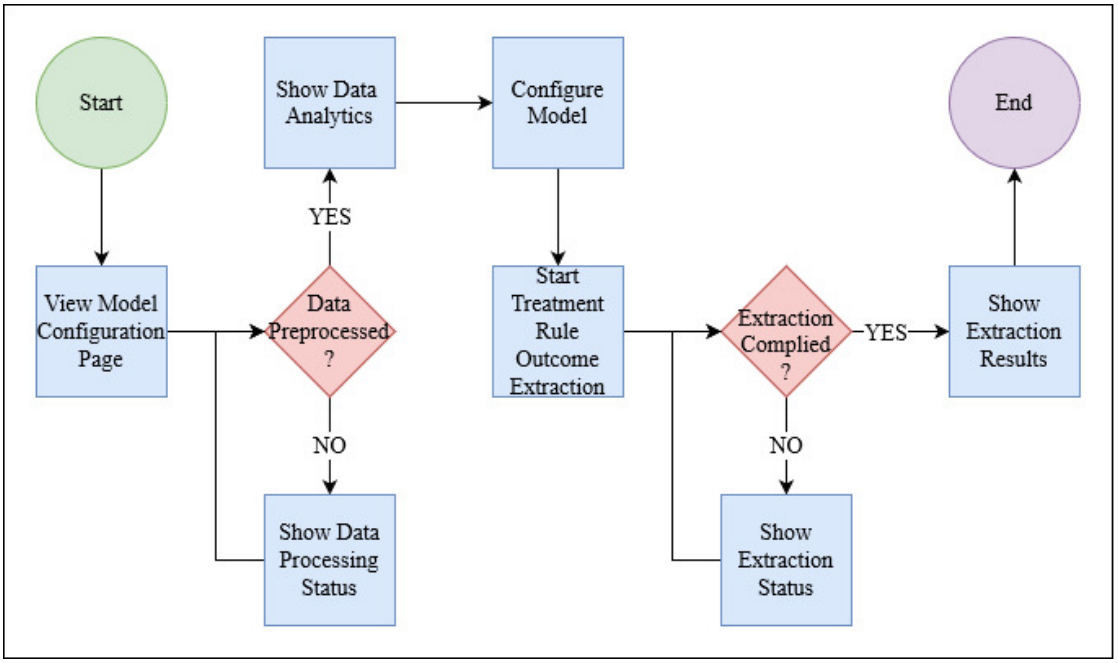

Figure 5. Treatment rule outcome extraction flow.

4.1.2 Wireframe

Wireframe is a simple two-dimensional scheme, that is used primarily to determine the position of your major elements within a page. When creating a wireframe, it is important to remember that a wireframe should not contain any styling, formatting, or images. The main attraction of a wireframe is its simplicity, which allows the creation of very complex schemes within a matter of hours. Despite its simplicity, wireframe is also incredibly beneficial, as it gives a good overview of the interface before the developing process, which in turn mitigates the chances of unnecessary and costly reworks.

Below, we have provided an image (Figure 6) depicting the initial schema that was used as a reference guide for developing all the interface pages.

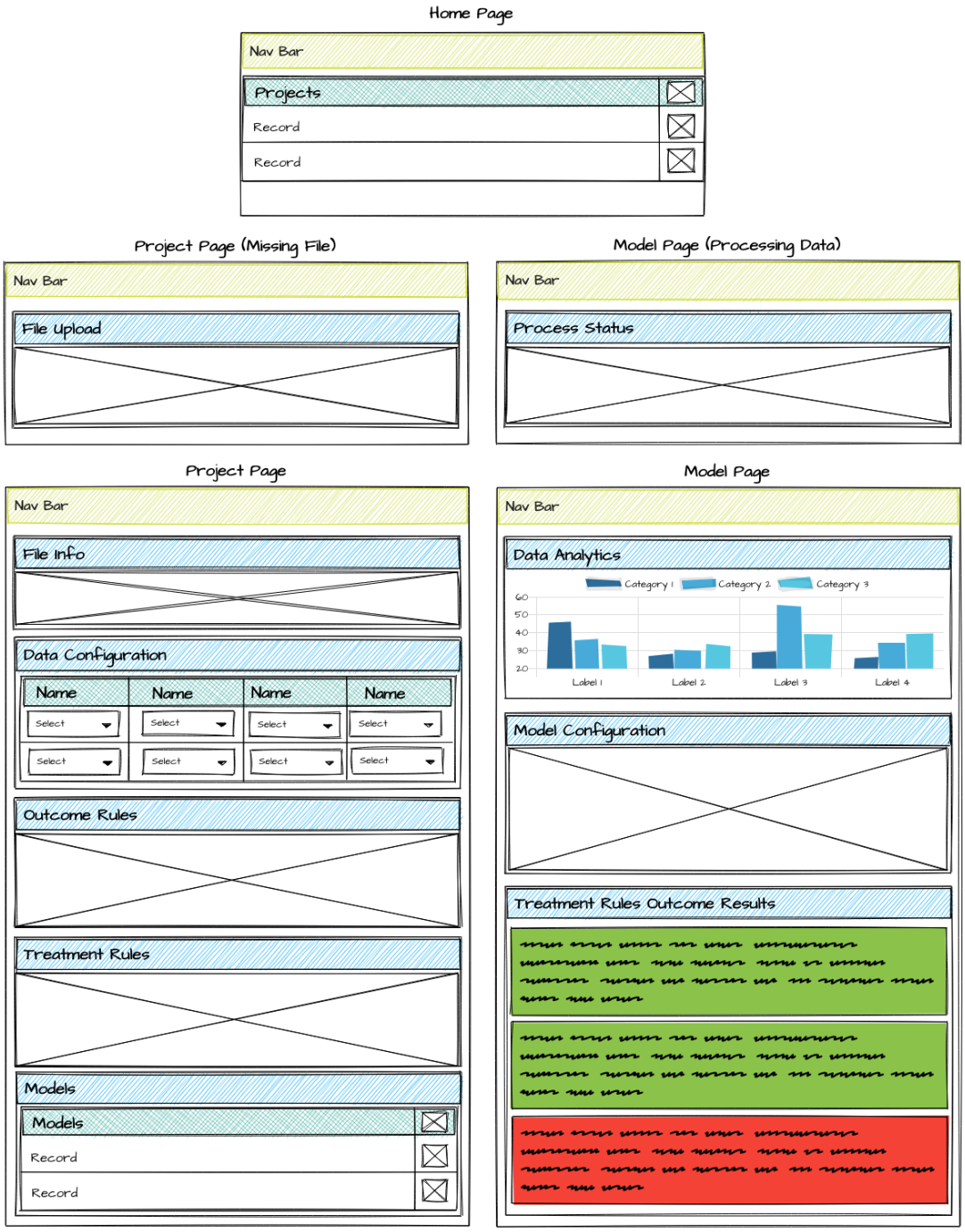

Figure 6. Wireframe depicting the initial structure of all interface pages.

4.2 UI development

The process of developing front-end applications is notoriously known for its constant reworks, and even though it is impossible to anticipate when and what part of our UI will require a rework, we can still mitigate some issues, that come with reworks, by implementing a proper code structure.

When it comes to structuring code in a front-end application, there are still debates in the software engineering community as to how it should be done. In the scope of our UI development process, we have decided to implement a structure that is commonly referred to as the Component or Component-Based.

Component structure is a type of software development pattern that advocates for structuring code in small, reusable components. The general advantage of such structure is that it forces us to write small reusable components, which in turn reduces the time spent on development, testing, and reworks. It should be also mentioned that component structure helps us to maintain a uniform style in all of our user interface pages, which is something that is generally advised to have in any user interface.

4.3 UI walk-through

In this last section of our UI chapter, we will provide a walk-through of our user interface and all of its components from an end-user perspective. In the process of this walk-through, as an example, we will be using a CSV event log file, obtained from [BTD+20] paper. The log file is essentially a list of interactions that have occurred between some anonymous bank and their clients, who are interested in getting a loan. In table 2, we have provided a quick description of the main columns in the log file.

