Prague University of Economics and Business
Faculty of Informatics and Statistics

Analytical Platform Integrating CleverMiner

MASTER THESIS

Study program: Information Systems and Technology
Specialization: Information Systems Development

Author: Bc. Petr Štípek
Supervisor: RNDr. Ing. Petr Máša, Ph.D.

Prague, May 2026

Declaration on the Use of Artificial Intelligence

I confirm that artificial intelligence tools were used in the preparation of the submitted work. Specifically, they were used in the following ways:

- improvement of grammar and language style,
- generation of parts of source code,
- other (Documenation generation).

Each use of any of the above methods is separately documented in an appendix to the thesis, which provides a more detailed explanation of which tool was used in which specific part of the work; the author's own contribution is highlighted.

I confirm that:

- my work is original and was prepared in accordance with ethical standards, in particular the Code of Ethics of the Prague University of Economics and Business; that all information sources used are properly cited in the work; and that any generated content has been verified by me;
- I accept full responsibility for the entire content of the thesis.

Acknowledgements

I would like to express my sincere gratitude to my thesis supervisor, RNDr. Ing. Petr Máša, Ph.D., for his constructive feedback, valuable guidance, and continuous support throughout the writing of this thesis.

Abstrakt

Tato diplomová práce představuje návrh a implementaci uživatelsky orientované analytické platformy pro knihovnu CleverMiner, jejímž cílem je umožnit doménovým expertům provádět analýzy dolování dat založené na GUHA procedurách bez nutnosti znalosti programování. Teoretická část práce pokrývá principy dolování asociačních pravidel, metodu GUHA a analýzu knihovny CleverMiner a jejích dostupných procedur. Na základě systematického hodnocení existujících GUI řešení jsou přijata architektonická a implementační rozhodnutí. Platforma je implementována jako webová aplikace zahrnující frontend v technologii React, backend postavený na frameworku Django a vrstvu distribuovaného zpracování úloh pomocí nástroje Celery, přičemž podporuje všechny čtyři procedury dostupné v knihovně CleverMiner. Aplikace poskytuje integrované prostředí pro pracovní postup dle metodiky CRISP-DM, včetně průzkumu dat, jejich předzpracování, definice a spuštění dolovacích úloh a vizualizace výsledků. Platformu lze nasadit lokálně prostřednictvím nástroje Docker nebo na cloudové infrastruktuře. Vývoj byl řízen metodikou MMSP-AV. Uživatelské akceptační testování provedené se studenty obeznámenými s obdobnými nástroji potvrdilo, že všechny definované funkční a nefunkční požadavky byly splněny a nebyly identifikovány žádné kritické defekty.

Klíčová slova

CleverMiner, asociační pravidla, dolování dat z databází, GUHA procedury

Abstract

This thesis presents the design and implementation of a user-oriented analytical platform for the CleverMiner library, aimed at enabling domain experts to perform data mining analyses based on GUHA procedures without requiring programming knowledge. The theoretical foundation covers the principles of association rule mining, the GUHA method, and an analysis of the CleverMiner library and its available procedures. A systematic evaluation of existing GUI-based solutions is conducted to inform the architectural and implementation decisions. The platform is implemented as a web application comprising a React frontend, a Django backend, and a Celery-based distributed task execution layer, supporting all four procedures available in CleverMiner. The application provides an integrated environment for the CRISP-DM workflow, including dataset exploration, preprocessing, mining task definition, execution, and result visualisation. The platform is deployable both locally via Docker and on cloud infrastructure. Development was guided by the MMSP-AV methodology. User acceptance

testing conducted with students familiar with comparable tools confirmed that all defined functional and non-functional requirements were met with no critical defects identified.

Keywords

CleverMiner, association rules, data mining, GUHA procedures

Contents

List of Figures

List of Tables

List of source codes

Introduction

Data mining methods, such as association and descriptive rule mining, represent a well-established approach to extracting knowledge from datasets (Padilla-Rascón et al., 2024). These techniques find application across a wide range of domains, from business and marketing, such as market basket analysis, to healthcare, industry, and social sciences, where they help uncover hidden patterns and relationships in data. Despite their broad applicability, the practical use of these methods remains largely limited to technically proficient users. Domain experts are frequently dependent on data analysts or developers to conduct analyses and interpret results, which reduces their autonomy and limits the practical utility of available tools (Behringer, 2023).

In recent years, the concept of the democratisation of data analytics has emerged as a response to this limitation, aiming to make advanced analytical techniques accessible to a broader audience beyond IT specialists (Scholz, 2023). Industry trends confirm growing demand for self-service, low-code, and no-code platforms (Varshney, 2023). For analytical tools to reach less technical users, they must produce outputs that are easy to understand and present them in an intuitive form. In the context of association rule mining, results take the form of human-readable IF-THEN rules, which offers an inherent interpretability advantage (Rane et al., 2024). However, without adequate explanation and visualisation, even these rules can prove difficult for end users to interpret and validate (Padilla-Rascón et al., 2024).

The CleverMiner library implements enhanced association rule mining (eARM) based on GUHA[1] procedures via a Python API (Little Big Company & Máša, 2026). While the library produces interpretable results in the form of analytical rules, it requires users to have knowledge of Python and several additional packages, making it inaccessible to domain experts without a programming background. This thesis addresses that gap by designing and implementing a user-oriented analytical platform for CleverMiner, enabling domain experts to perform the full data analysis workflow without requiring programming knowledge. The development of the platform is guided by the MMSP-AV methodology, which supports the requirements definition, architecture design, implementation, and testing phases. The theoretical and research chapters are based on desk research conducted through academic databases and case study analysis of existing solutions.

The result of this thesis is a fully implemented analytical web platform supporting all four GUHA based procedures available in CleverMiner, providing data exploration, preprocessing, task management, and result visualisation in a single integrated environment. The platform is deployable both locally via Docker and on cloud infrastructure, and was verified through user acceptance testing conducted with students familiar with comparable tools. Code and documentation for the platform is available in Appendix A.

[1] General Unary Hypotheses Automaton

Objectives of the Thesis

Main Objective of this thesis is to design and implement a user-oriented analytical platform for the CleverMiner library, which will enable the involvement of domain experts in the data analysis process utilising CleverMiner. The platform will provide a frontend interface for data inspection, definition, execution, and management of mining tasks, with particular emphasis on the SD4ft-Miner method, as well as visualisation of results, while building upon existing implementations and extending them with new functionalities.

The main objective is fulfilled through five sub-objectives defined in following section.

SO1: Conduct a systematic survey of available solutions in the area of visualisation and interactive work with association and descriptive rules, identify their benefits and limitations, and assess the possibilities of leveraging existing implementations and their functionalities.

SO2: Analyse the CleverMiner library and its available API with respect to functionality, inputs, outputs, and possibilities for integration with a web-based platform.

SO3: Based on the survey, define functional and non-functional requirements and user stories in accordance with the chosen methodology. Design a system architecture that enables separation of the client and server layers and supports both local and remote task execution.

SO4: Implement a web application comprising a user interface, a backend layer, and integration with the CleverMiner package.

SO5: Verify the functionality and usability of the application in accordance with selected methodology through testing on sample datasets and evaluate the achieved results.

Follow-Up on Other Theses

This thesis directly follows up on two theses completed at Prague University of Economics and Business. The two theses were developed in parallel and did not influence one another. However, both advance the starting point of this thesis, as the author can draw on two existing implementations of the CleverMiner library.

The first thesis, by Vala, titled “Web Frontend for CleverMiner” (Vala, 2025), focused on creating a multi-deployable solution for the 4ft-Miner GUHA procedure, establishing both a backend and frontend architecture for data mining with that procedure. The second thesis, by Zedníčková, titled “Web and User Frontend for CleverMiner” (Zedníčková, 2025), focused on the CF-Miner procedure with particular emphasis on the frontend and the UI components used in rule mining.

This thesis builds upon both works. In the Analysis of Existing Solutions chapter, see Chapter 3, the author evaluates whether to extend these implementations or develop a new solution

from scratch, per the SO1 of this thesis.

The decision was made not to build upon the existing codebases, as extending either would prove insufficient for encompassing all four procedures supported by CleverMiner. The architectural changes required would be so substantial that designing a new architecture with all four procedures in mind from the outset is the more practical and sustainable approach.

Nevertheless, both theses are referenced throughout the chapters, as they provide established patterns and implementation insights that the author acknowledges and, where applicable, reuses or argues against in favour of an alternative approach better suited to the newly created platform.

1. Methodology

Since this thesis contains both research and software artefact development, it is necessary to establish methodologies that structure and guide these activities. This chapter provides a brief overview of the methodologies employed across the thesis, as each is documented in greater detail within its respective chapter.

The thesis is structured in accordance with the IMRaD methodology. The MMSP-AV methodology is additionally employed to guide the foundational chapters related to application development.

1.1 Theoretical Chapters

The Chapter 2 progressively builds the reader's knowledge in the field of association rules, which the author considers the theoretical foundation of the thesis. This research was conducted through desk research on academic libraries and databases such as Google Scholar, ResearchGate, and the ACM Digital Library.

The survey of existing GUI-based solutions carried out in Chapter 3 is conducted through a case study analysis of identified applications. Given that the author has prior experience with similar software products and builds directly upon two existing platforms, the scope of this section is intentionally focused on these solutions.

1.2 Application Development Methodology

The development of the software artefact is supported by the MMSP-AV [1] methodology, developed at Prague University of Economics and Business. This methodology is grounded in the OpenUP methodology and adapts it for teaching software engineering principles at the university level. The agile version created by (Kemr, 2016b) employed in this thesis extends the original MMSP created by (Rejnková, 2010) by introducing agile principles to the established framework. MMSP-AV is designed for projects lasting between six and twelve months, with a maximum team size of ten members, targeting non-critical systems (Kemr, 2016a).

The methodology defines a set of working products to guide documentation and development across four lifecycle phases. Not all products are produced in this thesis, as doing so would exceed its intended scope. MMSP-AV was selected as the development foundation both because it clearly defines all project development products and because the author is already familiar with the original MMSP methodology.

[1] https://chi.vse.cz/mmsp-av/

The agile version of MMSP-AV was selected because the nature of the problem was well suited to iterative, sprint-based development. The task backlog was gradually refined throughout the project rather than being fully defined upfront, as subsequent tasks frequently depended on the outcomes of preceding ones. For example, each GUHA procedure implementation was treated as a separate sprint, and the backlog was later extended to include data analysis features such as CleverMiner Guidance, Exploratory Data Analysis, and dataset preprocessing. This incremental approach allowed the scope of the platform to evolve organically as the implementation progressed.

The Table 1.1 describes planned iterations for the development project in this thesis.

Table 1.1
Phase and Iteration Plan

Phase	Iteration	Primary Goals	Planned Start
Inception	I1	Definition of problem and product vision, research of existing solutions, analysis of CleverMiner library, system specification, requirements, user stories, and architecture design.	October 2025
Elaboration	I2	Base project setup, backend and frontend architecture, Django and Celery configuration, database models, authentication.	November 2025
Construction	I3	Implementation of SD4ft-Miner procedure, result visualisation, backend API.	December 2025
	I4	Implementation of 4ft-Miner, CF-Miner and UIC-Miner procedures.	January 2026
	I5	Dataset module, Exploratory Data Analysis, CleverMiner Guidance, dataset preprocessing.	February 2026
	I6	Frontend refinement, documentation, Docker deployment setup, cloud deployment.	March 2026
Transition	I7	User acceptance testing, bug fixing, final deployment.	April 2026

The Chapter 4 applies MMSP-AV to define functional and non-functional requirements, identify the intended users of the platform, and supplement the specification with user stories.

These outputs serve as essential inputs for the subsequent development chapter. The chapter also presents the Product Vision, encompassing both the problem and product definitions.

While the Chapter 5 does not follow a specific MMSP-AV output template, it thoroughly documents all architectural decisions and evaluates the advantages and disadvantages of each selected option, in the spirit of the methodology's documentation standards.

The Chapter 8 follows on from the system development chapter and rigorously tests the application using the working product templates provided by MMSP-AV, including the test plan, test sets, test cases, and test results templates. As this part is important, MMSP-AV working templates are partially used and presented in appendices of this thesis.

1.3 Code Documentation and Storage

The codebase is versioned and maintained using Git and stored in a publicly available GitHub repository, the url can be found in Appendix A. As the author is currently the sole contributing developer, no formal Git flow is enforced. However, for significant or breaking changes, a feature branch approach is adopted with pull requests targeting the main branch. A dedicated `prod` branch is maintained for production deployments, enabling changes to be released in larger, more stable increments rather than on a per-commit basis.

2. Theoretical Framework

Theoretical framework chapter introduces the principal topics of this thesis as well as explains fundamentals of selected technology and Python library. These foundations will be used in Section 2.5 to introduce the CleverMiner Python package and show how the described procedures from Section 2.4 are implemented and how they are utilized within this system. This chapter formally completes the SO2 objective of this thesis.

2.1 Data Mining

The term data mining undergone notable semantic changes over the years, and the term itself is frequently used in rather loose or inconsistent manner (Jackson, 2002). Despite this variability in usage, it can be clearly defined as the process of identifying valid, novel, potentially useful and understandable patterns or correlations in datasets (Chung & Gray, 1999). The term data mining is often used with different meaning across many industries and is sometimes interchanged with Knowledge Discovery in Databases or KDD.

(Feng et al., 2006) argues that one of the most prominent and fitting descriptions of KDD is defined in an article from 1996 with the title "From Data Mining to Knowledge Discovery in Databases." (Fayyad et al., 1996). According to this perspective, the term KDD points to a broader process of discovering knowledge in data, while data mining only refers to a certain step in the whole KDD process (Fayyad et al., 1996). KDD as per Fayad encompasses the entire pipeline of data preparation, data selection, data cleaning as well as data explanation (Fayyad et al., 1996). This broader understanding of KDD is closely related to the CRISP-DM methodology, which similarly structures the data mining lifecycle phases and is introduced in greater detail in Section 2.2. The proposed analytical platform developed in this thesis aims to support this full lifecycle.

Although the term data mining is used in diverse contexts, it generally encompasses two key approaches to extracting useful information from data, distinguished by their primary objectives, whether the goal is to find patterns or model construction. Model construction approach mainly refers to building classification trees, clustering analyses and regression models for prediction. The second approach, which is central to this thesis, adopts pattern recognition perspective. Rather than constructing predictive models, it seeks to identify patterns that deviate from expected norms or reveal non-trivial relationships within the data (Jackson, 2002). This approach is commonly referred to as association rule mining and is further discussed in Section 2.3.

2.2 CRISP-DM Methodology

Prior to the adoption of standardized methodologies, the success or failure of data mining project was tightly coupled with the experience of users or teams conducting the analysis. To address challenges in data mining projects, a structured approach was devised to ensure that the business requirements, problems or goals could be systematically translated into data mining projects while maintaining methodological approach and effectiveness (Wirth & Hipp, 2000).

As a result, the CRISP-DM (Cross Industry Standard process for Data Mining) methodology was created to address the issues and state a clear path of how to carry out data mining projects (Chapman, 2000; Wirth & Hipp, 2000). Even though the methodology was released in the year 2000, it remains highly relevant and widely adopted in practice. CRISP-DM is still regarded as the de facto standard when working on data mining projects (Schröer et al., 2021).

CRISP-DM consists of six key stages that are briefly explained in the following paragraphs. Stages are fundamental for the proposed platform as the goal is to provide user with full capability of the CRISP-DM methodology in one platform.

Business Understanding
Before initiating the data mining project, it is essential to establish a clear understanding of the business domain that the researchers want to explore. Although this phase tends to be overlooked in many projects, particularly when there is pressure to proceed directly to technical analysis, it plays a critical role in ensuring project success. The business understanding phase involves defining clear objectives, identifying constraints, specifying success criteria, and translating business problems into analytical tasks. (Hotz, 2018)

Data Understanding
Building upon conclusions of the previous phase, the data understanding stage of CRISP-DM establishes a direct link between the defined business objectives and the proposed data mining method in next stages. In this part it is necessary to collect and describe the data if not done so already and perform exploratory data analysis. Typical activities include identifying hidden relationships, examining attribute distributions through visualizations such as histograms or distribution plots, and detecting anomalies. (Hotz, 2018)

Data Preparation
Real world datasets are often incomplete, inconsistent or otherwise unsuitable for immediate analytical use. Therefore, it is necessary to prepare the data to achieve better results. There is a common rule that states that this segment takes around 80 % of the total effort.

During this stage, relevant subsets of data are selected, including specific rows (records) and attributes (features). Data cleaning procedures are applied, such as imputing missing values (e.g., using mean or median substitution), removing duplicates, or correcting inconsistencies. It is also common to transform or engineer new attributes derived from existing ones to better

support the intended analytical method. (Hotz, 2018)

Modelling

The modelling phase focuses on selecting and applying appropriate analytical algorithms to the prepared dataset. Although often perceived as the most technically engaging part of the process, it may be relatively shorter compared to the data preparation stage.