Table 2. Log file columns and their explanation

Column	**Description**
ApplicationType	Loan application type
LoanGoal	Client loan goal
RequestedAmount	The amount of credit requested by the bank client
Case ID	A unique identifier
Activity	Type of action taken by the bank(send an email, call the client, change contract, etc.)
Selected	An indicator if a client has selected the offer
time:timestamp	Timestamp of the event

4.3.1 Project list page

As soon as users navigate to our application, the first page that they are met with is the project list page (Figure 7). On this page, users can see all the created projects, navigate to, or delete any one of the existing projects. On this same page, users can also create new projects (Figure 8).

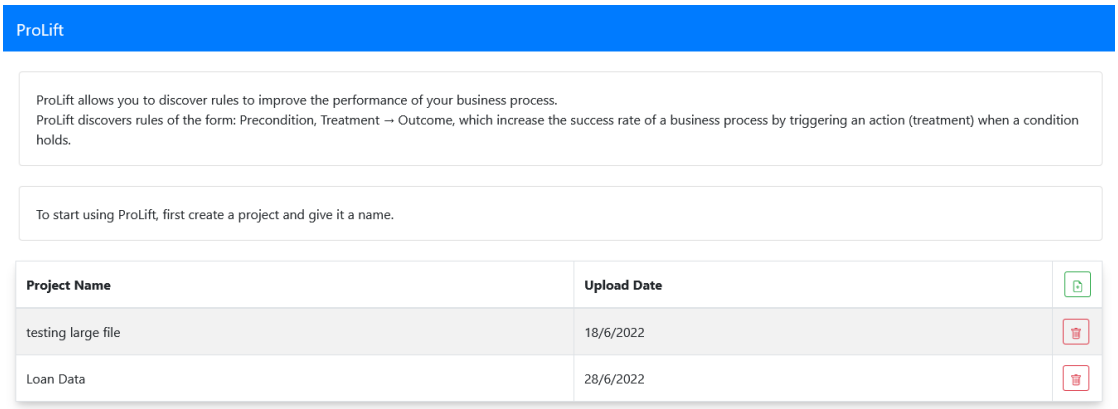

Figure 7. Project list page example.

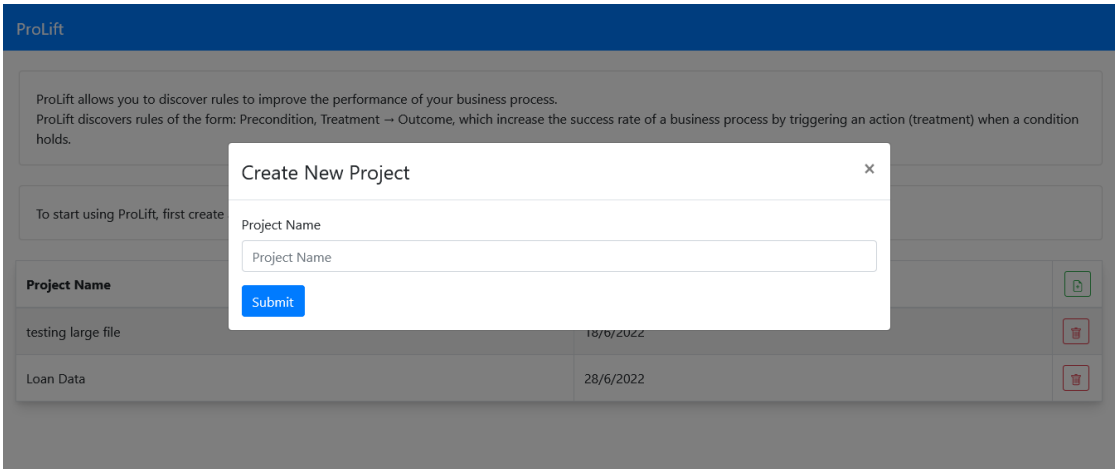

Figure 8. Project creation window example.

4.3.2 Project configuration page

After a user has selected a project or created a new one, the user is redirected to the project configuration page. Depending on whether the project has a file associated with it, the end user will see one of two options.

If a project doesn't have a file associated with it, the end user will see a window prompting them to upload a file for processing (Figure 9). If a projected dose has a file associated with it, then the end user will see sections where they can start configuring the project (Figure 10).

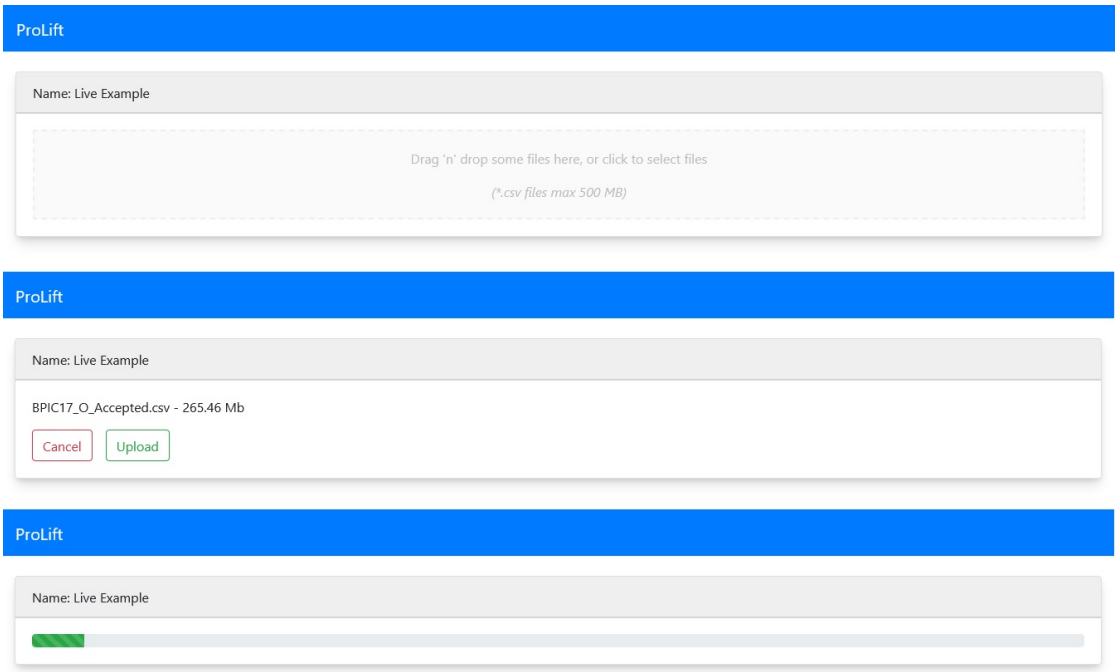

Figure 9. Project file upload example.

The project configuration (Figure 10) consists of 5 sections: overview, column configuration, outcome configuration, treatment configuration, and models.

Overview section gives a simple overview of the uploaded file and some of its metadata.

Column configuration section gives end users the ability to specify column type and data type of the uploaded file. After a user have uploaded a comma-separated values(CSV) file, our application is capable of discovering the columns that are present in the uploaded file however, it is up to the end user to specify the data type and the column type of each column.