In this phase, researchers typically experiment with multiple models or parameter configurations to determine which approach best fits the problem context and available data. Models are evaluated using appropriate technical metrics before proceeding to broader validation.(Hotz, 2018)

Evaluation

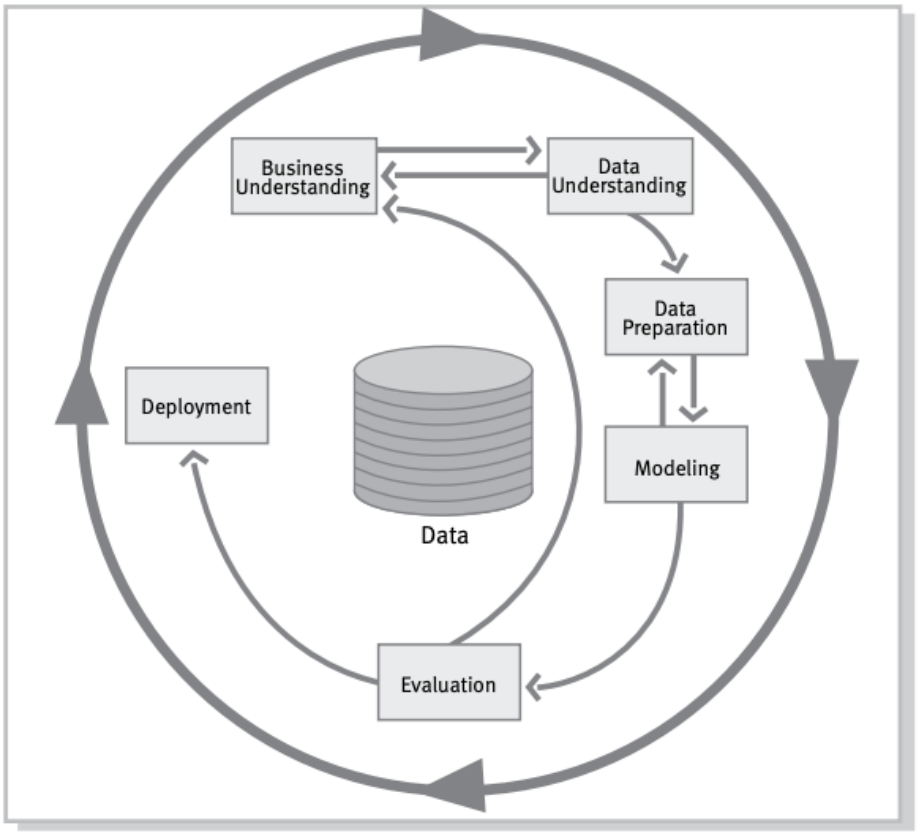
In this part the results of the model are checked in more general sense and are connected to the provided context of the business and the goal the data analysis strives to achieve. In this step all processes in the loop are evaluated and researchers often return to the data preparation phase to try new approach to the analysis. (Hotz, 2018)

Deployment

Deployment of the results often means creating reports of the found results and clearly showing and stating what might have been causing the problem defined in the business. However, this part can also mean deploying the solution to production in order to gain efficiency in business processes or to improve other business relevant metrics. (Hotz, 2018)

The Figure 2.1 shows the iterative nature of the CRISP-DM methodology. Often, as the results are evaluated, new ideas, challenges or topics might occur, which causes the next iteration. However, sometimes, it is also possible, that the prepared data and selected model just do not perform as per users expectation.

Figure 2.1
CRISP-DM



Source: (Chapman, 2000)

Each stage, except the first and last is implemented inside the platform. Data understanding is provided through showing preview of the data, allowing users to see missing values, their histograms or other relevant information. Data Preparation is done through prepared wizards in the platform, where users can fill missing values, create discretization strategy (binning) and proceed with dropping entire columns. Modeling consists of creating the configuration for the data mining project, with selecting correct rule mining procedure. Evaluation is then achieved through presentation of found association rules and their statistical relevance.

2.3 Principles of Association Rule Mining

The foundations for association rule mining were established in the ACM SIGMOD Conference in 1993 by Rakesh Agrawal, Tomasz Imieliński and Arun Swami. In their paper (Agrawal et al., 1993) authors introduced an efficient method to generate all important association rules from database of customer transactions (Agrawal et al., 1993).

Agrawal explained the association rule mining on a simple example, showcasing the strength

of such data analysis. The core data point was the basket data of what customers had bought together at a defined point of time. The time did not necessarily mean at one shop visit, but it could be over a longer period. From such data an illustration of association rule was defined with the statement: "90 % of transactions that purchase bread and butter also purchase milk." (Agrawal et al., 1993). With this rule, terms antecedent and consequent are introduced, where antecedent represents the first part of the rule and consequent is the underlining result. The percentage of transactions that represent this rule is called confidence (Agrawal et al., 1993).

To formally define the association rule Agrawal gives a clear explanation that is summarized and presented here.

Preconditions:

Let $\tau = \{I_1, I_2, \ldots, I_n\}$ represent a set of binary attributes that are called **items**.

Let T represent a database of transactions.

Association rules are represented by an implication $X \Rightarrow I_j$.

The X in the implication represents some items in τ and I_j is a single item in τ that is, however, not present in X, therefore X and Y are disjoint items:

$$X \cap Y = \emptyset$$

Source: (Agrawal et al., 1993)

The rule $X \Rightarrow I_j$ is satisfied in the defined set of transactions T with c as a confidence factor that lies in the given interval $\langle 0; 1\rangle$ if the count of transactions from T that satisfy antecedent X also satisfy the consequent I_j with at least $c\%$ of the transactions set T (Agrawal et al., 1993).

In addition to the formal definition of association rules, it is necessary to specify the constraints under which such rules are considered relevant. Given a transaction database T, the goal is typically not to generate all possible implications, but only those that satisfy predefined constraints. Agrawal et al. (1993) distinguishes between two primary types of constraints: **syntactic** constraints and **support** constraints (Agrawal et al., 1993).

There are also other measures relevant to quantify the significance of found rules. Among support and confidence, there is also lift, correlation and collective strength. However, it has been argued, that no single measure is consistently better than the others (Tan et al., 2002). It is also essential to interpret the results of the analysis with a certain caution. As emphasized by Tan et al. (2018): "The inference made by an association rule does not necessarily imply causality.". Association rules identify statistical co-occurrence patterns, but they do not establish cause-and-effect relationships. The fact that two items frequently

appear together in transactions does not imply that one item causes the occurrence of the other. Rather, association rule mining reveals correlations that may require further domain-specific interpretation.

The computation of support, confidence, and many other measures are based on the four-fold contingency table, see Table 2.1, which summarizes the co-occurrence frequencies of the antecedent and consequent.

Table 2.1
Four-fold Contingency Table

	consequent	¬consequent	$\sum$
antecedent	a	b	r
¬antecedent	c	d	s
$\sum$	k	l	n

To summarize this subchapter, association rule mining generates rules in if then format, and are often used to find patterns in the data, mainly when it comes to understanding relationships between items in the datasets. Rule mining is used in many industries and business problems like the already mentioned basket analysis from shops, but also in healthcare, targeted marketing, manufacturing, stock markets and recommendations engines (Salari, 2024).

2.3.1 Common Algorithms

As the amount of data stored in transactional databases grew, the original algorithm that was introduced by (Agrawal et al., 1993) and is now referred to as AIS algorithm was insufficient to mine large datasets. At the Eleventh International Conference on Data Engineering, a scientific paper was presented with introduction of new algorithm called SETM. This algorithm leverages SQL queries to perform candidate generation and support calculations directly within a relational database (Houtsma & Swami, 1995). Houtsma and Swami wanted to show, that an algorithm can be developed in simple language with using sorting and merge-scan joins, without creating a completely new black box algorithm. In 1994, one of the most popular algorithms for association rule mining was presented called Apriori (Agrawal & Srikant, 1994).

Before Apriori, the algorithms like AIS and SETM were making multiple passes through the whole dataset. The results, that have the minimum support from (are large enough) each pass are carried to the next iteration. The results from the n-1 passes are called seed in the nth pass and are used to generate new candidate itemsets. As this process is iterative, the stopping condition is that there are no other large itemsets present in the data (Agrawal & Srikant, 1994).

Apriori significantly reduced the number of generated candidate itemsets compared to earlier algorithms. As stated by Agrawal and Srikant (1994): “The basic intuition is that any

subset of a large itemset must be large." This principle, commonly known as the Apriori property or downward closure property, allows the algorithm to prune candidate itemsets efficiently. Specifically, if an itemset is not frequent, none of its supersets can be frequent. Unlike AIS and SETM, Apriori generates candidate itemsets using only the frequent itemsets from the previous iteration, without repeatedly scanning the entire database for unnecessary combinations.

2.4 The GUHA Method

Article from Hájek et al. (1966) titled "The GUHA Method of Automatic Hypotheses Determination", laid foundations of association rules nearly 30 years before Agrawal popularized it with the presented basket analysis (Rauch et al., n.d.). GUHA stands for General Unary Hypotheses Automaton and it is a method of offering hypotheses that are formulated and verified relations of the objects in the data (Hájek et al., 1966). In other terms, the method is generating as many as possible interesting hypotheses of the defined logical form that are supported by the data (Hájek et al., 2010). The GUHA method was at first considered as an exploratory analysis, where the idea was to not only validate the hypotheses but also to generate them (Rauch et al., n.d.).

2.4.1 GUHA Rule Structure

Let $P_1, P_2, \ldots, P_n$ be attributes describing objects in the dataset.

An **elementary conjunction** of length k from the interval $\langle 1; n \rangle$ is a conjunction of k literals where each attribute appears at most once. For example:

$$P_1 \wedge \neg P_3 \wedge P_7$$

An object satisfies an elementary conjunction if it satisfies all included literals.

Similarly, for **disjunction**:

$$P_1 \vee \neg P_3 \vee P_7$$

An object satisfies an elementary disjunction if it satisfies at least one of its literals. A **cedent** is either an elementary conjunction or an elementary disjunction of specified literals.

Source: (Hájek et al., 2010).

GUHA Hypothesis

A GUHA hypothesis has the form:

$$A \Rightarrow_p S$$

where:

- A is a cedent (antecedent),
- S is a cedent (succedent),
- $p \in [0, 1]$ is a threshold parameter.

The formula is considered true in the dataset if at least proportion p of objects satisfying A also satisfy S. Formally:

$$\frac{a}{r} \geq p$$

where:

- r is the number of objects satisfying A,
- a is the number of objects satisfying both A and S.

Additionally, the antecedent A is considered t-good if at least t objects satisfy it. A GUHA rule satisfying both conditions is denoted:

$$A \Rightarrow_{(p,t)} S$$

Source: (Hájek et al., 2010).

The conceptual similarity between GUHA implications and later association rules is evident, as both rely on measures analogous to support and confidence. However, unlike heuristic frequent pattern mining such as Apriori, GUHA originates from a formal logical framework aimed at systematic hypothesis generation.

Over time GUHA method was generalized and new procedures were developed. Six new procedures were developed at the Faculty of Informatics and Statistics at the Prague University of Economics and Business, that share the same bit string approach to analysis as the initial GUHA method (Hájek et al., 2010). These procedures have multiple usages in data mining as they do not only mine the association rules, but they also provide analysis for variety of patterns (Hájek et al., 2010).

The GUHA framework is foundational to this thesis, as the CleverMiner library implements these extended procedures. Consequently, the following subchapters introduce the individual procedures from a theoretical perspective and explain their core principles, preparing the ground for their practical implementation within the proposed analytical platform.

2.4.2 4ft-Miner Procedure

The 4ft-Miner procedure extends the earlier `ASSOC` procedure within the GUHA framework. It is designed for mining association rules based on **generalized quantifiers**. Unlike classical association rule mining, which typically evaluates rules using fixed measures such as support and confidence, 4ft-Miner evaluates rules using so-called **4ft-quantifiers**.

Structure of a 4ft-Miner Rule

The procedure mines rules of the form:

$$\varphi \approx \psi$$

and conditional rules:

$$\varphi \approx \psi / \gamma$$

where:

- φ and ψ are cedents (antecedent and succedent respectively),
- γ is an elementary conjunction (condition, for conditional rules),
- $\approx$ denotes a 4ft-quantifier.

The symbol $\approx$ specifies the type of dependency being tested between φ and ψ (Hájek et al., 2010).

Types of Quantifiers

The 4ft-Miner procedure implements multiple types of quantifiers, for example:

- Implication-like quantifiers (similar to confidence),
- Statistical hypothesis-based quantifiers,
- Above-average and below-average dependency,
- Other logical dependency relations.

Conditional Association Rules

In addition to standard rules, 4ft-Miner supports already presented conditional association rules of the form:

$$\varphi \approx \psi / \gamma$$

Such a rule evaluates the dependency between φ and ψ only within the subset of the dataset where condition γ holds. Formally, the **4ft-table** is constructed using only rows satisfying γ. This enables context-specific analysis.

Source: (Hájek et al., 2010).

2.4.3 CF-Miner Procedure

The CF-Miner procedure is a method designed for discovering patterns concerning the frequency distribution of categories of a single categorical attribute. While 4ft-Miner analyzes

relationships between two Boolean attributes using contingency tables, CF-Miner focuses on analyzing how the distribution of categories of an attribute behaves under certain conditions.

Structure of a CF-Miner Pattern

CF-Miner mines patterns of the form:

$$\sim R/\gamma$$

where:

- R is a categorical attribute,
- γ is a condition,
- $\sim$ is a CF-quantifier.

Source: (Hájek et al., 2010).

The pattern expresses that the frequency distribution of categories of attribute R satisfies a condition defined by the CF-quantifier, but only within the subset of data where γ holds (Hájek et al., 2010).

2.4.4 SD4ft-Miner Procedure

The SD4ft-Miner procedure extends 4ft-Miner by enabling the comparison of association rules across two groups. While 4ft-Miner evaluates whether a rule holds in a dataset (or under a condition), SD4ft-Miner evaluates whether the validity of a rule differs between two subsets of data.

Structure of an SD4ft-Miner Pattern

SD4ft-Miner mines patterns of the form:

$$\alpha \bowtie \beta : \varphi \approx^{SD} \psi/\gamma$$

where:

- α and β are Boolean attributes defining two subsets of data,
- γ is a Boolean condition,
- φ is the antecedent,
- ψ is the succedent,
- $\approx^{SD}$ is an SD4ft-quantifier measuring the difference.

Source (Hájek et al., 2010).

2.4.5 Other GUHA procedures

There are also other miners like KL-Miner procedure that is designed to analyze relationships between two categorical attributes under an optional condition. The last two procedures are SDKL-Miner and SDCF-Miner. These miners have the same relationship with KL and CF miners as SD4FT-Miner with 4ft-Miner (Hájek et al., 2010).

2.4.6 UIC-Miner Procedure

UIC Miner is a procedure developed for the CleverMiner library, outside of the original GUHA framework but based on the GUHA approach. It looks for circumstances under which a weighted improvement in category occurrence exceeds a specified threshold, compared either to a condition or to the entire dataset. (Little Big Company & Máša, 2026)

2.5 Analysis of the CleverMiner Library

The foundational principles of the association rules and GUHA procedures were introduced in the previous chapters, this chapter focuses solely on the CleverMiner python package. The goal of the analysis is to introduce how the package operates, what are its features and do a showcase of the miners in Python following the public documentation. This chapter works as an intermediary between the theoretical part and the main implementation of the proposed platform.

2.5.1 CleverMiner Overview

CleverMiner is Enhanced Association Rule Mining (eARM) package created by RNDr. Ing. Petr Máša Ph.D., that implements machine learning algorithms, specifically, three GUHA procedures (4ft-Miner, CF-Miner, SD4ft-Miner) and a GUHA based UIC-Miner. The package is written in Python and is available to download through a `pip` install command (Little Big Company & Máša, 2026). It is also important to note, that since the AI boom, the CleverMiner package with its analyses now belongs to the explainable AI (xAI) as the results are truly interpretable. The package provides users with an API, that can be called to get more functions from the library itself.

The package works with categorial attributes, so it is necessary to evaluate, that the data are of the correct type. To load the data into the CleverMiner instance, it is necessary to use `pandas DataFrame`[1] object. Users should also evaluate whether they need to insert the whole dataset to the instance, even if they do not want to use all the present attributes. The documentation states, that with loading many attributes, the computational time will

[1] https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html

increase. CleverMiner follows the definition of GUHA procedures regarding how the rules are build. Combination of attributes is called literals, and their conjunction or disjunction is called cedent. Therefore, the left and right side are called cedents and in the rule itself, the left-hand side is called antecedent, and the right-hand side of the rule is called succedent. Library allows to define the minimal lengths of the cedents as well as literals in a rule and many other quantifiers and conditions. (Little Big Company & Máša, 2026)

Literal types determine how attribute categories are combined when constructing conditions for rule mining. The subset type generates all possible category combinations within a specified size range. The *seq* type generates only consecutive sequences of ordered categories. The *lcut* and *rcut* types produce left and right threshold-like cuts of ordered categories, corresponding to lower-bound and upper-bound intervals. These strategies control the search space and allow the rule mining process to reflect the semantic nature of the analyzed attribute.

2.5.2 Example Usage

As is shown in the Source code 2.1 the CleverMiner instance is called with multiple parameters depending on the procedure that is specified in the string parameter `proc`. The CleverMiner API provides multiple function calls to allow for easy usage when showing results, printing histograms and presenting four-fold tables. Some of them are explained in the following list.

Source code 2.1

CleverMiner Example Usage (Little Big Company & Máša, 2026)

```
import pandas as pd
import sys
from cleverminer import cleverminer

# prepare panda dataset here

clm = cleverminer(df=df, proc=<procedure>,
quantifiers = <quantifiers>,
ante = <cedent>,
succ = <cedent>,
cond = <cedent>,
target = <varname>)

cleverminer.print_rulelist(clm)
```

`print_summary()`

: Provides a compact processing report (elapsed time, number of verifications/tests performed, number of rules found). Useful for describing performance and giving readers a quick overview of a mining run.

`get_rulecount()`

Returns the number of discovered rules in the result set. This is the simplest way to quantify output size and compare configurations (e.g., stricter thresholds → fewer rules).