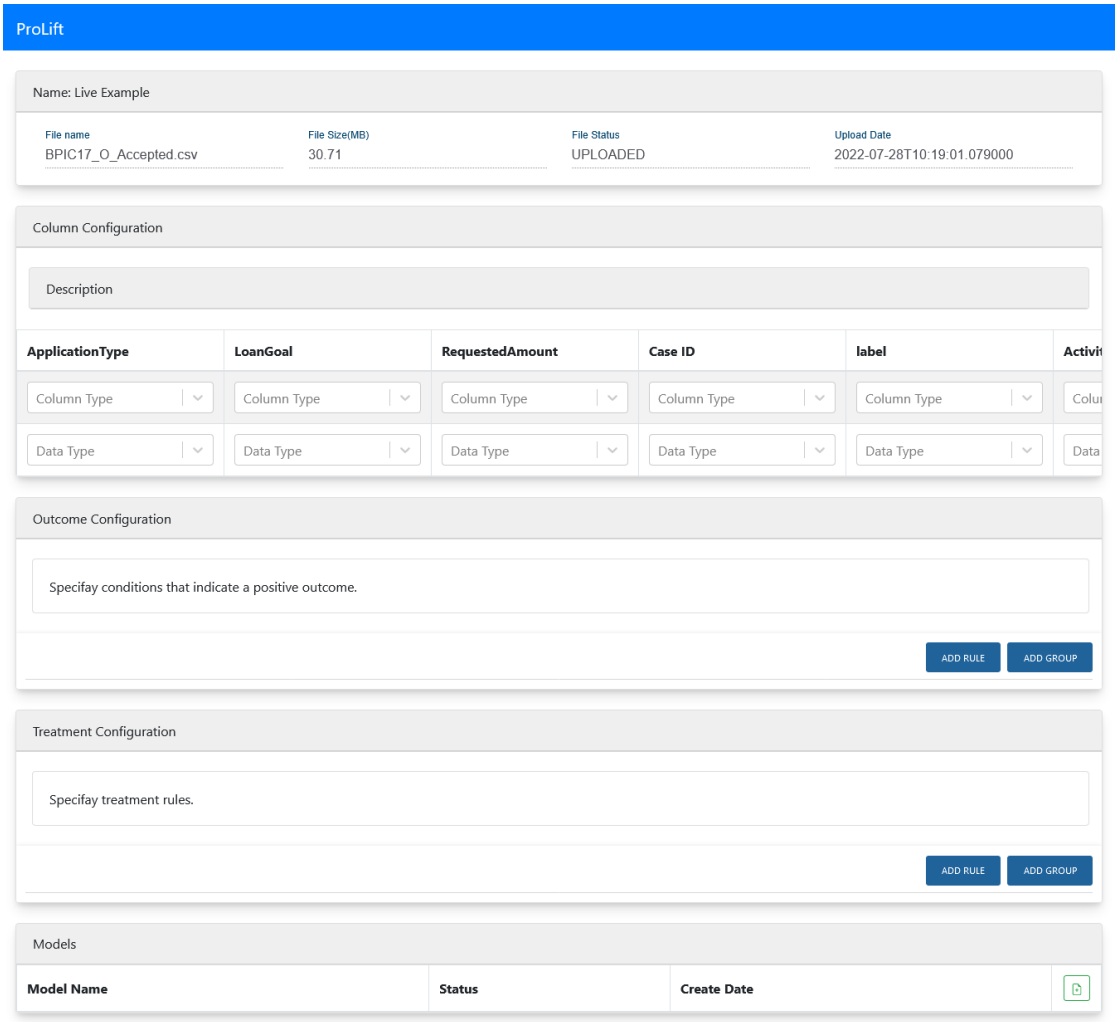

Figure 10. Project configuration example.

By specifying the data type, the end user provides information as to what type of data is present within a given column: numerical, string, date, Boolean, etc.

By specifying the column type, the end user provides information as to how should a given column be treated by the rule extraction process. The column type filed can have one of the following values:

- Uncontrollable - is a column where the value can not be modified by the file provider. Example: age of a client, or in the case of a loan application, the requested amount.
- Controllable - is a column where the value can be influenced by the file provider.

Example: in a case of a loan application, interest rate, or loan period. This value can also be used to mark columns that indicate actions taken by the company (calling a potential client, sending an email, etc.).

- Outcome - is a column that is used to determine the outcome of a record or a set of records.
- Group Identifier - is a column that is used to group records. Example: customer ID.
- Ignore - columns marked by this type will be ignored by the rule extraction process.
- Order By - columns marked by this type will be used to order all records in descending order.

Outcome configuration is a section where the end user has to specify how to determent that a given record or a set of records have a positive outcome. Figure 11 is an example of an outcome configuration. In this example, we mark records with a positive outcome if the value in the column 'Selected'(refer to table 2) is equal to 'True'.

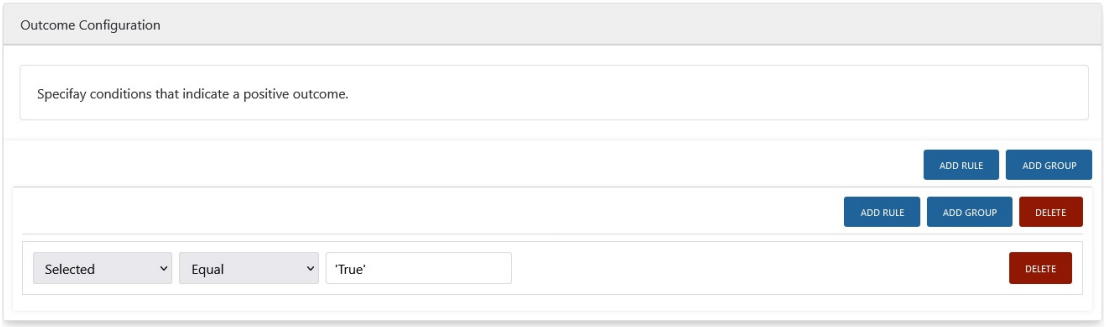

Figure 11. Outcome configuration example.

Treatment configuration is a section where the end user has to specify the treatment rules, that they are interested in analyzing. Figure 12 is an example of a treatment configuration. In this example, we search for cases where the value 'O_Sent (mail and online)' has occurred successively twice in the 'Activity' column (refer to table 2).

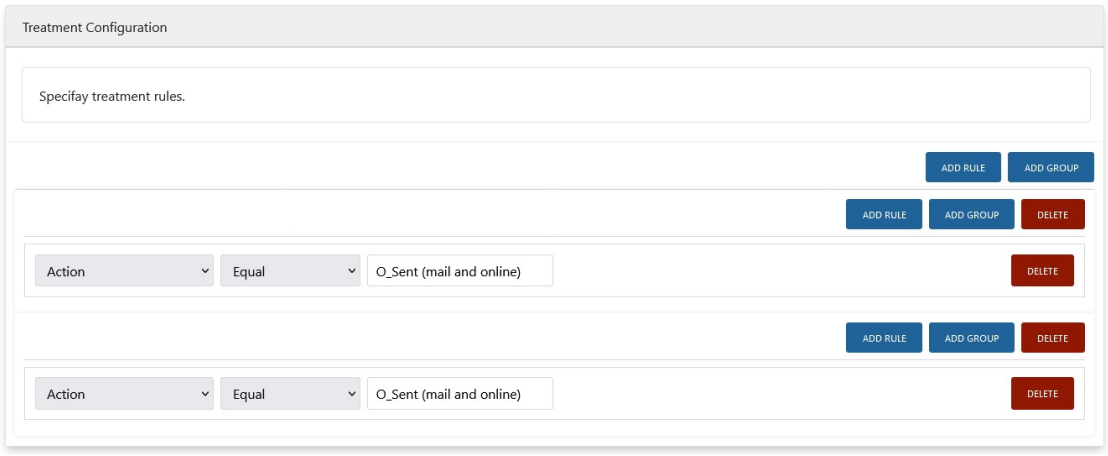

Figure 12. Treatment configuration example.

Models is the last section of the project configuration. After all the configurations have been provided, the end user can create a model configuration, which will initiate data prepossessing and navigate the end user to the Model Configuration page.

4.3.3 Model configuration page

The model configuration page is the last page in our application. Deepening to the data prepossessing status, the end user will either see a process bar (Figure 13), indicating how much percentage of the data has been prepossessed, or they will see a page containing 5 sections (Figure 14): data analytics, data balancing, confusion matrix, model settings, and results.

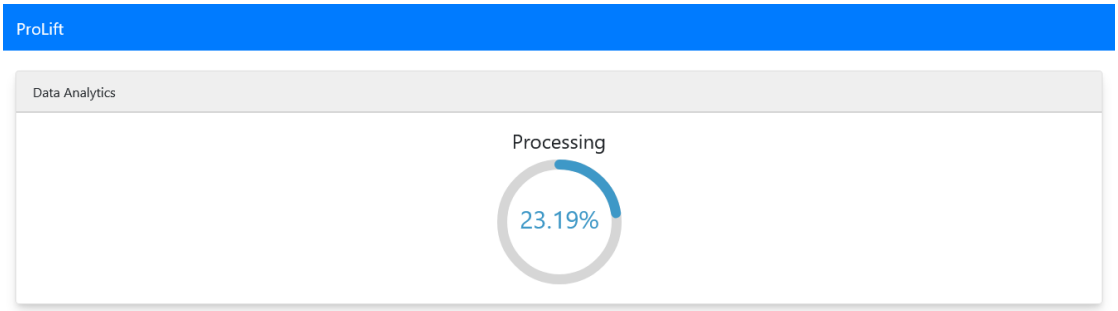

Figure 13. Data prepossessed progress bar example.

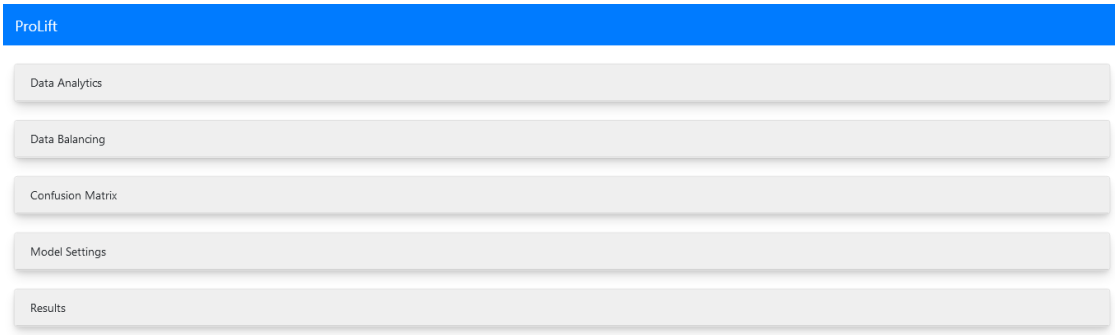

Figure 14. Model configuration page example.

Data analytics section consists of 2 stacked bar graphs (Figure 15), that are designed to give the end user an overview of the data with respect to the treatment and outcome rules that have been specified in the project configuration page(4.3.2)).

The 1st graph illustrates the stacked bar of the confusion matrix with respect to case length. In other words, the x-axis indicates the size of an individual case in a given log while the y-axis is a confusion matrix that depicts how many cases of the same length had a positive outcome, how many had a negative outcome, how many had a treatment present and how many didn't.

The 2nd graph illustrates a relation between positive/negative outcomes and the number of interactions before all the application of a treatment. In other words, the x-axis indicates the number of actions/steps that have been taken before all the treatment rules have been applied, and the y-axis indicates the outcome of a case.

Data balancing section is designed to give end users the ability to balance their data before training a model. In this section, users are given an option to provide a 'Cut-off Percentile' and 'MaxDelta'(Figure 16). 'Cut-off Percentile' essentially means what percentile of the data should be used in model training and 'MaxDelta' means what can be the maximum percentage difference between a minimum and maximum value in the data confusion matrix (refer to Figure 17). By pressing the 'Balance Data' button, users can execute a data balancing process that rebalances the data based on the provided configuration.

Confusion matrix section, displays a simple confusion matrix of the data, that will be used in model training (Figure 17).

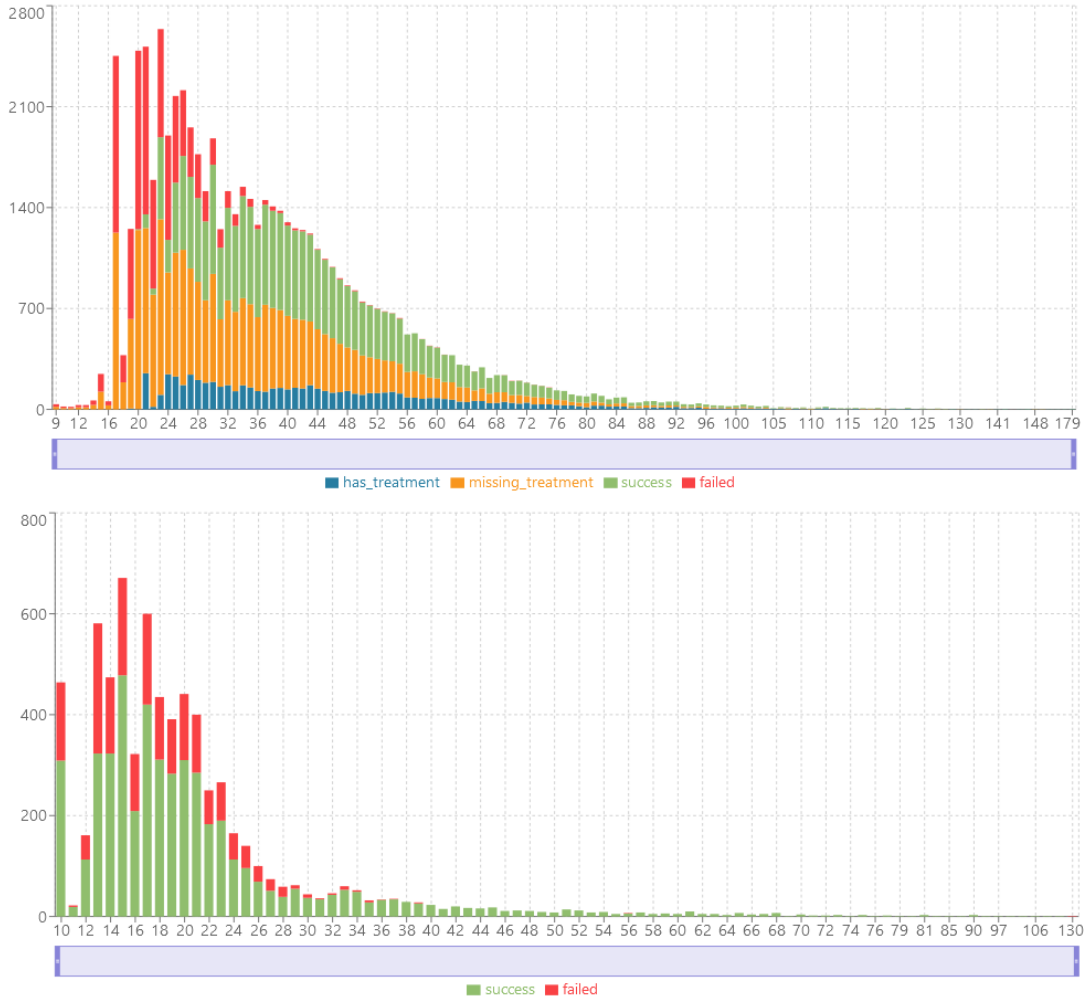

Figure 15. Data analytics section example.

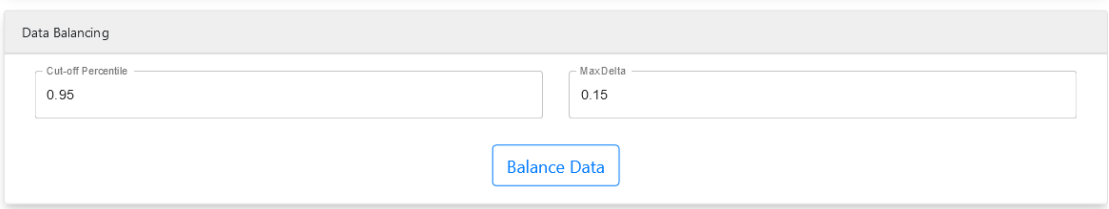

Figure 16. Data balancing section example.