`get_ruletext(rule_id)`

Returns a human-readable textual representation of a specific rule. Great for reporting example rules in the thesis without dumping full internal structures.

`get_fourfold(rule_id, order=0)`

Returns the contingency (four-fold) table underlying a rule, i.e., the core evidence used by 4ft-based quantifiers.

3. Analysis of Existing Solutions

Third chapter addresses objective SO1 and directly builds upon the foundations build in the previous chapter by researching applications that offer GUI based data mining solutions to its users. The reseach is not only focused on the GUHA implementations but rather on association rules in general.

The architectural choices as well as some functionalities identified in this analysis will serve as an input for the subsequent Chapter 4, where they are reflected in the formulation of functional and non-functional requirements.

3.1 Analysis Overview

This chapter contains research of applications that solve similar issues as the proposed platform. The research is guided by a qualitative approach to research using the Case study strategy as the main analytical framework. The case study approach comprises of an in-depth exploration of a program in this case a computer application (Creswell, 2009). As the field of association rule mining is wide, the research is conducted by a method of gradually building the searched term using the building blocks method (Fabián, 2012) that is connected with a Google Boolean search using AND/OR operators (Karch, 2021). The searched terms and key words are listed in the following list:

- Association rules
- Data Mining
- Desktop application
- Web application
- GUHA procedures
- CleverMiner package

The keywords presented were used by the author when searching the platforms mentioned in Chapter 1, following the query: *Association Rules AND Data Mining AND Desktop Application*. The results are later narrowed down, by using other keywords or by explicit selection of appropriate solutions by the author through abstract analysis or relevance to the thesis goals.

The main objective of this research is to find solutions that implement association rules and can provide ways of preprocessing the data before the analysis is run. As GUHA is a niche method, the author does not look necessarily for the solutions implementing this method. However, two authors (Zedníčková, 2025) and (Vala, 2025) already implemented parts of CleverMiner package in their theses. The proposed platform, in later stages of this thesis, partially benefits from the previous implementations, these solutions are also presented in this part to carefully evaluate, what will and what will not be used in the newly proposed

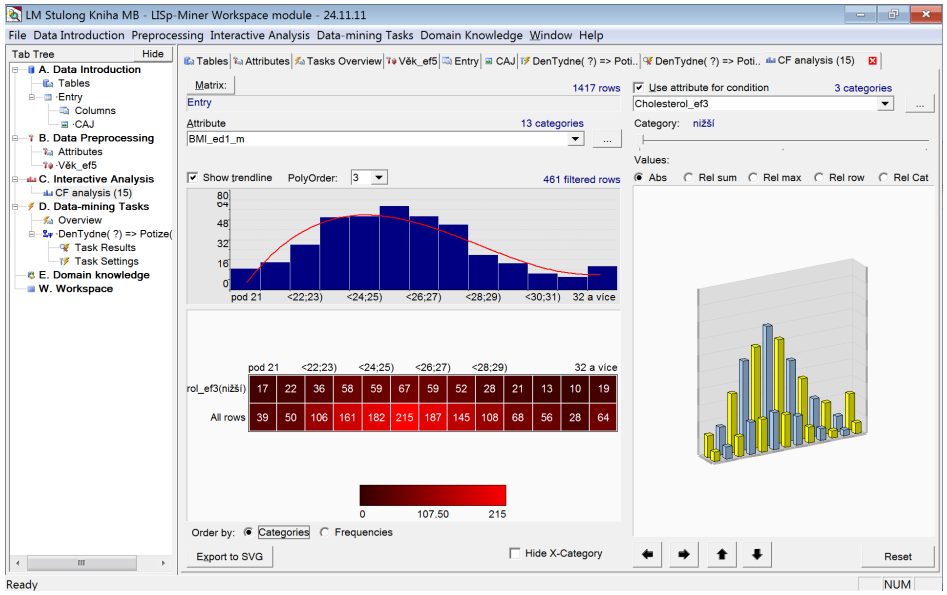
platform further fulfiling the SO1. Another system that is added to the research explicitly is LISp-Miner as this system was already discovered when general GUHA procedures were introduced in the Section 2.4.

3.2 LISp-Miner

LISp-Miner is a data mining software developed from 1996 at the Faculty of Informatics and Statistics at Prague University of Economics and Business to facilitate learning and researching the field of association rules, GUHA methods and to support the pattern data mining (Hájek et al., 2010; Rauch et al., n.d.). LISp-Miner system implements six generalized GUHA procedures described in previous chapter as well as other analyses regarding contingency tables.

The system is available as Windows desktop application. The dominant navigation in the applications is its sidebar providing access to data introduction, data preprocessing and the analysis overview as is shown in the Figure 3.1. Application also provides tabs like in a web browser, where users can manage their open tasks.

Figure 3.1
LISp-Miner - Application Overview

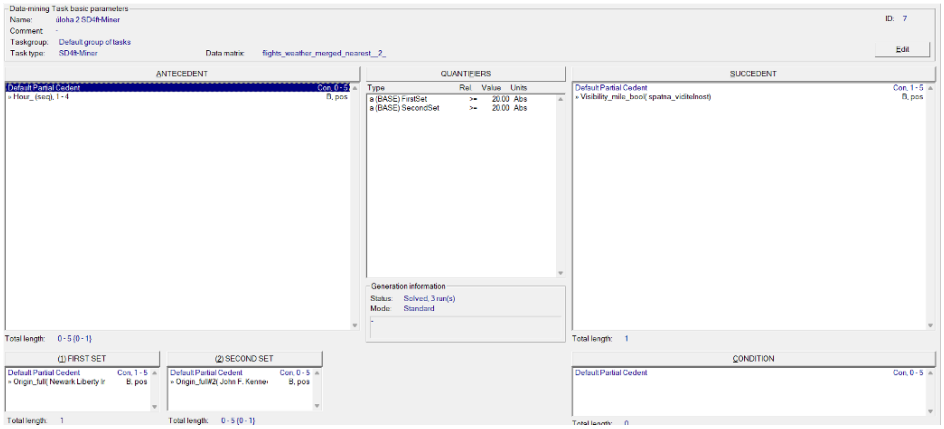


Note: Source: (Rauch, 2015)

Figure 3.2 introduces how creating new analysis works, the system provides new tab, where the rule is intuitively divided into antecedent, succedent and its conditions. The form configuration in Figure 3.2 belongs to the SD4ft-Miner procedure. Other procedures will however offer different possibilities regarding their configuration. For selecting the attributes into antecedent and succedent lists, user needs to click on the buttons on top of these lists. Then

a new window with attribute list is provided, where user selects attributes to be used in the analysis.

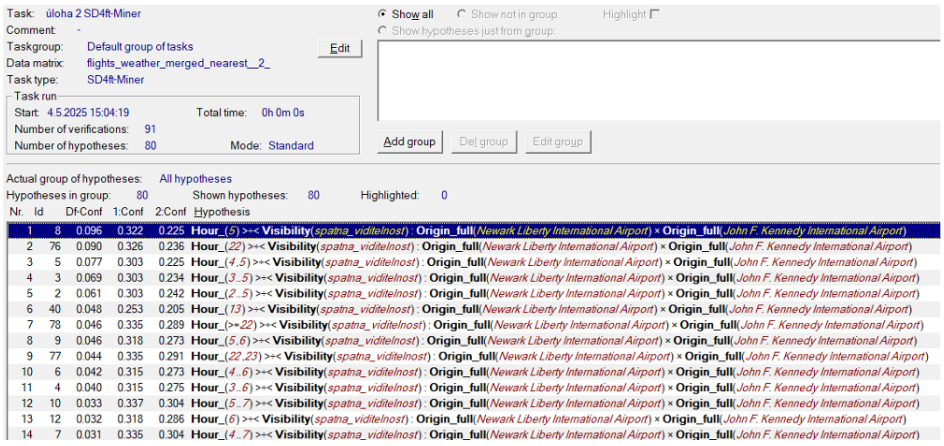
Figure 3.2
LISp-Miner - Analysis Configuration



Note: Source: Author, screenshot from the LISp-Miner system.

After system finishes its mining process a new window, as shown in Figure 3.3 , is presented to the user with list of rules that were found. System also provides important statistical measures about each rule and about the verification process in general.

Figure 3.3
LISp-Miner - Analysis Results



Note: Source: Author, screenshot from the LISp-Miner system.

Each rule is accompanied by additional statistical information and is presented in the characteristic IF-THEN format. When selecting certain rule, new tab is opened with detailed results for the rule. Figure 3.4 shows a four-fold table for rule from 4ft-Miner and a 3D result of the same data.

Figure 3.4
LISp-Miner - Rule Detail

Note: Source: Author, screenshot from the LISp-Miner system.

LISp-Miner provides a comprehensive set of features for data preparation and configuration of GUHA procedures. The system offers extensive parameterization options, enabling advanced users to fine-tune the analytical process according to specific requirements.

However, despite its functional richness, the user interface may present challenges in terms of usability. The application is not always intuitive, and the volume of displayed information can be overwhelming, particularly for users with limited prior experience in data mining systems. As a result, initiating a new analysis or configuring procedures may require a significant learning effort.

From a design perspective, the visual style of the application reflects earlier software design paradigms and may be perceived as outdated. Nevertheless, certain architectural elements such as the use of a persistent sidebar for navigation remain effective and demonstrate good usability principles by providing clear structural orientation within the application.

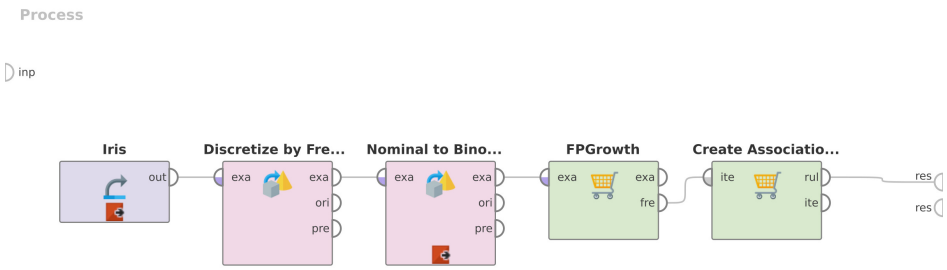
3.3 RapidMiner

RapidMiner is a data mining desktop platform that provides not only association rule mining, but also many other methods, like classification and neural nets. As this is a commercial platform it does not provide access to GUHA generalized methods but rather uses algorithms like FP-Growth for association rule mining (AltairEngineering, n.d.).

The platform provides multiple ways of preprocessing data and does not expect extensive

knowledge from the user in the data analysis field. The app works with so-called operators that are present in the Figure 3.5. Users work with these operators to create a workflow, where the data will flow through the defined operators until the final one is reached, which produces the output to be presented to the user.

Figure 3.5
RapidMiner (AltairStudio) - Flow Overview



Note: Source: (AltairEngineering, n.d.).

3.4 Implementation by Vala

Lukáš Vala implemented a web platform for mining with 4ft-Miner (Vala, 2025). The system is focused on this one main method, therefore, it does not contain some specific design choices, that are needed, when other procedures with different configurations come to the implementation scope. As of the time of writing this thesis, Valá's application is not currently available on the internet. There is a tutorial on how to set up the application locally, however this proved difficult, even when the application can be started easily with docker as the docs [1] state, it is impossible to log in to the application as it uses Google OAuth login and the application is missing key token in order to allow sign in. Therefore, the application is described purely on the results from Vala's thesis and not by hands on experience.

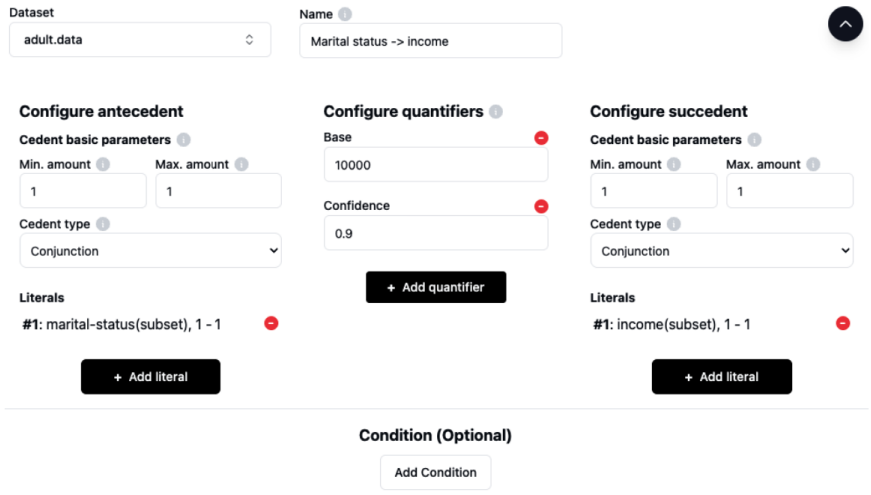
Vala's application is well structured and uses top bar for navigation. Procedure is configured through a single step configuration form, where cedents, and conditions as well as quantifiers are defined. The form in Figure 3.6 uses a simple black and white styling and works well with white spaces. When adding new literal to the analysis, a new dialog window is opened to show the configuration options regarding the literal for analysis.

Rules are shown through a table with statistical data, like AAD, confidence and Base, and the rules themselves. The data can be ordered by the statistical columns and when rule is clicked a detail dialog is opened in a Figure 3.7 to display a greater detail and also provide visual aid generated from the CleverMiner library.

[1] https://github.com/lukiina42/clever-miner-web-wrapper

Figure 3.6

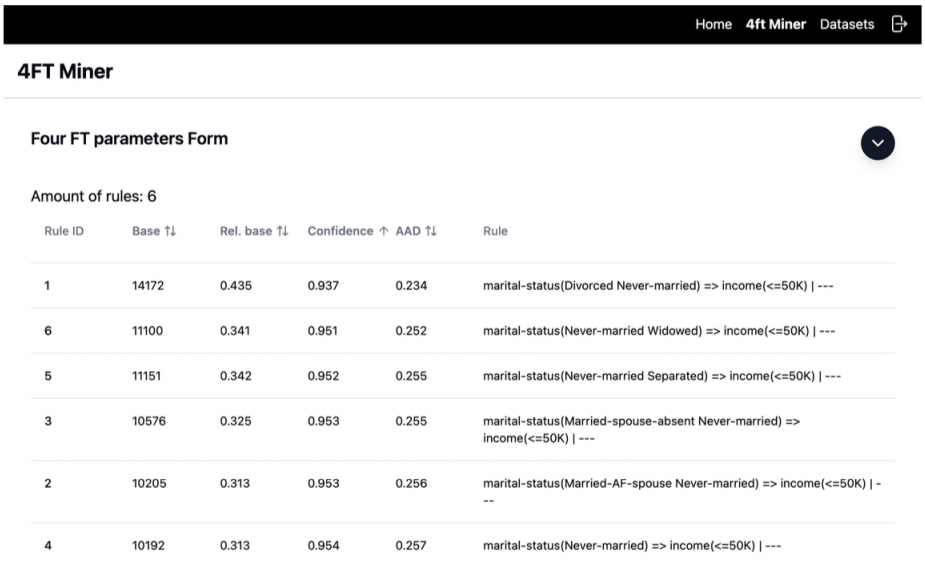
Vala's Implementation - Procedure Configuration



Note: Source: Screenshot from (Vala, 2025).

Figure 3.7

Vala's Implementation - Results View



Note: Source: Screenshot from (Vala, 2025).

Vala's application feels modern and provides all the configurations provided by the CleverMiner regarding the 4ft-Miner. However, no other procedures are implemented, and the application is built only around the 4ft-Miner. There is no dataset preprocessing available, neither users can collaborate actively on the same project. The rules detail dialog and presentation table are functional but would benefit from more engaging style.

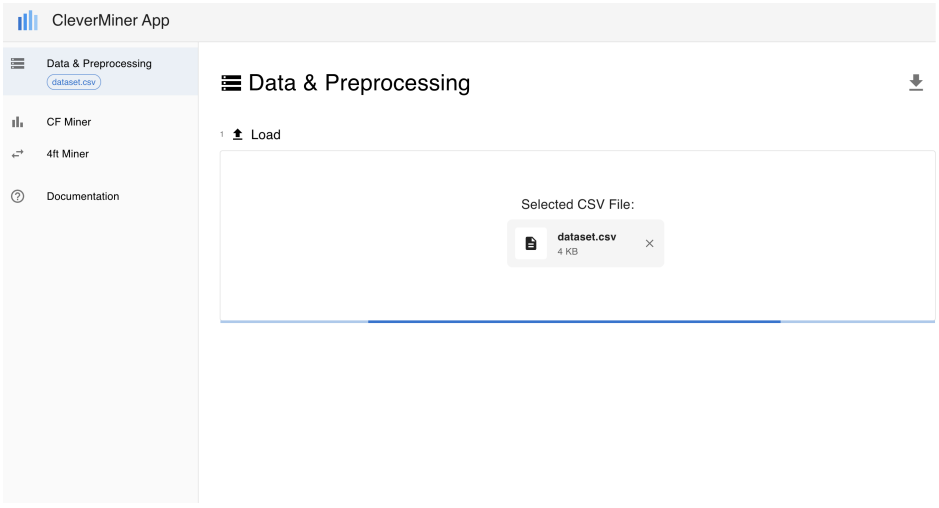
Regarding the backend, Vala does not use any approaches to distributed computing regarding the CPU bound data analysis, as he only describes some viable solutions like using microservices to delegate the analyses to other machines. This topic is later thoroughly discussed in Section 5.5.