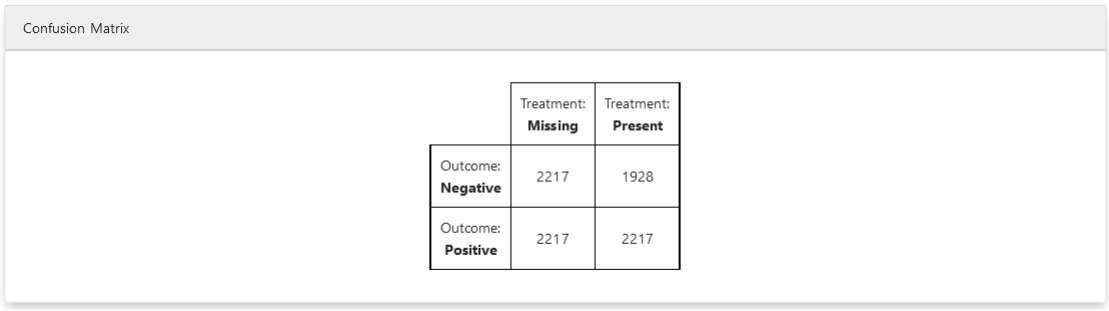

Figure 17. Confusion matrix section example.

Model settings section is the last step in the model configuration (Figure 18). Here, the end user can specify the configuration for the CasualML model and execute the rule extraction process by clicking the 'Build Model' button.

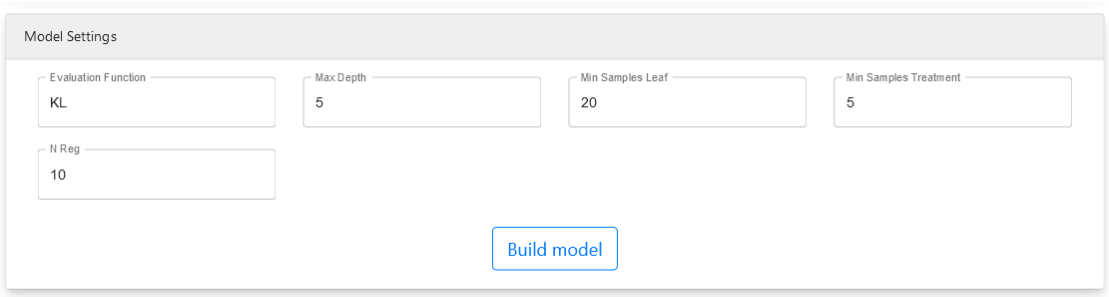

Figure 18. Model settings section example.

Results After a model has been constructed and rules have been extracted, the end users will see in this section a set of cases (Figure 19), with their description and the outcome percentage, for which the application of treatment rules has either a positive or a negative outcome.

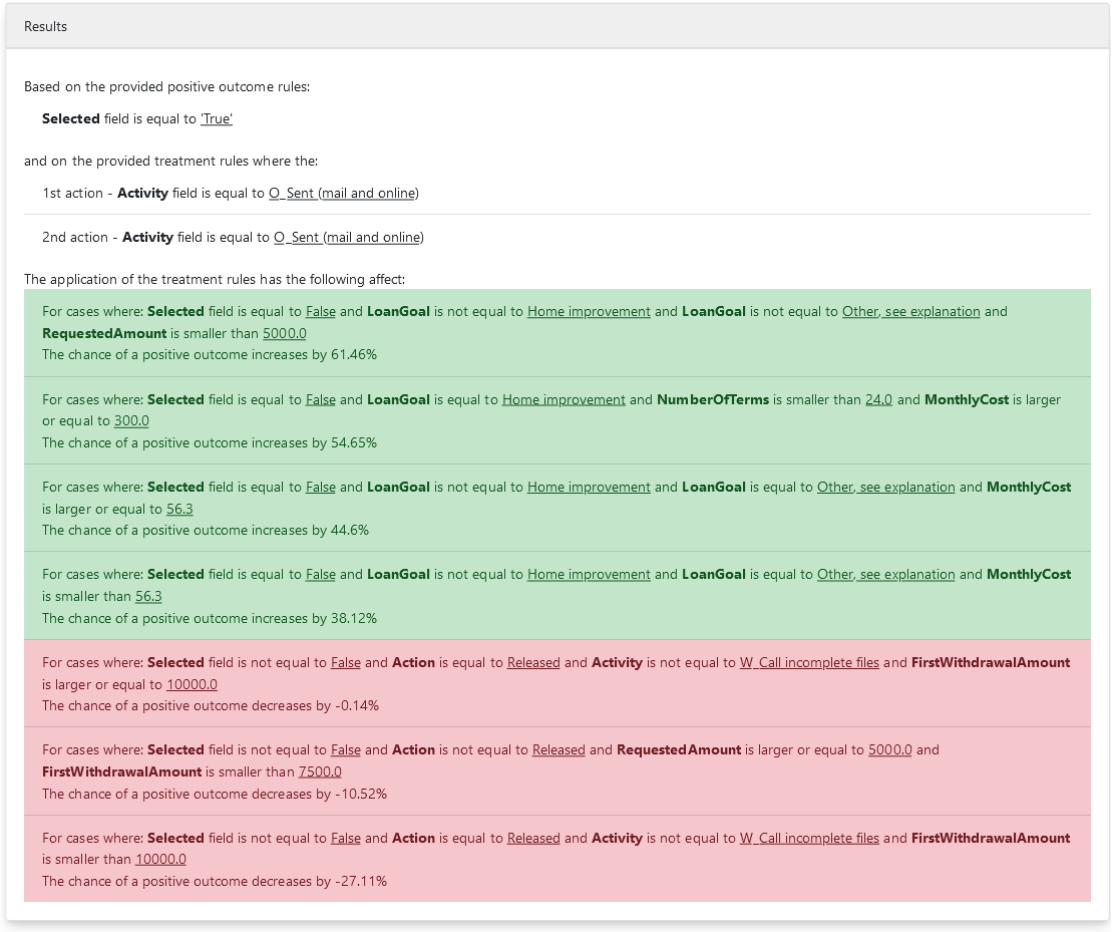

Figure 19. Results section example.

5 Data Structure

As already mentioned, one of our application requirements is to store necessary information and to provide it when requested. In this chapter, we will provide an overview of our application data structure. We will start by giving a brief introduction to the types of relationships existing in MongoDB, followed by a detailed explanation of our entities.

5.1 Data Model Types

In the process of creating relationships between different entities, it's important to choose an appropriate type. MongoDB provides two ways of binding entities, which are: referencing and embedding.

Referenced relation is the classical form of binding entities. In referenced binding, we describe relationships using a referenced attribute also commonly referred to as a foreign key. In MongoDB, it is recommended to use referenced binding in cases when you need to build complex many-to-many relationships or when you have to build large hierarchical data sets [Ince].

Embedded relation is a type of relationship where we bind related data in a single structure. Unlike in referenced relation, embedded relation allows us to store related pieces of information in one database record which in turn makes querying the information much faster, than in referenced relation.

Figure 20 illustrates an example of referenced and embedded relationship. In this example, we use two abstract entities: user and contact.

In the case of a referenced relationship, the user and contact entity are two completely separated entities, the records of which are stored in two completely separated files and are related to one another only through the user ID attribute. In this case, querying user information will not return contact information.

In the case of the embedded relationship, the user entity is the main entity and the contact entity is an embedded entity of the user entity. In this case, querying the user information will also return the contact information.