3.5 Implementation by Zedníčková

Last application that is presented in the research is Zedníčková's approach to making the frontend for CleverMiner. In contrast to Vala, Zedníčková's app is publicly available on the internet without any login and provides an easy way to operate without any data or account, as it has a demo dataset. Zedníčková's thesis was mainly focused on creating an engaging frontend for the app and was not focused on the backend of the application (Zedníčková, 2025). Equally as Vala's approach, this application implements only one procedure, and that is the CF-Miner, however it also provides data preprocessing options in the datasets tab. The preprocessing consists of creating categorical data from the attributes using ordinal or nominal data preparation approach.

Zedníčková's app is domintated by the sidebar that provides the main navigation for the app, allowing access to datasets and miners as shown in the Figure 3.8.

Figure 3.8
Zedníčková's Implementation - Data Upload

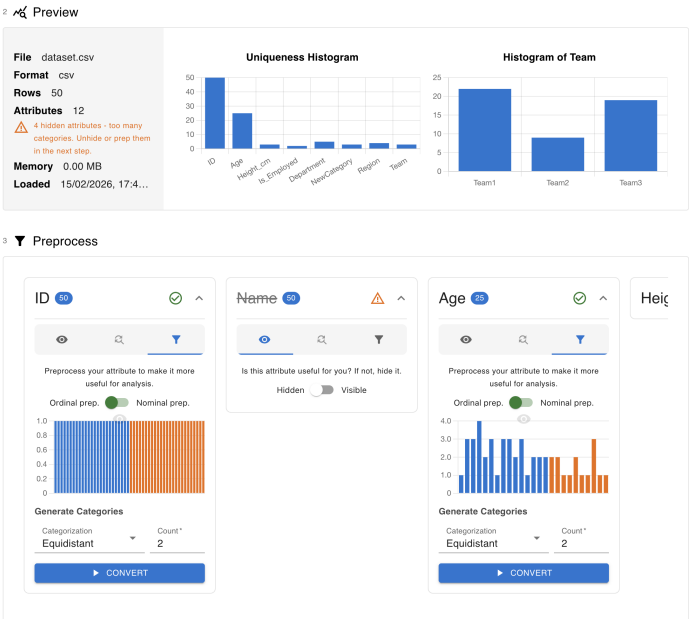


Note: Source: Author.

The dataset preprocessing in Figure 3.9 is clearly structured and provides histograms for data exploration step. The application provides guidance on whether the attribute is useful for the CleverMiner analysis by stating whether the attribute is categorical enough or not by checking the uniqueness of the items in the attribute. Description of each attribute is presented in a horizontal view with cards, that provide the information about the attribute.

Figure 3.9

Zedníčková's Implementation - Data Preprocessing



Note: Source: Author.

The application then follows this horizontal/vertical scroll pattern for other parts as well. The CF-Miner configuration in Figure 3.10 follows this particular design system with sections called *Observe and Conditions*. The most important section Conditions allows for CF-miner configuration with selecting the attributes, their parameters and also the target attribute since CF-Miner uses target for analysis.

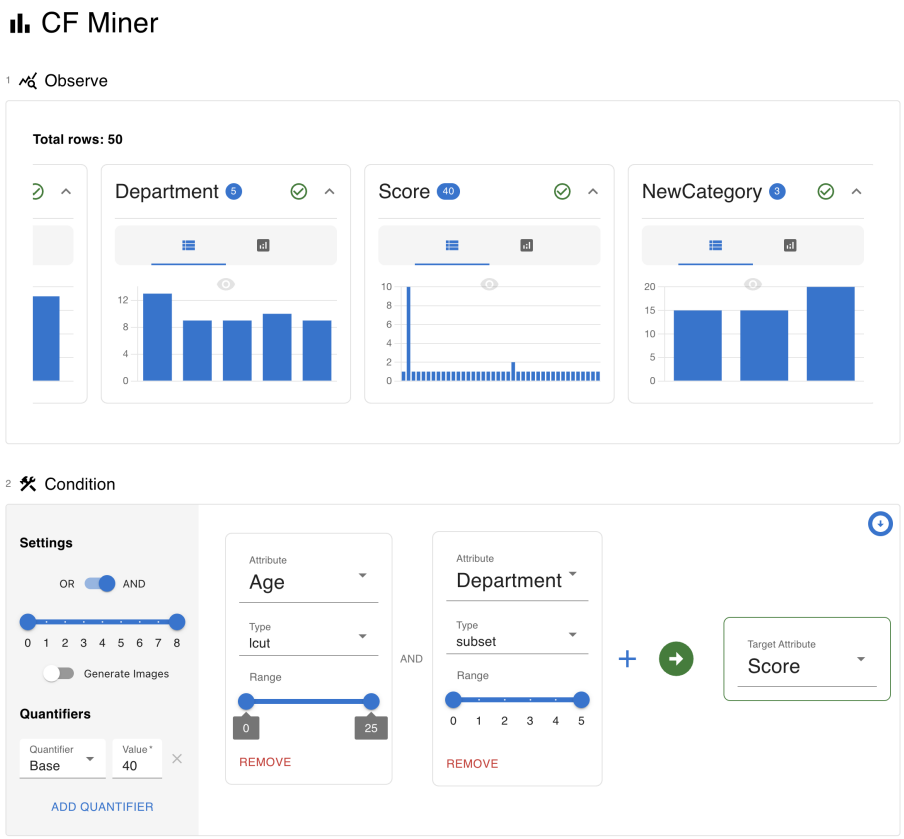
Rules are also presented in cards with horizontal scroll. Boxes present the cedents and histogram are presented to represent the target attribute. Application also provides a way of showing the raw output from the CleverMiner library as well as provide a way of exporting the data. Results part is shown in Figure 3.11.

The application is much more sophisticated regarding the frontend part of the application than already presented solutions. However, the application can sometimes feel a bit unintuitive, especially with the horizontal scroll regarding the conditions and the results. But overall, the application provides great visual components, like the base cards for the attributes.

As was stated, the application is mainly about building the frontend and not the backend. The backend build in FastAPI does not provide a way to sign up or store the analyses. The application, like Vala's, does not provide a way to work with distributed systems and the server can be overwhelmed. This is also presented when analysing big datasets with many conditions in the analysis, the application shutdowns the mining process, if it has not finished

Figure 3.10

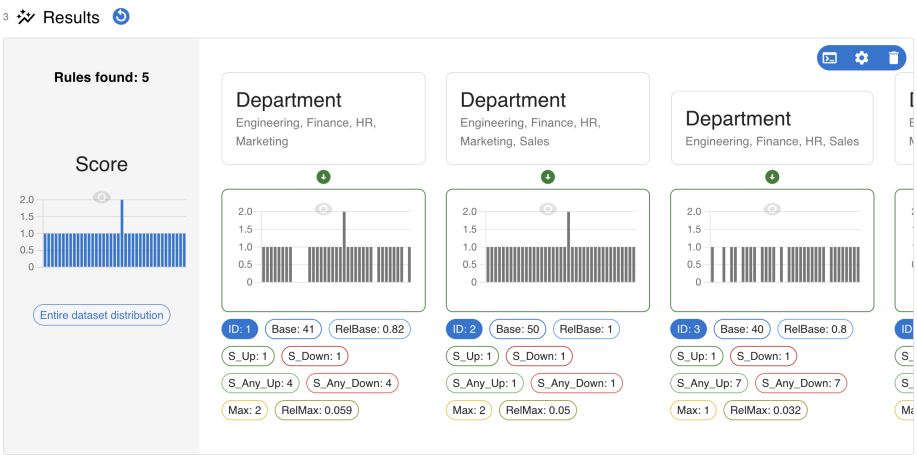
Zedníčková's Implementation - Analysis Setup



Note: Source: Author.

Figure 3.11

Zedníčková's Implementation - Results View



Note: Source: Author.

in 30 seconds. The dataset is also never stored in the backend but is sent every time the analysis is started. (Zedníčková, 2025)

3.6 Research Results

Described applications demonstrated, that association rule mining can be done in an intuitive and easy to use manner, as Zedníčková, and Vala showed in their theses. Integration with CleverMiner is possible but can cause some issues regarding the scalability of these platforms. However, the main outcome is that neither of the new solutions implemented the full CleverMiner library and also did not provide satisfactory solution to the heavy CPU-bound tasks that CleverMiner consists of when running the analysis.

Most importantly, in accordance to the thesis goals stated in Introduction, see Section , author states that these applications can be extended, but it would prove more difficult than creating a new architecture with more GUHA procedures in mind from the very beginning. The one procedure approach is too short-sighted and would prove difficult when extending the application further.

LISp-Miner provides all the necessary configurational steps as well as implements generalized GUHA procedures. However, the system feels a bit outdated and can be overwhelming to new users. RapidMiner is a full data mining platform, that implements everything in an easy-to-use intuitive way and also works as a desktop app. However, it does not provide an implementation of GUHA procedures.

Therefore, the proposed application fits perfectly into the current state, as it will conclude the implementation of CleverMiner to a usable, intuitive data mining platform, that implements generalized GUHA based procedures (available in CleverMiner) and also provides data preprocessing so that users do not have to have any knowledge of Python and subsequent libraries in order to work with this package.

4. System Specification

The system specification chapter, that addresses the SO3, defines the boundaries within which the proposed analytical platform is designed and implemented. The primary objective of this chapter is to ensure that the resulting solution systematically addresses the identified requirements and aligns with the expectations of the intended end users of the platform. This chapter therefore represents a key component of the thesis, as it establishes a clear link between the theoretical analysis and the practical implementation followed in next chapters.

The chapter begins with MMSP-AV vision and description of target users and their usage context. This analysis enables a focused selection of requirements that are both beneficial for the analytical platform and feasible within the scope of this thesis. Particular emphasis is placed on the formulation of system requirements, which describe how the objectives of the system are to be achieved in accordance with established principles of requirements engineering (Laplante et al., 2022).

The requirements are divided into functional and non-functional categories and are specified using the MMSP-AV methodology. While functional requirements often recieve primary study, the non-functional requirements presented in Subsection 4.3.2 are equally important, as the proposed analytical platform includes features such as long-running data analysis tasks, that may significantly affect performance, reliability and usability. In accordance with the selected methodology, the chapter further introduces user stories and a description of the user interface and user experience design supported by wireframes. The chapter concludes with a decsription of a conceptual model for database and suggested website map.

4.1 Vision

In accordance with the MMSP-AV methodology, this chapter opens with the Vision working product. This working product is centered around two core outputs: the Problem Definition and the Product Definition. The methodology prescribes that this discipline is carried out at the beginning of the project, in order to clearly establish the problem that the proposed product is intended to solve. The Vision working product also contains descriptions of stakeholders and product requirements, however, in the interest of maintaining the scope of this thesis, these outputs are skipped. As the platform is developed solely by the author under the supervision of the thesis supervisor, a formal stakeholder analysis would add limited value, and the requirements are defined in sufficient detail in the following chapters.

4.1.1 Problem Definition

Table 4.1
Problem Definition

Problem Definition	The CleverMiner library, which implements GUHA based procedures for enhanced association rule mining, lacks a user-friendly graphical interface. Working with the library requires knowledge of Python and several additional packages, making it inaccessible to domain experts without a programming background.
Who the Problem Affects	Domain experts and analysts who wish to perform data mining using GUHA procedures but lack the technical programming knowledge required to operate the CleverMiner library directly. Additionally, students at Prague University of Economics and Business who use GUHA procedures in coursework and research are affected by the absence of a modern, accessible interface.
Impact of the Problem	Domain experts are dependent on data analysts or developers to conduct analyses, reducing their autonomy and limiting the practical applicability of the CleverMiner library. Existing GUI-based alternatives either do not implement available GUHA procedures, are obsolete, or are closed, commercially licensed platforms that cannot be extended.
Benefits of a Successful Solution	A successful solution will allow domain experts to independently perform the full CRISP-DM data analysis workflow, including data exploration, preprocessing, task definition, execution, and result interpretation, without requiring knowledge of Python or any other programming language. The platform will broaden the reach of the CleverMiner library and contribute to the democratisation of data analytics.

4.1.2 Product Definition

Table 4.2
Product Definition

Name of the IS/ICT	Analytical Platform Integrating CleverMiner
For	Domain experts, researchers, and students who need to perform association and descriptive rule mining using GUHA procedures without requiring programming knowledge. The platform targets both technically limited users seeking self-service analytics and academic users at Prague University of Economics and Business.
Which	The platform provides an integrated environment for the complete data analysis workflow: dataset upload and exploration, data preprocessing and discretisation, definition and execution of four GUHA based mining procedures, visualisation and interpretation of discovered rules, and both local and cloud-based deployment options.

4.2 System Actors and Usage Context

This subsection clearly defines the users that the platform is intended for and further describes the usage context that will be important in following chapters.

4.2.1 System Actors

The target user groups of the proposed system are largely consistent with those indentified in the theses by Vala (Vala, 2025) and Zedníčková (Zedníčková, 2025). However, the scope of the platform is extended by the fact that the proposed solution is designed as a fully functional analytical platform that does not require programming experience from its users. For this reason, the original actor definitions are adapted and refined, while preserving conceptual continuity with the existing research.

University students and academic staff As the system is developed at the same institution where the extended GUHA procedures originated, a primary target group consists of university students and academic staff engaged in subjects related to association rule mining and GUHA based data analysis. For these users, the platform serves both for educational and exploratory purposes, enabling hands-on interaction with GUHA procedures without the need to directly work with the underlying programming interfaces.

General public and industry users By supporting the CRISP-DM data analysis life cycle, the platform significantly lowers the entry barrier to the CleverMiner library and its procedures. This increased accessibility makes the platform suitable not only for academic users, but also for domain experts and practitioners from industry who seek interpretable and explainable data mining results. As a result, the proposed system enables new use cases beyond the original academic scope of the library.

4.2.2 Usage Context

The proposed system defines a single global user role with full access to the platform's functionality. Introducing additional global roles is considered beyond the scope of this thesis. Nevertheless, role differentiation is applied within the context of collaborative projects. Each project distinguishes between a project administrator, who is responsible for managing project memberships, and project members (editors), who are allowed to create and modify tasks and execute runs within the project.

The platform is primarily intended for use on desktop devices. Due to the analytical nature of the system and the complexity of the presented information, usage on mobile devices is not considered a primary use case, as limited screen space would significantly reduce usability.

4.3 Requirements

The specification of software requirements is a fundamental prerequisite for a successful system implementation, as it provides a shared understanding of the system's objectives and serves as a reference throughout the development process (AbdulRahman, 2025). In this thesis, the requirements are formulated in accordance with the MMSP-AV methodology. Although the formal requirement templates defined by the methodology are not applied, their underlying structure and principles are preserved.

Each requirement is described using a consistent set of attributes, namely a unique identifier, a requirement name, a comprehensive textual description, and a priority value. The priority scale ranges from 1, denoting the highest priority, to 5, denoting the lowest priority (Kemr, 2016a). References to specific use cases are intentionally omitted, as the requirements are further elaborated and contextualized through user stories in a subsequent section.

The requirements defined in this chapter are partially based on the requirements presented in the theses by Vala (Vala, 2025) and Zedníčková (Zedníčková, 2025), where conceptual overlap exists. However, they are extended and refined to reflect the broader scope of a fully featured analytical platform. Additional requirements were derived from the comparable solutions research through an analysis of existing tools and platforms from Chapter 3. Furthermore, the requirement set was informed by consultations with students who had previously completed courses related to association rule mining and GUHA based data analysis.

4.3.1 Functional Requirements

Functional requirements specify the behavior of the system by defining the features and services that the system is expected to provide to its users in accordance with the selected MMSP-AV methodology (Kemr, 2016a).

Unless stated otherwise, the functional requirements defined in this section apply to authenticated users of the platform. The only exception is requirement FR-01, which covers user registration and authentication and is therefore accessible without prior login.

The chapter is structured to first provide a tabular overview Table 4.3 of the requirements using a table with all the attributes besides the description which follows after the table.

Table 4.3
Overview of Functional Requirements

ID	Requirement Title	Priority
FR-01	User Registration and Authentication	4
FR-02	Dataset Upload and Dataset Overview	2
FR-03	Dataset Analysis, Guidance, and Preview	2
FR-04	Dataset Preprocessing	3
FR-05	Mining Task Creation and Procedure Configuration	1
FR-06	Quantifier and Logical Structure Configuration	1
FR-07	Task Persistence and Execution Control	2
FR-08	Visualization of Mining Results	1
FR-09	Export of Mining Results	3
FR-10	Task Execution and Run Management	3
FR-11	Project based Collaboration	2
FR-12	Project Membership Management	2
FR-13	Access and Participation of Project Members	3
FR-14	User Notification Service	3
FR-15	Search and Filtering of Analytical Artifacts	3

FR-01: User Registration and Authentication
The system shall allow users to register and authenticate in order to access the platform. The system should not complicate the login for users using the local version of the application with the set up of an authentication through an external identity provider. Therefore, the application should provide a simple login and register forms for users to use.

FR-02: Dataset Upload and Dataset Overview
The system shall allow users to upload datasets to the platform for subsequent analysis. The platform shall support datasets provided in the CSV format, which is selected due to its widespread adoption and compatibility with the underlying data processing components. User should be allowed to specify the delimiter in the dataset, with default set to ";". The

system should further provide an overview of all datasets uploaded by the user, enabling efficient navigation and selection of datasets for analysis tasks.

FR-03: Dataset Analysis, Guidance, and Preview
After a dataset is uploaded, the system shall provide an automated exploratory analysis of the dataset. This analysis shall include a structured description of individual attributes, their inferred data types, and recommendations regarding potential data transformations, such as discretization (binning) or the handling of missing values.

The system shall further provide a dataset preview, allowing users to inspect a subset of the uploaded data in order to verify its correctness and structure.

In addition, the system shall offer guidance related to the CleverMiner library by suggesting suitable target attributes for selected mining procedures, based on the characteristics of the uploaded dataset.

To support domain understanding, the platform shall also present basic descriptive visualizations that summarize key properties of the dataset and assist users in interpreting the underlying data.

FR-04: Dataset Preprocessing
The system shall provide users with functionality for preprocessing dataset attributes in order to improve the suitability of the data for analysis using the CleverMiner library. In particular, the system shall support attribute discretization (binning) to address cases where numeric attributes contain a high number of distinct values.

The system shall further allow users to handle missing data by providing common imputation strategies, enabling the dataset to be transformed into a structure appropriate for subsequent mining procedures.

In addition, the system shall allow users to remove selected columns from the dataset in cases where their analytical relevance is limited or where they negatively affect data quality.

FR-05: Mining Task Creation and Procedure Configuration
The system shall support the creation of mining tasks by implementing GUHA based procedures provided by the CleverMiner library, namely UIC-Miner, SD4ft-Miner, CF-Miner, and 4ft-Miner.

The system shall allow users to select an appropriate mining procedure for a given uploaded dataset and to configure a new task accordingly. During task creation, the system shall provide input fields for specifying descriptive metadata, such as a task title, enabling users to clearly identify and manage tasks in later stages of analysis.

FR-06: Quantifier and Logical Structure Configuration
The system shall provide full support for quantifiers and related configuration options available in the CleverMiner library. In accordance with the selected GUHA procedure, the

system shall offer an intuitive mechanism for specifying antecedents, succedents, or attribute sets required for task definition.

The system shall further allow users to define structural constraints of the task, including the minimum and maximum lengths of cedents, as well as the logical operators used to combine attributes, such as conjunction and disjunction.

FR-07: Task Persistence and Execution Control
The system shall allow users to persist newly created mining tasks and control their execution. During task creation, the system shall provide options to either save and immediately execute a task, or to save the task without execution for later use, including the option to retain the task in a draft state.

All saved tasks shall be accessible through a dedicated task overview, enabling users to revisit, modify, and execute previously defined tasks and to continue their work from the point at which it was interrupted.

FR-08: Visualization of Mining Results
The system shall provide clear visual representations of the results produced by completed mining tasks. As the primary value of the proposed platform lies in the interpretability of discovered patterns, the provided visualizations shall be designed to be intuitive and easy to interact with.

The system shall offer its own visualization mechanisms for presenting discovered association rules, including graphical representations and four-fold contingency tables for individual rules, enabling users to analyze the statistical properties and validity measures of the results.

In addition, the system shall allow users to view visual outputs generated directly by the CleverMiner library Section 2.5, ensuring compatibility with existing representations and supporting advanced or comparative analysis.

FR-09: Export of Mining Results
Upon completion of a mining task, the system shall allow users to export the discovered association rules and their corresponding visual representations in a format suitable for presentation or further analysis.

The system shall also support the export of visual outputs generated directly by the CleverMiner library, ensuring that users can retain and reuse both platform-specific and native representations of the mining results.

FR-10: Task Execution and Run Management
The system shall allow defined mining tasks to be executed multiple times while preserving their identity. Users shall be able to modify task parameters between executions without changing the task's name or descriptive metadata.

To support this functionality, the system shall introduce the concept of runs, where each run

represents a single execution instance of a task with a specific configuration and resulting output. This separation shall enable users to compare results across multiple executions of the same task.

FR-11: Project based Collaboration
The system shall support collaborative work through the introduction of dedicated projects. A project shall serve as a shared workspace in which users can create mining tasks and share their execution results (runs) with other project members.

FR-12: Project Membership Management
The system shall allow project creators to manage project membership by inviting new users to join a project as project members.

FR-13: Access and Participation of Project Members
Upon becoming a member of a project, the user shall be granted access to all datasets, tasks, and runs associated with the project.

The system shall provide project members with appropriate mechanisms for exploring project-specific datasets, tasks, and execution results. In addition, project members shall be allowed to create new mining tasks within the project context and to execute these tasks, thereby contributing new runs to the shared workspace.

FR-14: User Notification Service
The system shall provide a notification service that informs users about significant events occurring within the platform. In particular, the notification service shall notify users when a mining task execution (run) has successfully completed or has failed. Notifications related to task execution shall be delivered in near real time upon completion of the corresponding run.

FR-15: Search and Filtering of Analytical Artifacts
The system shall provide search and filtering capabilities for lists of analytical artifacts. These mechanisms shall enable users to efficiently locate relevant items within the platform.

4.3.2 Non-Functional Requirements

The non-functional requirements are classified into four categories derived from the FURPS quality model, namely usability, reliability, performance, and supportability. This classification follows the interpretation of the FURPS model as described by Eeles (2001) and is fully compatible with the MMSP-AV methodology adopted in this thesis.

Each identified non-functional requirement is assigned to one of these categories in order to clearly express the quality attributes expected from the proposed system. Similar to functional requirements, non-functional requirements are described using a consistent set of attributes, including a unique identifier, a requirement name, a detailed description, and a

priority level.

Usability

This category focuses on how the user interacts with the platform and how the user interface in the platform is consistent to allow seamless experience (Eeles, 2001). Table 4.4 shows an overview of the requirements later followed by description.

Table 4.4
Non-Functional Requirements - Usability

ID	Requirement Title	Priority
NFR-01	Intuitive and Consistent User Interface	1
NFR-02	Support for Data Democratization	2
NFR-03	Desktop Oriented User Interface Scope	2

NFR-01: Intuitive and Consistent User Interface
The system shall provide a clean, consistent, and intuitive user interface that enables users to efficiently understand and operate the platform. The user interface shall present all information necessary for the correct configuration of mining tasks in a clear and accessible manner.

NFR-02: Support for Data Democratization
The system shall adhere to the principles of data democratization by shielding users from unnecessary technical complexity and advanced configuration details. Interaction with the platform shall be designed in a way that abstracts low-level technical aspects of data mining procedures.

NFR-03: Desktop Oriented User Interface Scope
The user interface of the proposed platform is primarily designed for desktop usage. Given the analytical nature of the system and the volume and complexity of information presented to the user, full optimization for mobile devices is not considered a primary requirement.

Reliability

In accordance with the FURPS quality model, the reliability category addresses the system's ability to operate correctly and consistently over time. This includes aspects such as system availability, resilience to unexpected failures, recovery mechanisms, and the accuracy and correctness of the results produced by the application (Eeles, 2001). Table 4.5 shows an overview of the requirements later followed by description.

Table 4.5
Non-Functional Requirements - Reliability

ID	Requirement Title	Priority
NFR-04	Parallel Processing of Analytical Tasks	1
NFR-05	System Availability and Uptime	2

NFR-04: Parallel Processing of Analytical Tasks
The system shall support parallel execution of analytical tasks in order to prevent performance degradation under concurrent workload. Given that data mining tasks may be computationally intensive and long-running, the platform shall be capable of processing multiple task executions simultaneously while continuing to handle other user interactions, such as data preprocessing operations and standard API requests. This request also comes from the factor stated in the CleverMiner library documentation, that the mining tasks migh be CPU heavy (Little Big Company & Máša, 2026).

NFR-05: System Availability and Uptime
The system shall ensure a high level of availability when deployed in a cloud environment. System uptime shall be supported by the reliability guarantees provided by the selected cloud infrastructure and hosting services.

Performance

In the context of the FURPS quality model, the performance category focuses on the system's ability to efficiently process workloads and handle multiple concurrent requests. This includes aspects such as throughput, resource utilization, and the system's responsiveness under load, particularly during the execution of computationally intensive operations (Eeles, 2001). Table 4.6 shows an overview of the requirements later followed by description.

Table 4.6
Non-Functional Requirements - Performance

ID	Requirement Title	Priority
NFR-06	Efficient Handling of CPU-Bound Workloads	1

NFR-06: Efficient Handling of CPU-Bound Workloads
As an analytical platform, the system shall efficiently handle computationally intensive, CPU-bound operations, including dataset preprocessing and the execution of CleverMiner mining procedures.

Building upon the reliability requirement for parallel task execution, the system shall apply appropriate CPU utilization strategies to ensure that multiple analytical tasks can be pro-

cessed concurrently without causing significant degradation in overall system performance or responsiveness.

Supportability

Within the FURPS quality model, supportability addresses the ease with which a system can be maintained, extended, deployed, and operated over time (Eeles, 2001). Table 4.7 shows an overview of the requirements later followed by description.

Table 4.7
Non-Functional Requirements - Supportability

ID	Requirement Title	Priority
NFR-07	Maintainability and Extensibility	3
NFR-08	Deployment Flexibility	1

NFR-07: Maintainability and Extensibility
The system shall be designed to support long-term maintainability and extensibility. This includes providing clear user-facing documentation that explains how to operate the platform and perform common analytical tasks, as well as mechanisms for collecting user feedback and reporting issues in order to support continuous improvement.

From an implementation perspective, the system shall be structured in a modular and well-documented manner, enabling individual components to be maintained, extended, or replaced without affecting the overall system logic. This includes the use of clear internal documentation and explanatory comments in key parts of the source code.

NFR-08: Deployment Flexibility
The system shall support deployment both on local machines and in cloud-based environments. This requirement reflects one of the key value propositions of the platform, namely its ability to operate in different execution contexts depending on user needs and available infrastructure.

4.3.3 Other Requirements

This sub-chapter concludes Section 4.3 with requirements, that did not directly fit into the previous two categories and are not directly connected to the developed application, but rather play a support role to the application itself.

Documenation

The authors of the LISp-Miner application provide users with a wiki page[1] presenting how the application can be used through a case study build around a specific dataset and data mining analysis.

The proposed analytical platform will fundamentally share the same processes when configuring and creating analytical tasks. Moreover, since GUHA procedures require users to have at least a basic understanding of association rule mining, an updated documentation resource is necessary to guide non-technical users on how to operate the platform.

4.4 User Stories

User stories are an important input into architectural decisions, creation of user interface and design and mainly help with the core implementation of essential features as described by the MMSP-AV methodology (Kemr, 2016a). Therefore this chapter is fundamental for chapters Chapter 5 and Chapter 6.

User stories should follow a defined format (Cohn, n.d.):

As a ***<who>****, I want* ***<what>****, so that* ***<why>****.*

This template description is complementary to the required MMSP-AV methodology structure of user stories and helps developers to understand what user will do with the application and therefore justify the need for the defined features in Section 4.3. User stories defined in this chapter follow the MMSP-AV output file for user stories, which describes the stories from multiple points of view.

The user stories are described by these attributes:

- Why should the functionality be implemented (what is the added value).
- Own description of the story.
- Prerequisities for the user story.
- Basic Information (actor, actor restriction, actor action, action result and testing)

It is also important to connect the user stories to existing requirements defined in Subsection 4.3.1 therefore author also provides a reference to the specific functional requirements. Five main user stories were selected by the author for detailed description as describing all the user stories for the proposed platform would be beyond the scope of this thesis.

[1] https://lispminer.github.io/wiki/start.html

Table 4.8
Overview of User Stories

ID	User Story Title	Implements Requirements
US-01	Configure Analysis Task	FR-05, FR-06, FR-07
US-02	Visualize Analysis Results	FR-08, FR-09
US-03	Dataset Preparation and Analytical Guidance	FR-02, FR-03, FR-04,
US-04	Collaborative Analysis within Projects	FR-11, FR-12, FR-13,
US-05	Execution Flexibility and Repeated Analysis	FR-07, FR-10

US-01: Configure Analysis Task
As a platform user, I want to configure an analysis task by selecting a suitable GUHA based procedure and define its parameters, so that I can perform data mining analysis on a selected dataset.

This user story primarily supports functional requirements FR-05 (Mining Task Creation and Procedure Configuration), FR-06 (Quantifier and Logical Structure Configuration), and FR-07 (Task Persistence and Execution Control). Together, these requirements represent the core analytical functionality and the main value proposition of the platform. Basic information about the story are presented in the Table 4.9.

Table 4.9
US-01 Configure Analysis Task - Basic Information

ID: US-01	**Configure Analysis Task**
Refers to	FR-05, FR-06, FR-07
Actor	Student, Researcher
Preconditions	The user is logged in and has access to at least one uploaded dataset.
Actor Action	User configures a task via a form (procedure, dataset, parameters).
Action Result	A mining task is created, then either executed or saved as a draft.
Testing	Validation via task creation for all GUHA procedures and verification of parameter integrity.

US-02: Visualize Analysis Results
As a platform user, I want to visualize the results of a successfully completed analysis run, so that I can interpret the discovered association rules and assess their relevance.

This user story primarily supports functional requirements FR-08 (Visualization of Mining Results) and FR-09 (Export of Mining Results), which together address the interpretability and practical usability of the analytical outcomes. Basic information about the story are

presented in the Table 4.10.

Table 4.10
US-02 Visualize Analysis Results - Basic Information

ID: US-02	**Visualize Analysis Results Task**
Refers to	FR-08, FR-09,
Actor	Student, Researcher
Preconditions	At least one analysis run has been successfully completed.
Actor Action	The user selects a specific run from the list of executed runs and views its results.
Action Result	The system presents a comprehensive visualization of the mining results, including discovered association rules, their validity measures (e.g., confidence), four-fold contingency tables, and graphical summaries such as histograms.
Testing	The user story is validated by selecting completed runs and verifying that all associated results and visual representations are correctly displayed and correspond to the executed analysis.

US-03: Dataset Preparation and Analytical Guidance
As a platform user, I want to upload a dataset and obtain structured information and guidance about it, so that I can appropriately prepare the data for subsequent analysis.

This user story primarily supports functional requirements F002 (Dataset Upload and Dataset Overview), F003 (Dataset Analysis, Guidance, and Preview), and F004 (Dataset Preprocessing). These requirements collectively address the data understanding and data preparation phases of the CRISP-DM methodology. Basic information about the story are presented in the Table 4.11.

Table 4.11
US-03 Dataset Preparation and Analytical Guidance - Basic Information

ID: US-03	**Dataset Preparation and Analytical Guidance Task**
Refers to	FR-02, FR-03, FR-04
Actor	Student, Researcher
Preconditions	The user is logged in and has access to at least one uploaded dataset.
Actor Action	The user uploads a dataset or accesses the details of an existing dataset.
Action Result	System presents four different tabs providing dataset analysis, dataset preview, CleverMiner guidance and an analysis of the provided attributes.
Testing	The user story is validated by uploading a new dataset and verifying that the dataset analysis, guidance, preview, and preprocessing features are correctly accessible and functional.

US-04: Collaborative Analysis within Projects
As a platform user, I want to create or join analytical projects, so that I can collaborate with other users on data mining tasks and share analysis results.

This user story primarily supports functional requirements FR-11 (Project-Based Collaboration), FR-12 (Project Membership Management), and FR-13 (Access and Participation of Project Members), which together define the collaborative capabilities of the platform. Basic information about the story are presented in the Table 4.12.

Table 4.12
US-04 Collaborative Analysis within Projects - Basic Information

ID: US-04	**Collaborative Analysis within Projects**
Refers to	FR-11, FR-12, FR-13
Actor	Student, Researcher
Preconditions	The user is logged in.
Actor Action	The user either creates a new project or receives an invitation to join an existing project.
Action Result	Upon accepting an invitation, the user becomes a project member and gains access to the project's shared resources.
Testing	The user story is validated by testing two scenarios: creation of a new project by a user, and acceptance of a project invitation by an invited user.

US-05: Execution Flexibility and Repeated Analysis
As a platform user, I want to create analysis tasks and decide when to execute them, so that I can flexibly experiment with different configurations and perform repeated analyses.

This user story primarily supports functional requirements FR-07 (Task Persistence and Execution Control) and FR-10 (Task Execution and Run Management), which together define the execution lifecycle and experimental flexibility of the platform. Basic information about the story are presented in the Table 4.13.

Table 4.13
US-05 Execution Flexibility and Repeated Analysis - Basic Information

ID: US-05	**Execution Flexibility and Repeated Analysis**
Refers to	FR-07, FR-10,
Actor	Student, Researcher
Preconditions	The user is logged in and at least one analysis task has been defined.
Actor Action	The user executes an existing task one or more times, optionally modifying task parameters between executions.
Action Result	Each execution produces a distinct run containing the results of the analysis, allowing the user to compare outcomes across multiple runs of the same task.
Testing	The user story is validated by executing a single task multiple times with different parameter configurations and verifying that separate runs are created and that their corresponding results are correctly stored and accessible.

4.5 User Interface and User Experience Design

User interface is the link between the created website and its intended user (Plego, 2018), it is therefore important to prepare user interface that allows engaging user interactions with the application, so that user intuitively knows how to work with the page (Stone et al., 2005). This chapter discusses and presents design templates for the proposed platform, as it is important to avoid user dissatisfaction or frustration with the platform, that would discourage users from using it (Stone et al., 2005). Mid-fidelity wireframes are presented in this chapter, supported by colour palette for the platform to prepare design for the implementation chapter Section 6.2.

Wireframes are low-fidelity design sketches that allow to focus on basic functionality that aligns with the requirements presented in the Subsection 4.3.1. In wireframing, it is impor-

tant to keep the wireframes simple and maintain design consistency (Figma, n.d.). Further presented wireframes contain mid-final text, layout and work with white space to provide understanding to the connection of design and requirements. Requirements fulfilled by following wireframes are FR-03, FR-05, FR-08, FR-09.