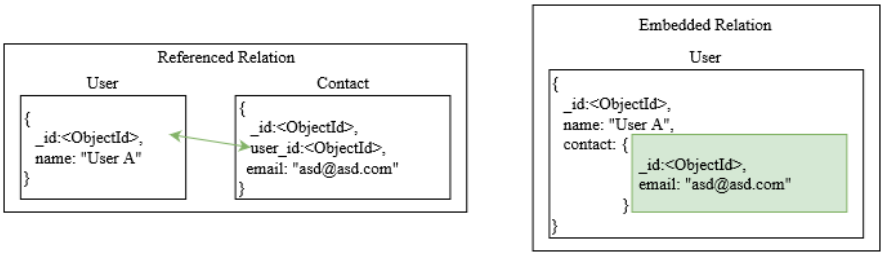

Figure 20. Example of referenced and embedded relationship.

5.2 Database Structure and Entities

Below, we have provided 3 diagrams (Figure 21, 22 and 23) depicting the structure of our entities and their relationship to one another. The first diagram (Figure 21) depicts the project entity and entities that are in a direct referenced relationship with it. The second diagram (Figure 22) depicts the model configuration entity and entities that are embedded within it. And lastly, the third diagram (Figure 23) depicts the model configuration entity and entities that are in a direct referenced relationship with it. In the next few paragraphs, we will provide a quick description of our entities.

Project is the main entity in our database. This entity is designed to store some basic project-related information and to bind file-related information to the model configuration entity (Figure 21).

Folder and File Data are entities that allow our application to keep track of user-provided file metadata and the location of the file in the file repository (Figure 21). Since our application stores all the user-uploaded files in a remote file repository, it's important to know where exactly are those files located in the repository so that we can access them when it's required.

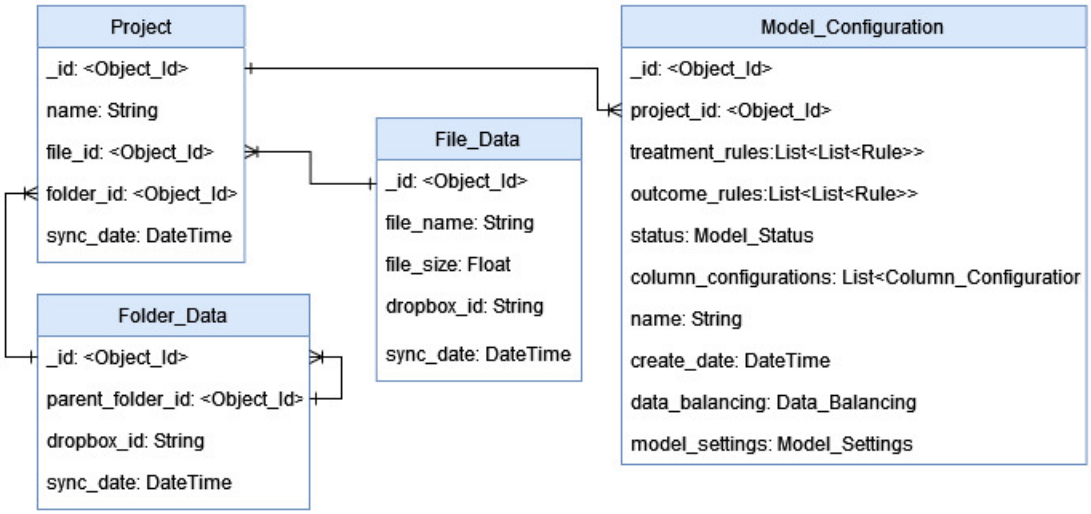

Figure 21. Project-related entities.

Model Configuration is the largest entity in our database. This entity is responsible for storing all the necessary information for executing the rule extraction process. This entity has a complex structure, which consists of a few attributes and 5 embedded entities (Figure 23), which are: model settings, data balancing, column configuration, treatment rules, and outcome rules.

Model Settings is an entity that stores model configuration settings (Figure 22). These settings indicate what type of model, and what model configuration, will be used in the model-building process.

Data Balancing is an entity (Figure 22) that stores configuration related to the data balancing process. Besides the data balancing configuration, this entity also stores a

confusion matrix that indicates the distribution of data for that configuration.

Column Configuration is an entity (Figure 22), that is responsible for storing the configuration of each column. After a user has uploaded a CSV file, our application will extract all the columns from that file and store their information within this entity.

Rule is an entity (Figure 22 and 23), that is used to describe a certain rule. This entity is used as a base entity for the construction of treatment, outcome, and result rules.

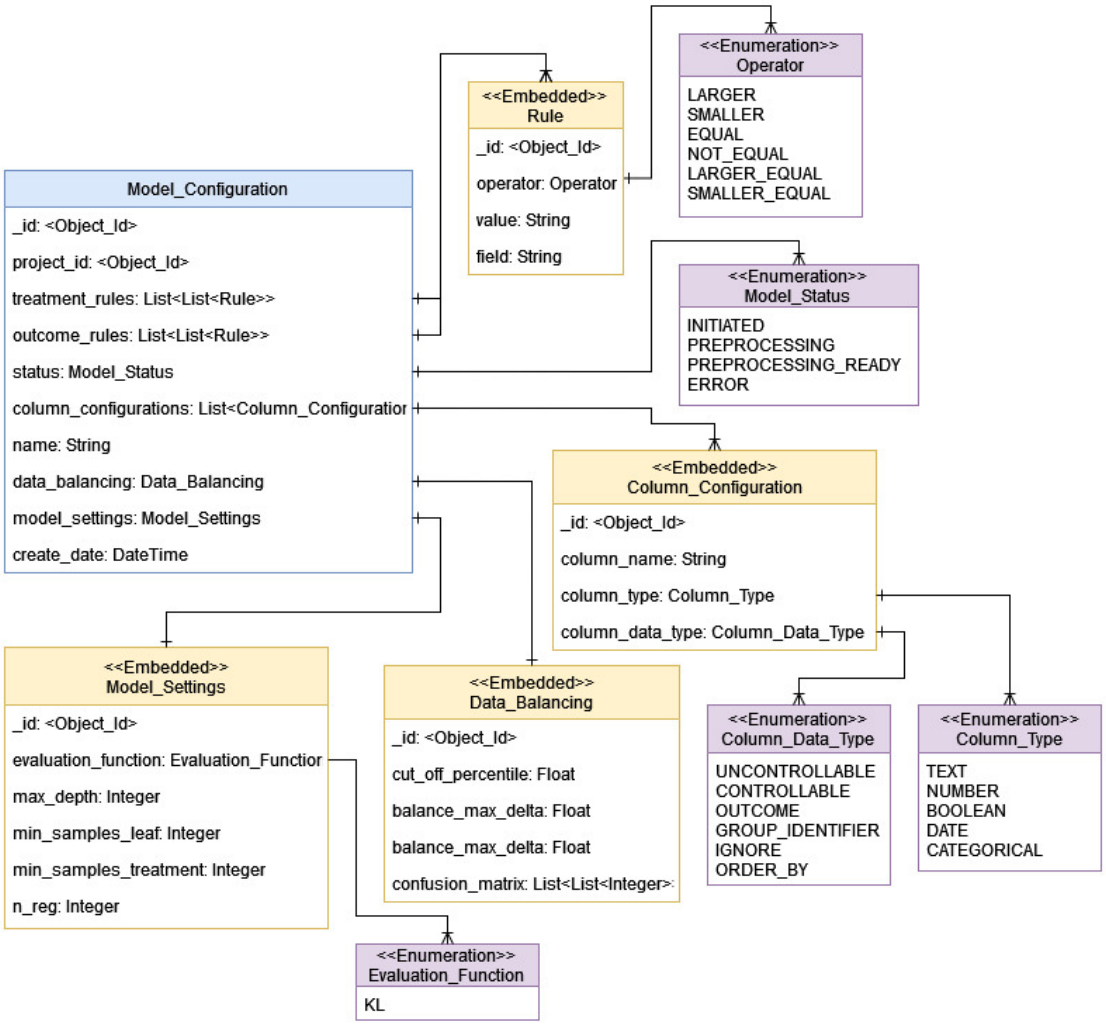

Figure 22. Model configuration and its embedded entities.