Author selected Shadcn (shadcn.io, n.d.) as the main component library for the proposed platform based on its great usability and configurability with Tailwind CSS (Labs, n.d.). The selection of component library is further discussed in the Subsection 5.2.2 as it is not the main goal of this chapter. Wireframes are created with an open-source Figma Shadcn library (Studio, n.d.) that provides most of the components that Shadcn provides for web development. Wireframes are created in Figma desktop application.

4.5.1 Application Wireframes

The wireframes were created for the following key screens of the platform: Dataset Analysis, Runs overview, Analysis task creation and configuration, and Visualization of running or completed results. These screens represent the core user interactions identified in the functional requirements and user stories.

All screens share a consistent layout dominated by a persistent sidebar, which provides the primary navigation mechanism of the platform. The sidebar is structured using collapsible navigation groups that organize pages according to their functional domains, such as Tasks, Runs, and Datasets. This approach supports clarity and reduces visual clutter by exposing only the relevant navigation options at a given time.

At the bottom of the sidebar, additional navigation items are provided for user management, feedback submission, and support-related actions. The sidebar itself can be collapsed, allowing the main content area to expand and provide additional space for analysis configuration and result visualization.

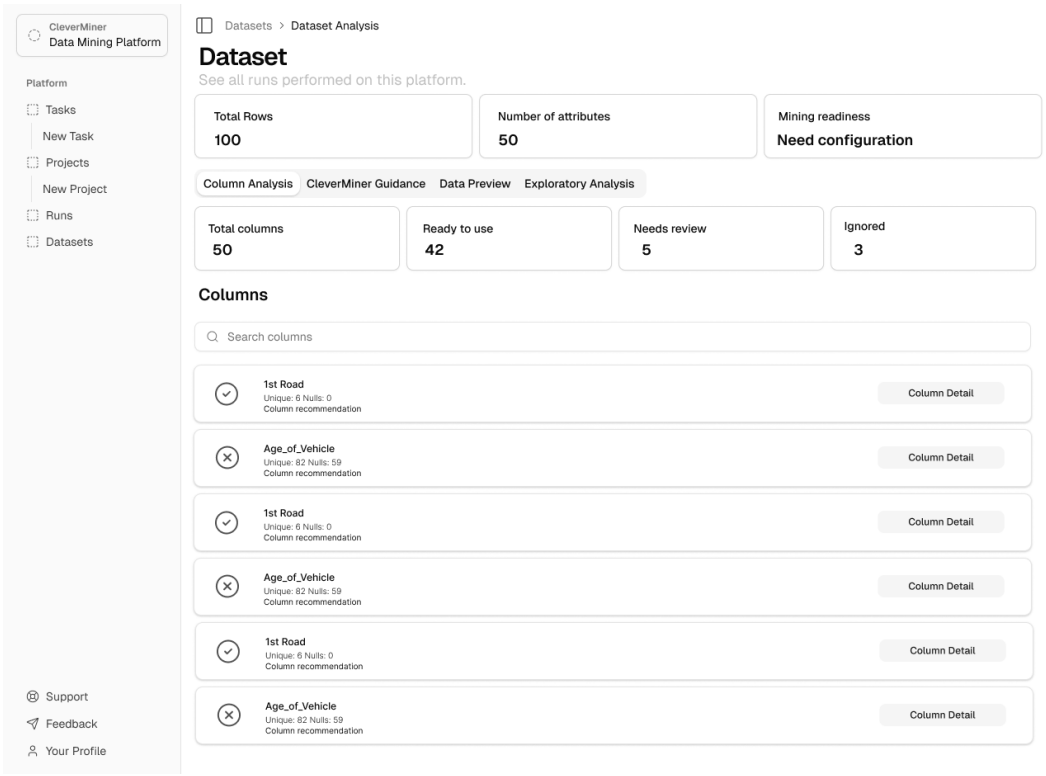
For navigation within deeper levels of the platform hierarchy, a breadcrumb component is displayed at the top of the main content area. This component enables users to maintain awareness of their current location within the platform and supports efficient navigation across different levels of the application.

Dataset Page

The wireframed dataset page provides basic overview of dataset statistics and then allows user to pick from four tabs that are available: Column Analysis, CleverMiner Guidance, Dataset Preview and Exploratory Analysis. The modelled tab in Figure 4.1 presents Column Analysis. Column Analysis provides a list of all the attributes present in the dataset and suggests whether the attribute should be pre-processed or can be used as is. By clicking on the item in the columns list or using the Column Detail action a drawer from the right

side should be invoked to provide column transformations and further details of the selected attribute.

Figure 4.1
Wireframe - Dataset Page

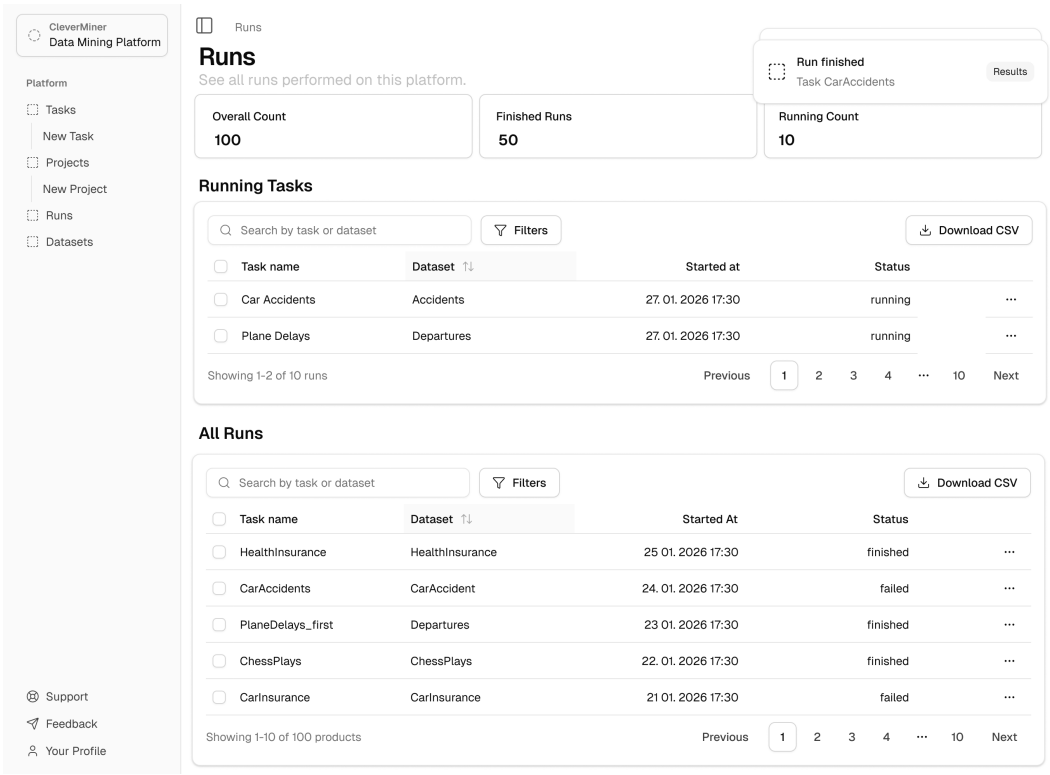


Note: Created by author using Figma.

Runs Page

To provide consistency across the pages, the runs page Figure 4.2 starts with an overview of all runs, providing basic information about the number of runs, finished runs and runs currently being executed. Below this overview user can find two tables. One provides information about the currently running runs and the other shows all the finished runs. Both tables provide filtration, search bar and capability of exporting the runs to csv file.

Figure 4.2
Wireframe - Runs Page

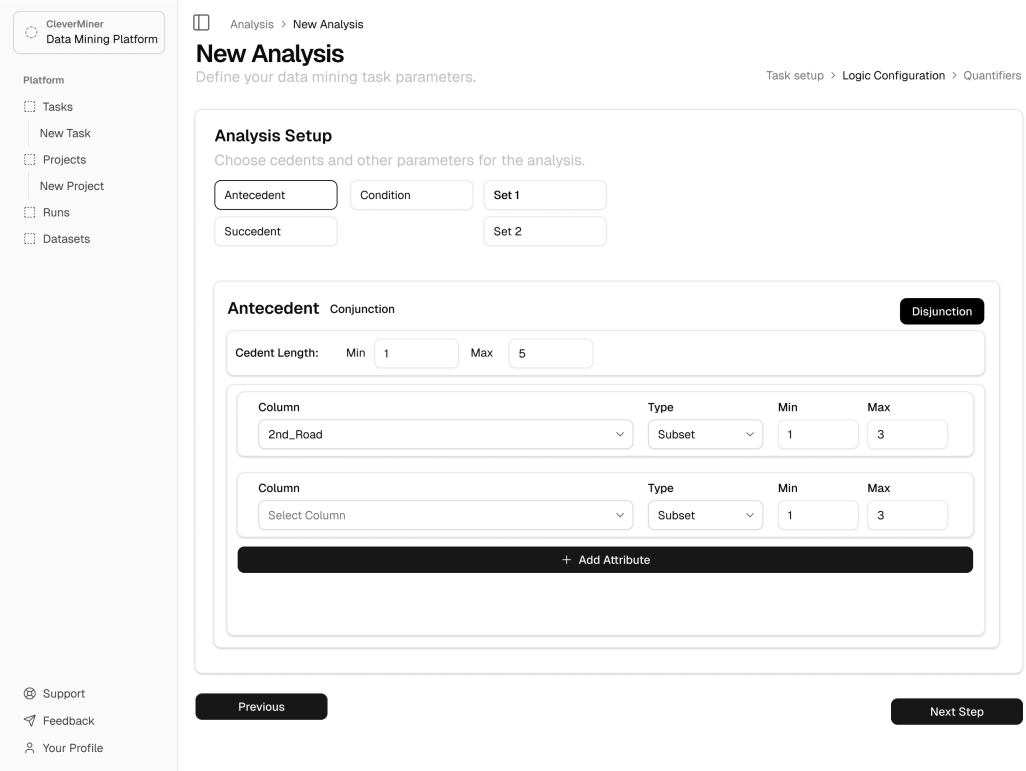


Note: Created by author using Figma.

New Analysis Page

New analysis page presented in Figure 4.3, shows the creation of new analysis. The selected tab displays Logic Configuration where user selects cedents that will be used in the analysis. Simple form implements all the necessary features, that selected GUHA procedure should contain. User has the ability to select sizes of each cedents as well as the whole antecedent or succedent. Added attributes then allow for selection of the type and length of the selected values for the analysis. Button to switch from conjunction or disjunction of the attributes is provided to finalize all the options offered by the CleverMiner library.

Figure 4.3
Wireframe - New Analysis Page

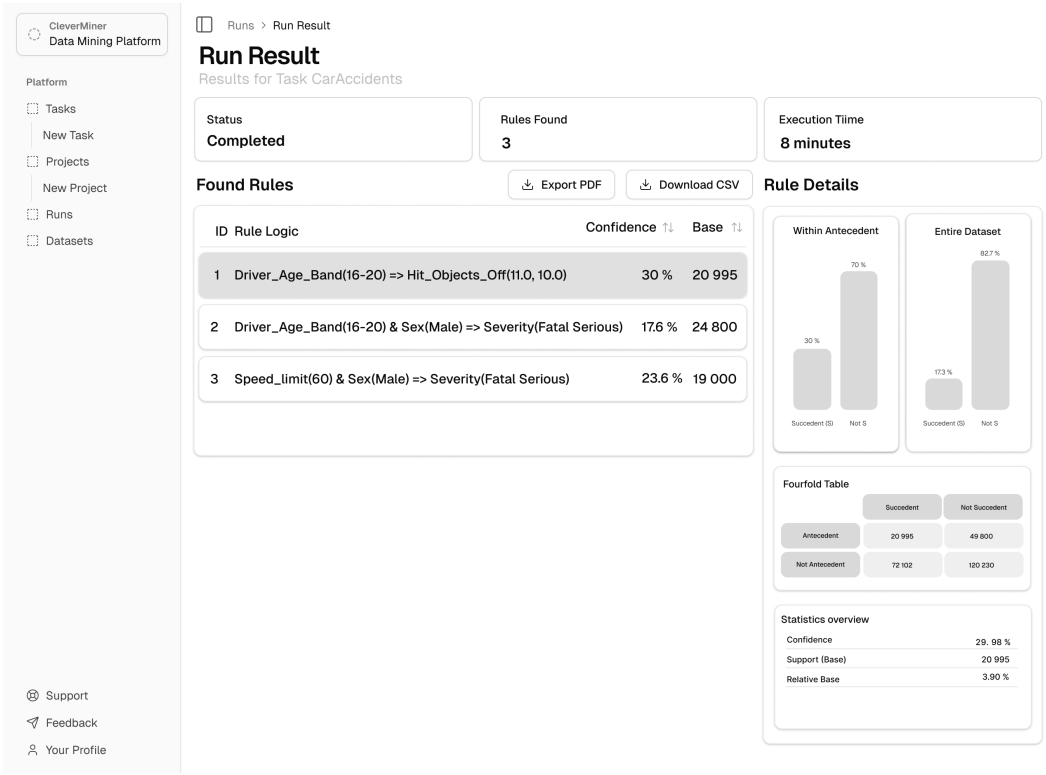


Note: Created by author using Figma.

Run Results Page

Run result page Figure 4.4 provides a clear way of interacting with the found rules. To complete the consistency, the page starts with Cards component overview. Below the overview follows rules table which contains all the found rules. Right side of the page shows data for the currently selected rule (showed by the highlighted rule). Base histograms are provided as well as four-fold table.

Figure 4.4
Wireframe - Run Results Page



Note: Created by author using Figma.

4.5.2 Colour Palette

The colour palette is specified in this chapter to finish the shape of the final UX. Colours enhance the aesthetics of the web page, help to stir the emotions of the users and create a certain relationship between the user and the web page so that the intended message reaches the end-users when using the application (Soegaard, 2026).

Colours were created by working with the colour scheme of the main CleverMiner page. However, to rather fit the platform, the colours were later updated, and final colour palette is introduced in the picture Figure 4.5.

Figure 4.5
Colour Palette

Note: Created by author using Coolors application (Coolors, n.d.).

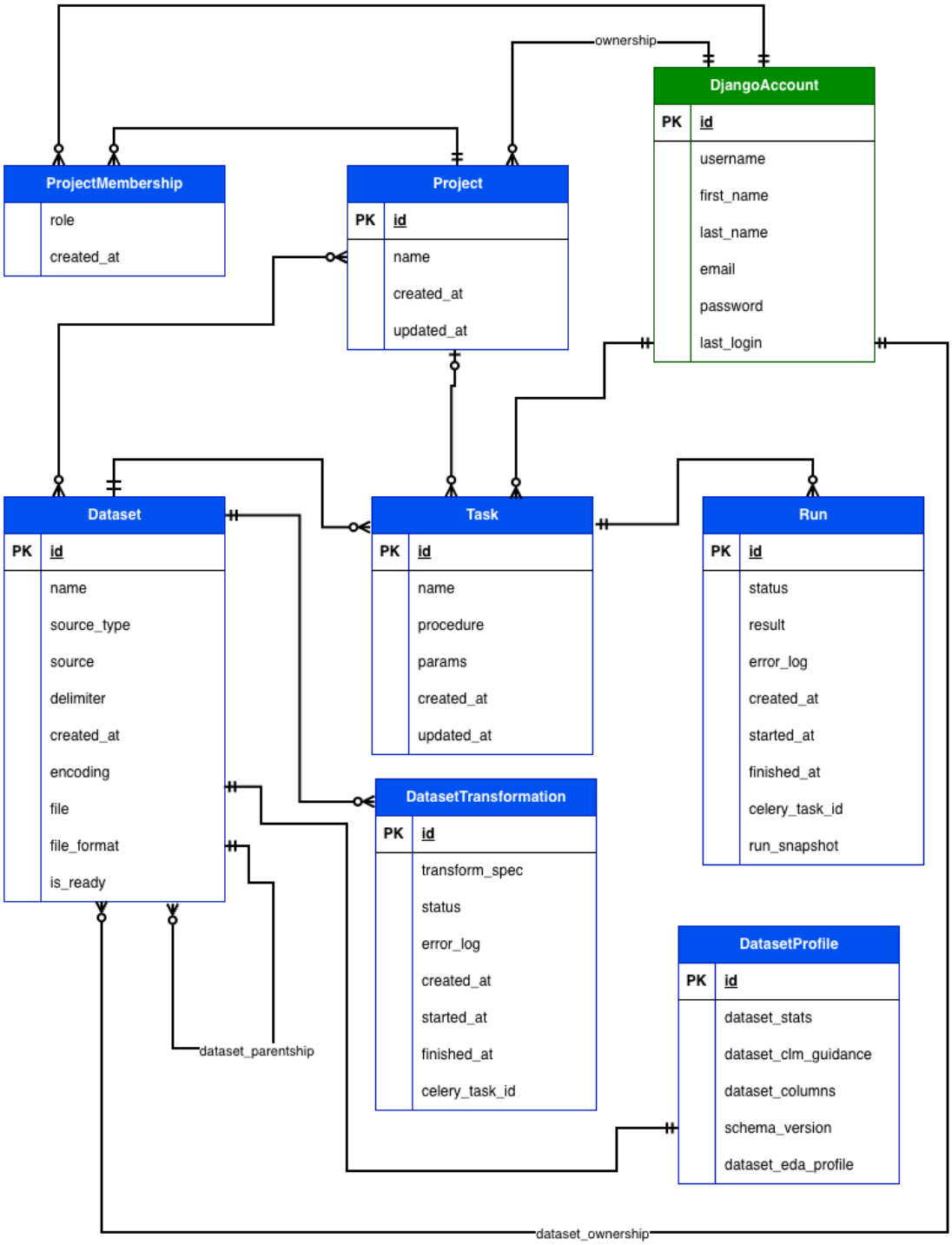
4.6 Database Conceptual Schema

Database conceptual schema is essential when researching the field domain and getting to know all the neccessary factors, attributes and relationships that create the examined world (Chlapek et al., 2019). This chapter provides an essential view into the modeled world of association rules and describes in high level, how the database will be implemented. The schema does not contain native Django tables, since that is not relevant to the inspected world.

The Figure 4.6 provides a clear view into the modeled domain and shows the main identified entities that are relevant to the proposed platform. The schema does not contain any foreign key restrictions neither does it contain data types, as this is not part of the conceptual schema (Chlapek et al., 2019). Relationships are shown using cardinality and parciality and are later in detail described. The relationships are presented through Crow's foot[2] notation.

[2]https://www.drawio.com/blog/crows-foot-notation

Figure 4.6
Database Conceptual Schema



Note: Created by author using draw.io website.

The DjangoAccount entity is not further discussed by the author, since it provides base integration of users into the platform and is not explicitly created by the author, but rather used from the Django implementation. DjangoAccount entity has connections to Dataset, as this describes dataset ownership. Entity contains relationship to the ProjectMembership to allow for project members and also connects to Project to satisfy project ownership constraint. Furthermore, account is connected to task, as to provide the essential functioanlity to the platform.

Table 4.14
Database Conceptual Schema - Entities

Entity	Entity Description
Task	Task definition including name and mining parameters.
Run	Represents a single execution of a specific Task.
Dataset	Metadata regarding the uploaded CSV files.
DatasetTransformation	Records of preprocessing steps applied to a dataset.
Project	A collaborative container for sharing tasks and datasets.
ProjectMembership	Mapping table for users invited to specific projects.
DjangoAccount	Table showing the base user account, that is created by Django.
DatasetProfile	Table showing the current overview and statistical data about dataset.

Task Entity

Task is the main entity in the whole system. When user is creating new analysis as described in the Configure Analysis Task user story, see Table 4.4, the task instance is created. As this entity is essential for passing on the parameters of the configured analysis, the solution on how to map the complex reality of the association rules mining with cedents, literals and their quantifiers went through a number of iterations to come up with the presented design. As this platform provides all the available miners from CleverMiner libarary instead of just one as with the theses by Vala (Vala, 2025) and Zedníčková (Zedníčková, 2025) a new approach to task management and saving of the parameters was devised and is described in next paragraphs.

As discussed by Vala in Chapter 5.2 of his thesis (Vala, 2025), implementing multiple GUHA mining procedures within a single platform may lead to a rapid increase in the number of domain entities and overall system complexity. Such an approach, while structurally explicit, can negatively affect maintainability, extensibility, and long-term evolution of the system. In particular, the propagation of procedure-specific entities introduces redundancy and increases the effort required to support additional mining methods or modify existing ones.

The solution proposed by Vala (Vala, 2025), which relies on the use of abstract classes at the

database level to unify multiple mining procedures, was carefully considered during the design of the proposed platform. However, this approach was ultimately not adopted. Instead, an alternative design was selected in order to reduce structural complexity while preserving flexibility and correctness.

The params attribute in Task entity is implemented as a structured JSON field that contains the complete specification of antecedents, succedents, quantifiers, and other procedure dependent settings required for task execution. Validation of this configuration is performed outside the persistence layer using an internal backend schema, implemented with the Pydantic validation framework, later explained in Subsection 6.1.4. This validation ensures that only well-formed and semantically correct task definitions are stored and passed to the CleverMiner library.

This design explicitly separates persistence concerns from procedural logic. The database layer is responsible solely for storing task configurations, without encoding knowledge of GUHA-specific structures such as cedent composition. Procedural correctness is enforced at the application logic level, which results in reduced coupling between system layers and improves maintainability.

An additional advantage of this approach lies in its natural compatibility with the CleverMiner library, which expects procedure configurations to be provided as flat dictionary structures. By storing task parameters in a JSON-based format, the platform avoids unnecessary data transformations and preserves a direct mapping between stored configurations and library inputs.

By prioritizing a flexible schema over rigid inheritance hierarchies at the database level, the proposed architecture remains robust, maintainable, and easily adaptable to future extensions of the CleverMiner library or the introduction of additional GUHA mining procedures.

The Task entity therefore has six main attributes that are described in the Table 4.15.

Table 4.15
Database Conceptual Schema - Task Entity

Attribute	Attribute description
`id`	Primary key.
`name`	Name of the created task.
`procedure`	Describing the GUHA procedure that the task is created for.
`params`	Configuration of the analysis.
`created_at`	Timestamp of initial task creation.
`updated_at`	Timestamp of the last modification.

Run

Run provides results of the mining. Every time an analysis is runned on a task a new run is created in the database. It has 9 attributes that represents it. The result attribute provides a JSON field of found rules and their subsequent parameters. Another important attribute is celery_task_id which connects the run to a certain task runned on the Celery worker, this solution is connected to non-functional requirements defined in Subsection 4.3.2 and will be later thoroughly described in the Section 6.1. All the attributes are described in the Table 4.16.

Table 4.16
Database Conceptual Schema - Run Entity

Attribute	Attribute description
`id`	Primary key.
`status`	Current state of the execution (e.g., Pending, Finished, Failed).
`result`	Output of the mining procedure.
`error_log`	Detailed logs in case of execution failure.
`created_at`	Timestamp when the run was requested.
`started_at`	Timestamp when processing actually began.
`finished_at`	Timestamp when processing completed.
`celery_task_id`	Reference to the background worker task ID.
`run_snapshot`	Provides the analysis configuration for the specific run.

Dataset

The Dataset entity provides uploading capabilities and also provides source for DataTranformation entity. This entity provides both ways of storing the datasets, and that is using the file upload or uploading a web url for the dataset. All the attributes are described in the Table 4.17.

Table 4.17

Database Conceptual Schema - Dataset Entity

Attribute	Attribute description
`id`	Primary key.
`name`	Filename or user-defined name for the dataset.
`source_type`	Indicator of the data origin.
`source`	Path or URL to the raw data.
`delimiter`	Character used to separate values (e.g., comma, semicolon).
`created_at`	Timestamp of upload.
`encoding`	File encoding format (e.g., UTF-8).
`file`	Pointer to the physical file storage.
`file_format`	The extension or type of the file (e.g., .csv).
`is_ready`	Boolean flag indicating if the data is processed for analysis.

Dataset allows for creating new datasets by the mentioned DatasetTransformation entity. As the conceptual model Figure 4.6 shows, there is hierarchival relationship between the datasets. By creating transformation, new dataset is created in order to save the initial one. The child dataset has a relation to the parent dataset, so that the path from base to transformed can be later reconstructed.

4.6.1 Dataset Profile

The Dataset Profile entity introduces performance wise optimalization for the platform, when descriptive data about the uploaded datasets are requested. This entity is mainly employed on the dataset overview page, where users can learn about dataset columns and guidance from internal CleverMiner logic. This approach decreases the requirements on server, as it is not necessary to recount statistical data every time the dataset is requested. The fields are of JSON type, which provides clear connection to the API service and the request type that frontend expects. All the attributes from this entity are described in following Table 4.18.

Table 4.18

Database Conceptual Schema - Dataset Profile Entity

Attribute	Attribute description
`id`	Primary key.
`dataset_stats`	Contains data about columns, missing values, uniqueness.
`dataset_clm_guidance`	Guidance on what could be used in CleverMiner, what could become target attribute and what needs preprocessing.
`dataset_columns`	Attribute containing all the columns in the dataset.
`dataset_eda_profile`	Stores the EDA results.
`schema_version`	Stores the version of current profile schema.

Dataset Transformation

This entity provides a way to transform datasets based on the requirements stated in FR-04 from Subsection 4.3.1. DatasetTransformation entity contains a way of mapping the transformations to Celery workers. Attributes are described in the Table 4.19. Attribute contains specification of how to preprocess selected dataset. The specification is stored in form of JSON dictionary.

Table 4.19

Database Conceptual Schema - DatasetTransformation Entity

Attribute	Attribute description
`transform_spec`	Technical specification of the applied transformation.
`status`	Completion status of the transformation task.
`error_log`	Logs documenting issues during the transformation process.
`created_at`	Timestamp when the transformation was initiated.
`started_at`	Timestamp when the worker started the transformation.
`finished_at`	Timestamp of transformation completion.
`celery_task_id`	Background task identifier for asynchronous processing.

The Source code 4.1 presents the usage transform_spec attribute with filling missing values in dataset. Format is divided into an array of steps, where each step has `op` attribute representing operation, column attribute declaring on which column should the transformation be applied and strategy, that allows for how the preprocessing should be performed.

Source code 4.1

transform_spec Attribute Example

```
1       {
2       "steps": [
3       {
```

```
4               "op": "fillna",
5               "column": "Age",
6               "strategy": "median"
7           },
8       {
9               "op": "fillna",
10              "column": "2nd_Road",
11              "strategy": "median"
12          },
13      ]
14      }
```

Project

Project entity helps to achieve the collaborative functioanlities of the platform as described in the functional requirements FR-11 from Subsection 4.3.1. Attributes are described in the Table 4.20.

Table 4.20
Database Conceptual Schema - Project Entity

Attribute	Attribute description
`id`	Primary key.
`name`	Name of the project.
`created_at`	Timestamp when the project was created.
`updated_at`	Timestamp of the last project modification.

ProjectMembership

This entity helps Project entity to implement the FR-12 from Subsection 4.3.1. Project administrator can manage users in the project. Attributes are presented in the Table 4.21.

Table 4.21
Database Conceptual Schema - ProjectMembership Entity

Attribute	Attribute description
`role`	The specific permissions or role of a user within a project.
`created_at`	Timestamp when the membership was initiated.

4.6.2 Entity Relationships

This subsection provides a description of how the entities are connected to each other and what is their parciality and cardinality.

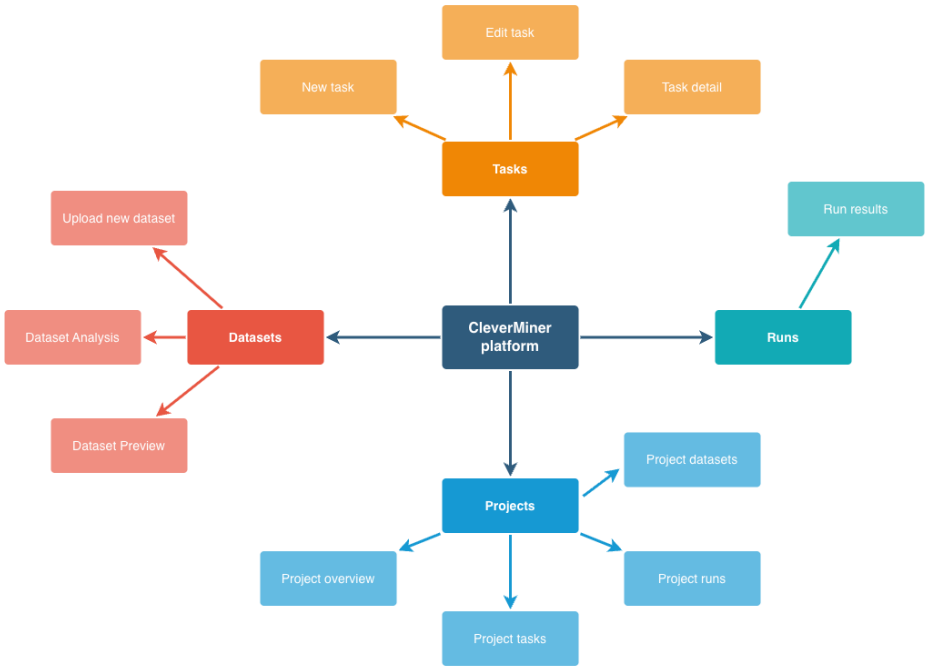
Each Dataset may be associated with zero or more Tasks, while each Task is associated with exactly one Dataset. A Task may have zero or more Runs, and each Run is associated with exactly one Task. A Project may be associated with zero or more Datasets, and a Dataset may belong to zero or more Projects. Each Dataset may have zero or more associated DatasetTransformations, whereas each DatasetTransformation is associated with exactly one Dataset. Finally, a Project may be associated with zero or more ProjectMemberships, and each ProjectMembership is associated with exactly one Project.

4.7 Website Map

Website map in this chapter clearly shows a blueprint of how the application is structured regarding the navigation on the platform. This artefact helps to visualize how the application architecture will look like and helps to spot issues early with the navigation design (VisualSiteMaps, n.d.). The website map presents a clear alignment with CRISP-DM process, providing everything from data understanding, preprocessing, mining, and evaluating.

In the provided map author presents four main domains of the website: Datasets, Tasks, Runs and Project. These four domains have their own sites that provide additional functionalities to the main domain.

Figure 4.7
Website Map



Note: Created by author using draw.io website.

5. Architectural Design

The fifth chapter presents the architectural decisions made during the design of the analytical platform, discussing the advantages and disadvantages of each decision and concluding with the final proposed architecture of the application. The chapter builds upon the previous chapter and fulfills the SO3.

5.1 Solution Design

For the proposed system, a number of architectural decisions must be made in order to satisfy the non-functional requirements defined in the previous chapters. As this thesis represents an evolutionary extension of earlier work by Vala (Vala, 2025) and Zedníčková (Zedníčková, 2025), the architectural approach is not fundamentally restructured but rather builds upon and refines the existing solution. This chapter follows the MMSP-AV methodology to clearly state and explain architectural choices for the proposed platform.

The multi-layered architecture proposed and implemented by Vala (Vala, 2025) provides a suitable foundation for this project. According to Microsoft's architectural terminology, layers represent a logical separation of responsibilities, while tiers describe their physical deployment (Microsoft, n.d.). Following this distinction, the proposed system adopts a base three-layer architecture consisting of the User Interface, Business Logic, and Data Access layers, which are primarily deployed within a single application tier.

This architectural approach promotes a clear separation of concerns, improves maintainability, and allows direct access to the backend API should the user interface evolve or be replaced in the future. As noted by Vala (Vala, 2025), such separation also supports scalability and extensibility of the system while keeping the overall structure comprehensible.

A key distinction between the proposed solution and previous implementations lies in the execution model of computational tasks. As is discussed in Section 5.5, data mining procedures and dataset preprocessing represent CPU-bound workloads, for which Python exhibits structural limitations due to the Global Interpreter Lock (GIL). To address these limitations and improve computational efficiency, the proposed architecture introduces Redis and Celery as dedicated components for asynchronous task execution.

Although the inclusion of Redis and Celery introduces additional physical processes into the system, the logical three-layer architecture remains unchanged, but is rather enhanced with features of event-driven architecture. These components do not introduce entirely new business logic. Instead, they support the existing logic layer by enabling asynchronous and parallel execution of tasks. In this model, Celery workers execute computationally intensive operations independently of the main application process, while Redis serves as a technical messaging component responsible for task coordination and result propagation.

Vala (Vala, 2025) further suggested that a microservice-based architecture could be employed to improve modularity by decomposing the application into independently deployable services. Microservices aim to divide an application domain into loosely coupled subdomains, enabling independent development and deployment (Richardson, n.d.). While such approach can be beneficial in large-scale systems, its adoption in the context of this project would introduce significant architectural and operational complexity. As discussed in the article "You Want Microservices, But Do You Really Need Them?", microservices are particularly suited to systems operating at massive scale, such as global streaming platforms, where independent domain evolution is a primary concern (Hatwalne, 2025).

In contrast, the dominant scalability challenges of the proposed platform stem from CPU-bound data mining computations performed by the CleverMiner library and dataset pre-processing, rather than from the need to manage multiple independently evolving business domains. Additionally, microservice architectures typically rely on eventual consistency and compensation mechanisms instead of ACID transactions, increasing implementation complexity and reducing determinism. For these reasons, the platform adopts an asynchronous task-based architecture using Redis as a message broker and Celery workers as an execution layer instead of a fully distributed microservice design. (Hatwalne, 2025)

This architectural decision enables a clear separation between the control responsible for API access, authentication, dataset metadata, task configuration and the execution, which performs the computationally intensive mining tasks. As a result, the system supports independent scaling of worker processes, improved responsiveness of the web application, and simpler guarantees of reliability and reproducibility compared to a multi-service distributed architecture fulfilling the requirements stated in Section 4.3.

5.2 Client-Server Separation Model

As established in Section 5.1, the proposed system is architecturally divided into a user interface and a backend responsible for business logic and data access, which is further discussed in Section 5.3. Therefore frontend serves as a presentation layer and backend exposes funcionality exlusively through defined API contract. Such design supports independent evolution of both parts of the system and aligns with the layered architecture.

The backend is implemented using the Django framework, while the frontend is developed as a single-page application using React and TypeScript with client-side routing provided by React Router. The following subsections describe the roles of the selected technologies and justify their use from an architectural perspective.

5.2.1 Backend

The backend serves as the control and orchestration layer of the platform. Its responsibilities include authentication, dataset management, configuration of data mining tasks, persistence of metadata, and coordination of asynchronous execution. The execution part of the backend which is responsible for computationally expensive tasks is described in Section 5.5.

Django

Django (Django, n.d.) is used as the primary backend framework due to its mature ecosystem and strong support for rapid development of structured web applications. Its built-in mechanisms for authentication, request handling, and database integration support the layered architecture of the platform and reduce the need for boilerplate code (W3Schools, n.d.-a). From an architectural standpoint, Django provides a stable foundation for implementing the Business Logic and Data Access layers while maintaining a clear separation of concerns.