Operator, Model Status, Evaluation Function, Column Data Type and Column Type are enumerators, that are designed to store a list of limited values (Figure 22 and 23). The main point of enumerators is to limit the range of possible values that an attribute can have. Enumerators are also commonly used to mitigate typo errors and are considered to be a generally good clean code and clean data practice.

Prefix Data and Prefix Row are entities (Figure 23) that will store information on preprocessed data. During the data preprocessing, our application will generate a mask, which is later on used to determine what to do with a given set of data. That mask data is stored within these two entities.

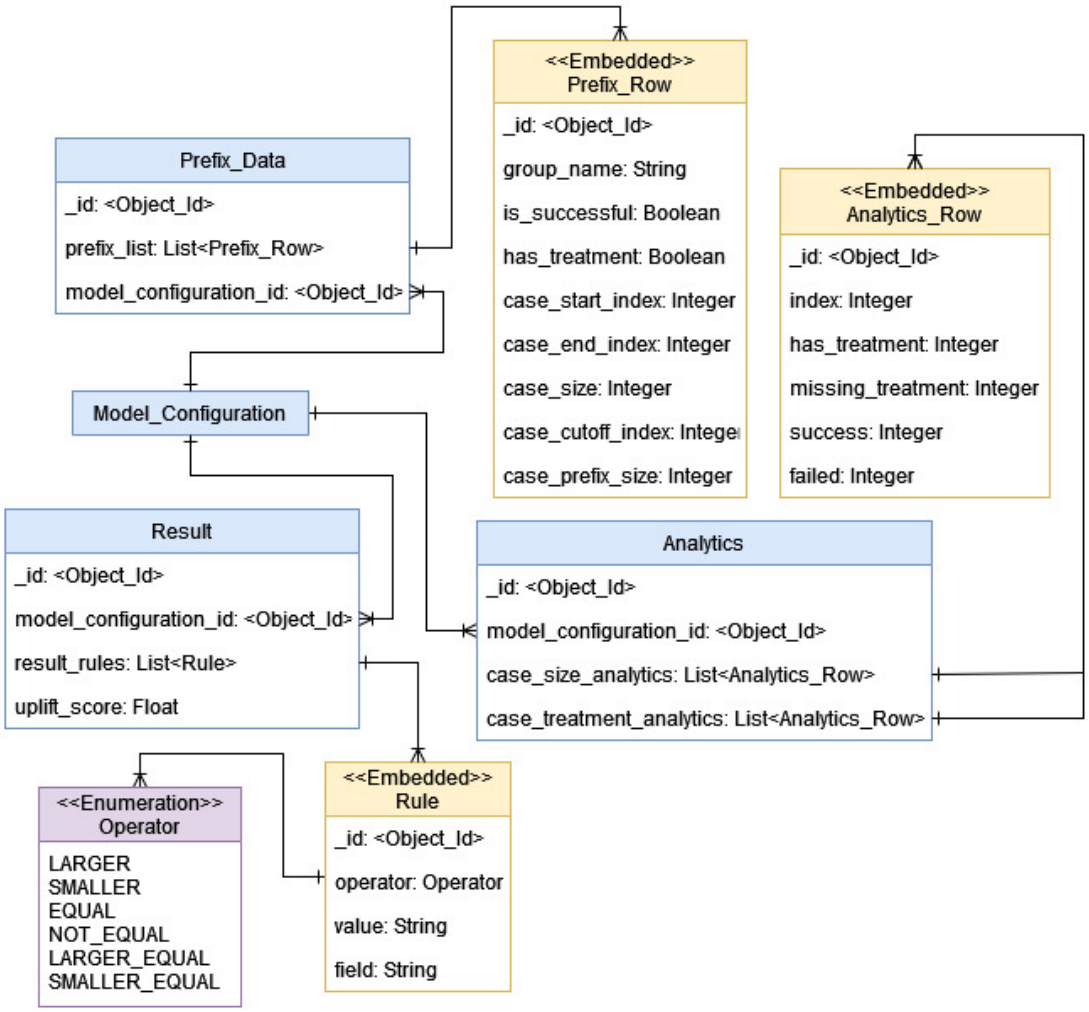

Figure 23. Model configuration-related entities.

Analytics and Analytics Row are entities (Figure 23) that store information related to the analytics segment of our UI interface (Figure 15). Since processing large sets of data on the front-end is not reasonable nor feasible, the process of gathering and formatting analytical data is performed in the back-end of the application. After the analytical data is gathered, our application will store all the analytical data in these two entities for quick and easy access.

Result is the last entity (Figure 23) in our database, its purpose is to store information related to the outcome of the business rule extraction process.

6 Production Deployment

One of our end goals was the deployment of the application into a live environment accessible to the public. In this chapter, we will describe the deployment process of our application, the additional tools used to simplify the deployment process, and the structure of our application in the live environment.

6.1 Containerization

The process of application deployment can be a very painful one, especially if done using the older methods of just moving code from one hardware onto another. This traditional method often results in unexpected bugs or errors, occurring primarily because of the new environment, where the application hasn't been fully tested. In order to address commonly occurring problems in the deployment process, engineers have come up with a solution commonly referred to as containerization.

In simple terms, containerization is the process of packaging application code with a set of operating system(OS) libraries and dependencies required to run that code. This process creates an isolated system that is easy to deploy, run, and maintain in nearly any environment. Moreover, containerization even improves the development process as it removes the necessity for maintaining OS-related dependencies by the developing team in their development environment.

In order to reduce any unnecessary problems related to deployment and maintenance, we have containerized most of our application elements with the help of Docker.

Docker is an incredibly powerful tool that is designed primarily to simplify the process of packaging containers and their deployment [Inca]. Besides the containerization process, Docker also has a very large software image repository that helps developers to share and re-use pre-build images and a wide range of tools that simplify the deployment of multi-container applications.

6.2 Live environment

In the appendix of this thesis, we have added a listing (Listing 1) which describes the configuration of the Docker compose file that was used in the deployment process. Since we wanted to utilize containerization to its fullest potential, our multi-container application ended up consisting of five isolated containers and one shared volume. Within the scope of your application, the shared volume is used primarily for storing temporary, files that were provided by the end users.

Figure 24 depicts the structure of our live application, which is currently deployed and running on one of the Tartu University servers, accessible by the following URL: `https://prolift.cloud.ut.ee`.

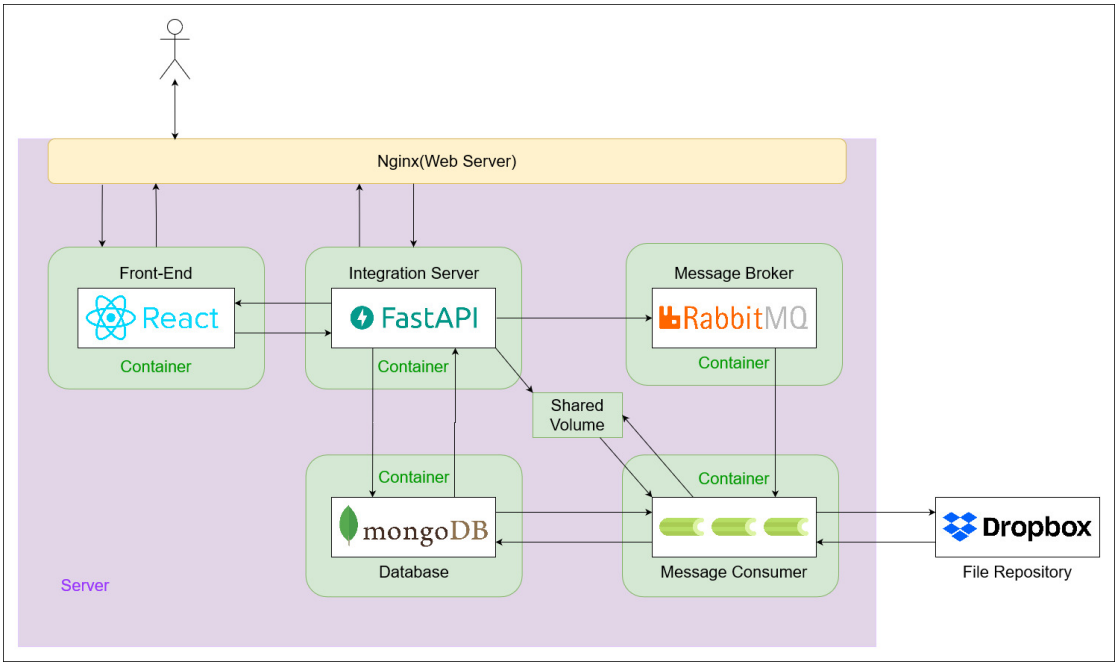

Figure 24. Application live structure.

7 Conclusion

In this thesis, we've provided an overview of our rule discovery application. We have given a detailed description of our application architecture, with an in-depth explanation of each component present within the application.

As a result, we have a scalable, user-friendly rule discovery application that utilizes the latest methodology present in the field of casual machine learning and business process mining. The application can:

- Process CSV files provided by the end user.
- Provide end users with a simple UI for configuring the rule extraction process.
- Extract rules based on the provided data and configuration.
- Present extracted rules in a humanly readable way.

We were also able to make the application publicly available by deploying it on one of the Tartu University servers, accessible by the following URL: `https://prolift.cloud.ut.ee`.

The source code with the deployment instructions of the project can be found in the following repository `https://gitlab.com/aleksei.kopolov/action-recommendation`. As for the future work, it can be divided into 2 categories: work related to software engineering and work related to computer science.

From the software engineering perspective, the treatment and outcome rule configuration components (Figure 11 and 12) are far from ideal and do require a rework. Some data configuration steps could be automated, such as the selection of column data type (Figure 10). Also, data preprocessing can be improved by implementing a rolling window.

From the computer science perspective, it would be interesting to also implement other causal machine learning algorithms and not just tree-based ones. We are also missing a good analysis of the results extracted by our application.

References

[BTD+20] Zahra Dasht Bozorgi, Irene Teinemaa, Marlon Dumas, Marcello La Rosa, and Artem Polyvyanyy. Process mining meets causal machine learning: Discovering causal rules from event logs. In Boudewijn F. van Dongen, Marco Montali, and Moe Thandar Wynn, editors, *2nd International Conference on Process Mining, ICPM 2020, Padua, Italy, October 4-9, 2020*, pages 129–136. IEEE, 2020.

[CHL+a] Huigang Chen, Totte Harinen, Jeong-Yoon Lee, Mike Yung, and Zhenyu Zhao. Causalml documentation. `https://causalml.readthedocs.io/en/latest/about.html`.

[CHL+b] Huigang Chen, Totte Harinen, Jeong-Yoon Lee, Mike Yung, and Zhenyu Zhao. Causalml repository. `https://github.com/uber/causalml`.

[CHL+20] Huigang Chen, Totte Harinen, Jeong-Yoon Lee, Mike Yung, and Zhenyu Zhao. Causalml: Python package for causal machine learning, 2020.

[DMV18] Floris Devriendt, Darie Moldovan, and Wouter Verbeke. A literature survey and experimental evaluation of the state-of-the-art in uplift modeling: A stepping stone toward the development of prescriptive analytics. *Big Data*, 6(1):13–41, 2018. PMID: 29570415.

[DRMR18] Marlon Dumas, Marcello La Rosa, Jan Mendling, and Hajo A. Reijers. *Fundamentals of Business Process Management*. Springer Berlin, Heidelberg, 2018.

[Dum] Marlon Dumas. Causal process mining. `https://www.linkedin.com/pulse/causal-process-mining-marlon-dumas/`.

[Edu] IBM Cloud Education. What is mongodb? `https://www.ibm.com/cloud/learn/mongodb`.

[Inca] Docker Inc. Docker documentation. `https://docs.docker.com/`.

[Incb] Dropbox Inc. Dropbox documentation. `https://www.dropbox.com/developers`.

[Incc] Meta Platforms Inc. React documentation. `https://reactjs.org/`.

[Incd] Meta Platforms Inc. React repository. `https://github.com/facebook/react/`.

[Ince] MongoDB Inc. Mongodb documentation. https://www.mongodb.com/docs/.

[Incf] VMware Inc. Rabbitmq documentation. https://www.rabbitmq.com/.

[KLL+22] Jean Kaddour, Aengus Lynch, Qi Liu, Matt J. Kusner, and Ricardo Silva. Causal machine learning: A survey and open problems, 2022.

[Pan12] T. Panagacos. *The Ultimate Guide to Business Process Management: Everything You Need to Know and how to Apply it to Your Organization*. Amazon Digital Services LLC - KDP Print US, 2012.

[Rama] Sebastián Ramírez. Fastapi documentation. https://fastapi.tiangolo.com/.

[Ramb] Sebastián Ramírez. Fastapi repository. https://github.com/tiangolo/fastapi.

[SKa] Ask Solem and Omer Katz. Celery documentation. https://docs.celeryq.dev/en/stable/index.html.

[SKb] Ask Solem and Omer Katz. Celery repository. https://github.com/celery/celery.

[vdA22] Wil M. P. van der Aalst. *Process Mining: A 360 Degree Overview*, pages 3–34. Springer International Publishing, Cham, 2022.

Appendix

I. Application configuration file

```
version: "3.9"
services:
    rabbitmq_server:
        image: rabbitmq:3-management
        container_name: rabbitmq
        ports:
            - "5672:5672"
        environment:
            RABBITMQ_DEFAULT_USER: mq_admin
            RABBITMQ_DEFAULT_PASS: mq_admin
    mongo_db:
        image: mongo
        container_name: mongo_db
        ports:
            - "27017:27017"
        environment:
            MONGO_INITDB_ROOT_USERNAME: mongo_admin
            MONGO_INITDB_ROOT_PASSWORD: mongo_admin
    pyrule_client:
        build: ./pyrule-client
        container_name: pyrule-client
        ports:
            - "443:3000"
        stdin_open: true
        tty: true
        environment:
            - "REACT_APP_BACK_END_URL=http://localhost:80/
              "
    pyrule_backend:
        build:
            context: ./pyrule-backend
            dockerfile: Dockerfile-fastapi
        container_name: pyrule-backend
        environment:
            FRONT_END_URL: http://pyrule_client:443/
            BACK_END_URL: http://pyrule_backend:80/
```

```
            RABBIT_MQ_URL: amqp://mq_admin
                :mq_admin@rabbitmq_server:5672
            MONGO_DB_ULR: mongodb://mongo_admin
                :mongo_admin@mongo_db:27017
            DROP_BOX_TOKEN: db_token
        ports:
            - "80:8000"
        depends_on:
            - mongo_db
        volumes:
            - data_folder:/src/temp
    worker:
        build:
            context: ./pyrule-backend
            dockerfile: Dockerfile-celery
        hostname: worker
        environment:
            FRONT_END_URL: http://pyrule_client:443/
            BACK_END_URL: http://pyrule_backend:80/
            RABBIT_MQ_URL: amqp://mq_admin
                :mq_admin@rabbitmq_server:5672
            MONGO_DB_ULR: mongodb://mongo_admin
                :mongo_admin@mongo_db:27017
            DROP_BOX_TOKEN: db_token
        links:
            - rabbitmq_server
        depends_on:
            - rabbitmq_server
            - mongo_db
        volumes:
            - data_folder:/src/temp
volumes:
    data_folder:
```

Listing 1. Container ensemble configuration

II. Licence

Non-exclusive licence to reproduce thesis and make thesis public

I, **Aleksei Kopõlov**,
(author's name)

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to

   reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

   ProLift: A Web Application to Discover Causal Treatment Rules From Business Process Event Logs,
   (title of thesis)

   supervised by Marlon Dumas.
   (supervisor's name)

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Aleksei Kopõlov
06/08/2022