Django REST Framework

As the backend in the proposed application serves only as an API for the frontend, an API framework called Django Rest framework (Encode, n.d.) is used to expose functionality via RESTful endpoints. DRF integrates tightly with Django's data models and view mechanisms, enabling consistent serialization of domain data and standardized HTTP-based communication (Giacomelli, n.d.).

Pydantic

Pydantic (Pydantic, n.d.) is employed as a core component for validating and structuring data mining task configurations. In contrast to traditional approaches where domain structures are fully represented in the persistence layer, the domain model of CleverMiner tasks is primarily defined in the business logic layer. Pydantic models provide runtime validation, type safety, and explicit schema definitions for task configurations received via the API (Tuychiev, n.d.). This approach ensures correctness and consistency of complex analytical configurations while keeping the relational database schema focused on metadata rather than algorithm-specific structures.

Pandas

Pandas (NumFOCUS, n.d.) is used for dataset manipulation and preprocessing within the backend, as the CleverMiner library operates directly on data represented in the form of `pandas DataFrames`. Its role in the architecture is limited to data preparation and transformation required prior to task execution like imputing missing data and discretization (W3Schools, n.d.-b). This choice ensures compatibility with the mining library while keeping preprocessing logic encapsulated within the backend's business logic layer.

5.2.2 Frontend

The frontend completes the three-layer architecture by providing the user interface through which users interact with the analytical platform. It is designed as a single-page application to ensure responsiveness and efficient interaction with the backend API.

React and React Router
React (MetaPlatforms, n.d.) is used as the primary frontend framework due to its component-based architecture and suitability for building complex interactive user interfaces. The application is structured as a single-page application, with React Router (ReactRouter, n.d.) providing client-side routing and layout composition. This approach enables clear separation between different functional areas of the platform while maintaining a consistent user experience across views.

Axios and TanStack Query
Communication between the frontend and backend API is handled using Axios (Axios, n.d.), which provides a unified abstraction for HTTP requests and response handling (Kollegger, 2018). Axios is complemented by TanStack Query (TanStack, n.d.), which manages asynchronous data fetching, caching, and mutation workflows on the client side. Together, these tools simplify state management related to server data and improve responsiveness of the user interface by reducing unnecessary network requests and enabling automatic data synchronization.

5.3 Database and Data Model Design

5.3.1 Data Storage Strategy

The proposed system builds upon the data storage approach introduced in Vala's implementation, where datasets are stored outside the relational database. In cloud deployments, datasets are persisted in external object storage, while in locally deployed environments they are stored using the backend's media storage mechanisms (Vala, 2025). This approach reflects the role of the database in the system, which is primarily responsible for managing application metadata rather than storing large analytical datasets.

Storing datasets directly in a relational database is not suitable for this platform, as the database is designed to support user management, task configuration, and result metadata. Analytical datasets, which can be large and frequently accessed in a column-oriented manner, are therefore handled separately using file-based storage.

To further improve efficiency, the platform extends the original solution by introducing Apache Parquet (Apache, n.d.) as the primary dataset storage format. While CSV represents a traditional row-based storage format, it requires reading entire rows even when only a subset of columns is needed (Bhosale, 2024). This results in increased memory us-

age, input/output overhead, and CPU consumption during dataset access and preprocessing (DeWitt & Stonebraker, 2008).

Apache Parquet provides a columnar storage layout that enables selective column access without reading irrelevant data (R. Kumar, 2024). Parquet files are also self-describing, as they store schema information and data types within file metadata, simplifying dataset interpretation and validation (R. Kumar, 2024). Additionally, Parquet offers significant storage savings compared to CSV. Empirical results indicate that datasets stored in Parquet format can require substantially less disk space than their CSV counterparts (Databricks, 2018).

This choice directly supports the platform's analytical workflows and builds upon the requirements from Section 4.3. During dataset preprocessing and transformation, only relevant columns need to be loaded into memory. Similarly, during data mining execution, users typically specify a limited set of attributes for analysis, such as antecedents and succedents. In such scenarios, the ability to selectively load columns significantly reduces processing time and resource consumption. As it is uncommon for users to operate on all dataset attributes simultaneously, the use of Apache Parquet represents a key architectural decision that improves performance and cost efficiency compared to row-based formats.

5.3.2 Relational Database and Physical Model

PostgreSQL is used as the primary relational database of the system. It was selected due to its maturity, open-source nature, strong ACID compliance, and seamless integration with Python-based backend frameworks (Group, n.d.). PostgreSQL provides a reliable foundation for storing structured application data, including user accounts, dataset metadata, task configurations, and execution results.

The physical database model is designed in accordance with the capabilities of the PostgreSQL database system. As defined by Chlapek et al. (2019), a physical data model captures table structures, primary and foreign key relationships, attribute constraints, and integrity rules specific to the selected database technology. In the proposed system, the physical model explicitly defines relationships between entities, mandatory attributes, and referential constraints, ensuring data consistency and supporting reliable application behavior.

A diagram of the physical database model is presented in this section in Figure 5.1 to illustrate the structure of persisted entities and their relationships. The model reflects the architectural decision to store only metadata and configuration data in the relational database, while large analytical datasets are managed separately through external storage mechanisms.

Figure 5.1

Database Physical Model

Note: Created by author using DataGrip diagram view.

5.4 API Interface Design Strategy

The API in the proposed system represents a stable contract between the presentation layer and the business logic and orchestration components of the platform. Its design is driven by the non-functional requirements defined in Chapter Subsection 4.3.2.

Two primary API communication styles were evaluated during the design of the platform: REST and GraphQL. Other protocols, such as SOAP, were considered but quickly discarded, as they would introduce unnecessary architectural and implementation complexity without providing clear benefits for the proposed system.

The REST architectural style is based on stateless interactions with well-defined resources that are accessed via standardized HTTP endpoints (K. Kumar et al., 2023). GraphQL, on the other hand, addresses one of REST's commonly cited limitations, over-fetching of data. GraphQL solves this by allowing clients to request only the fields they require. However, GraphQL introduces additional complexity in terms of setup, tooling, and caching strategies, and does not rely on standard HTTP semantics to the same extent as REST (Team, 2025). Given these considerations, REST was selected as the most appropriate protocol for defining a clear and stable contract between the frontend and backend components of the platform.

The API exposes core entities of the system, including datasets, dataset transformations, analytical tasks, execution runs, and user management. Each entity is represented as a distinct resource with a corresponding set of endpoints. CRUD style operations are used for metadata oriented interactions, such as user authentication, dataset registration, and configuration management.

Operations related to data mining execution follow an asynchronous orchestration pattern. Mining tasks are not executed within the request–response lifecycle. Instead, the client submits an execution request, upon which the API validates the configuration, creates an execution record, and returns an acknowledgment indicating that processing has been initiated. The execution itself is performed asynchronously by the execution layer described in Section 5.5. This approach ensures responsiveness of the user interface and enables independent scaling of computational resources.

Authentication and authorization mechanisms are integrated into the API to ensure that access to resources is restricted according to user identity and assigned roles. Finally, the API is documented using the OpenAPI standard with automatically generated documentation provided via Swagger, supporting transparency, reproducibility, and ease of integration.

5.5 Execution Model and Scalability Considerations

Association rule mining procedures provided by the CleverMiner library are computationally intensive and primarily CPU-bound (Little Big Company & Máša, 2026). This characteristic

has significant implications for the execution model of the platform. In a traditional synchronous request–response architecture, long running computations would occupy web server workers for the entire duration of task execution. Under concurrent load, such behavior could exhaust the available worker pool, leading to resource starvation and rendering the web application unresponsive even for lightweight operations, such as fetching metadata or displaying user interface elements.

This risk is particularly pronounced in Python based web applications due to the Global Interpreter Lock (GIL) present in the standard CPython implementation. The GIL ensures that only one thread executes Python bytecode at a time within a single process, effectively preventing true parallel execution of CPU-bound tasks (Ajitsaria, n.d.). While production deployments of Django typically rely on multiple worker processes (e.g., via WSGI servers), each process maintains its own GIL. Consequently, if multiple workers are occupied by data mining or preprocessing tasks, the application's ability to respond to incoming requests can be severely degraded.

To address these limitations, the proposed platform adopts an asynchronous task-based execution model that separates web request handling from computationally intensive processing. Instead of executing mining tasks within the web application processes, such tasks are delegated to a dedicated execution layer implemented using Celery. This design ensures that the backend remains responsive and capable of serving user requests independently of ongoing analytical computations.

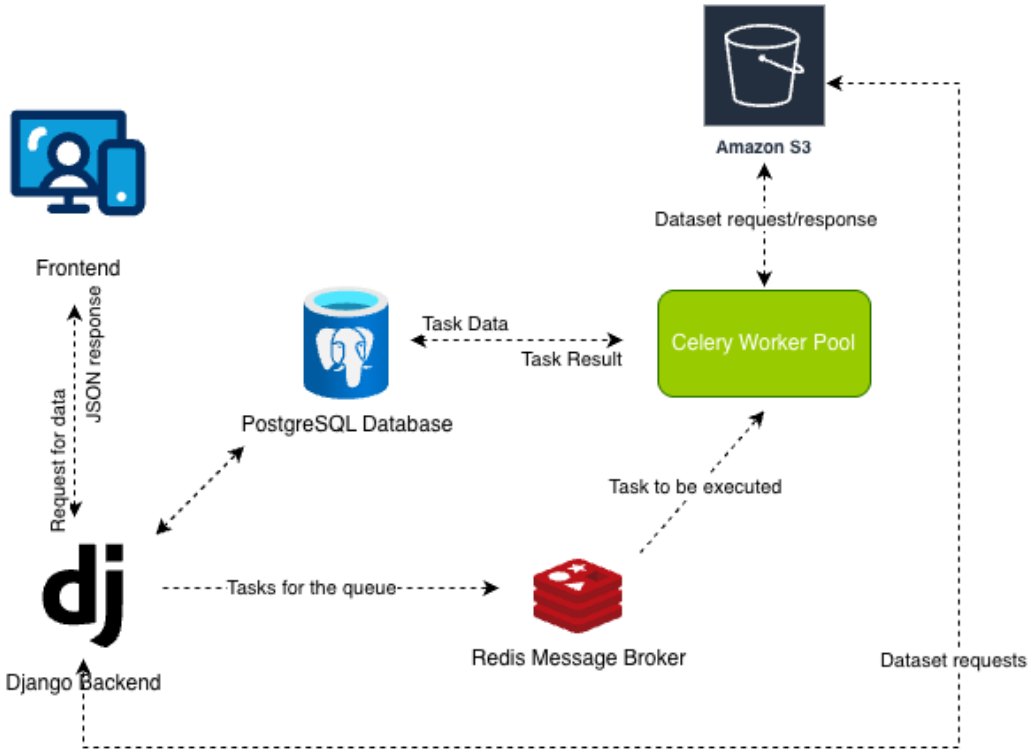
Celery executes tasks using a prefork concurrency model, in which multiple independent Python processes are spawned. Each process maintains its own memory space and its own instance of the GIL, enabling true parallel execution across multiple CPU cores. This model allows the platform to efficiently utilize available hardware resources and to scale computational throughput by increasing the number of worker processes as needed (Solem & contributors, n.d.).

Task communication between the backend and the execution layer is mediated by a message broker. In the proposed architecture, Redis is used for this purpose. Redis provides an in-memory data store capable of managing task queues with low latency, enabling asynchronous communication and loose coupling between system components (Fernandez, 2023). The backend publishes task execution requests to the message broker, while Celery workers consume and process these tasks independently. This event-driven execution model introduces a clear contract between the control plane and the execution plane of the system.

From a scalability perspective, this architecture supports horizontal scaling of computational resources. Additional Celery workers can be deployed to increase parallel processing capacity without affecting the web application layer. At the same time, isolating long-running computations from request handling prevents blocking behavior and preserves responsiveness of the user interface. The execution model therefore addresses both performance and scalability requirements while maintaining architectural simplicity and deployability in both cloud-based and local environments.

The final architecture researched, revised and concluded in this chapter is summarized in the diagram Figure 5.2 which shows how described systems depend on each other and how they create data mining platform.

Figure 5.2
Analytical Platform Main Architecture



Note: Created by author using drawio.

5.6 Deployment Considerations

One of the key non-functional requirements of the proposed platform, defined as NFR-08, is deployment flexibility. This requirement emphasizes the ability to operate the platform in different environments, including both cloud-based deployment and local execution on a user's machine. The motivation behind this requirement is to ensure that users are not forced to rely on paid cloud infrastructure when running analytical tasks, particularly in scenarios where local computational resources are sufficient.

The platform therefore distinguishes between two primary deployment targets. The cloud deployed version is intended for general users who expect a readily available, managed service without the need for technical setup. In contrast, local deployment is aimed at technically skilled users who prefer full control over execution, data locality, and resource usage. This distinction directly influences the choice of deployment technology.

Previous implementations addressed local deployment in different ways. Vala (Vala, 2025) proposed a containerized local deployment using Docker Compose, where all platform components are executed as services and accessed through a web browser. Zedníčková (Zedníčková, 2025), on the other hand, adopted a desktop oriented approach using Electron, packaging the platform as a standalone application. While Electron simplifies initial startup for end users, it introduces additional complexity when orchestrating multiple backend services and managing background execution processes.

After careful consideration, Docker Compose was selected as the primary solution for local deployment in the proposed platform. This decision is motivated by the architectural structure of the system, which consists of multiple cooperating components, including the backend API, Celery workers, a Redis message broker, and a database. Docker Compose provides a declarative and transparent mechanism for orchestrating such multi component systems, allowing services to be started, configured, and scaled in a controlled manner (Docker, n.d.-a). In contrast, integrating these components within an Electron based desktop application would introduce significant orchestration overhead while providing limited additional benefit for the intended target group of technically proficient users (Foundatin, n.d.).

The use of Docker Compose also aligns with the cloud deployment strategy. The same containerized components can be deployed locally or in cloud environments with minimal configuration changes, ensuring architectural consistency and simplifying maintenance. This approach supports reproducibility of experiments and reduces divergence between development, local execution, and production environments.

In addition to containerized deployment, comprehensive documentation is provided with the source code repository to allow advanced users to run individual components directly without Docker if desired. This further enhances flexibility and supports experimentation and extensibility of the platform.

Consistent with earlier presented solutions, analytical datasets are stored outside the relational database, either in object storage (such as S3 compatible buckets) for cloud deployments or on the local filesystem when deployed locally.

5.7 Documentation

To fulfil the requirement stated in Subsection 4.3.3, the author has decided to use a statically built documentation generator that can be maintained through Markdown files.

A number of documentation tools are currently available (Read the Docs, n.d.), with popular options including MkDocs, Zendesk, Sphinx, and Docusaurus. The author selected Docusaurus for two primary reasons. First, as the platform is built on a React-based frontend stack, Docusaurus fits naturally into the existing JavaScript ecosystem, requiring no additional tooling or context switching. Second, Docusaurus provides a polished and profes-

sional design out of the box, making it well suited for user-facing documentation intended for non-technical audiences.

6. Application Development

The Application Development chapter presents the results of the development process carried out in this thesis and fulfills the SO4 objective. It begins with the backend implementation, gradually building the reader's understanding of the system, before moving on to the frontend implementation, illustrating how the architectural components collaborate with one another. The chapter concludes with a brief presentation of the documentation created to support the developed platform.

6.1 Backend Implementation

The Backend Implementation chapter is divided into two distinct parts. The first is the execution layer, which handles data mining administration and interfaces with the CleverMiner library for task execution. This layer also establishes the connections introduced in the proposed architecture from the previous chapter, namely the distributed computing model built on Celery and the Redis in-memory message broker. The second part serves as a communication layer between the execution layer and the frontend, implementing the main backend API that exposes resources to the frontend.

The chapter opens with a broad introduction to the overall backend architecture, outlining the base structure and directory layout.

6.1.1 Overall Backend Structure

The codebase structure can be seen in Source code 6.1. Django allows multiple applications to be defined within a single project. In this implementation, two such applications are present: accounts and `cleverminer_tasks`. The accounts[1] application handles user management and authentication, providing core functionality for logging in and out, user registration, and session management. However, since Django is used here solely as a backend framework, the application required adjustments to reflect that role. The `cleverminer_tasks` application represents the core functionalities of this platform with the execution layer and internal API.

In line with modern development standards, pip was not used for package installation. Instead, the project adopts uv[2] by Astral, a next-generation package and project manager written in Rust. As a single tool, it is capable of replacing both pip and traditional virtual environment managers. Dependencies are declared and locked via pyproject.toml and uv.lock, ensuring a reproducible and correctly configured environment.

[1] https://docs.djangoproject.com/en/6.0/topics/auth/default/
[2] https://docs.astral.sh/uv/

Source code 6.1

Backend Project Structure

```
1       |-- accounts
2       |   |-- __init__.py
3       |   |-- __pycache__
4       |   |-- admin.py
5       |   |-- api
6       |   |-- apps.py
7       |   |-- migrations
8       |   |-- models.py
9       |   |-- tests.py
10      |   |-- urls.py
11      |   `-- views.py
12      |-- cleverminer_tasks
13      |   |-- __init__.py
14      |   |-- __pycache__
15      |   |-- admin.py
16      |   |-- api
17      |   |-- apps.py
18      |   |-- execution
19      |   |-- migrations
20      |   |-- models.py
21      |   |-- registry
22      |   |-- signals.py
23      |   |-- tests.py
24      |   |-- urls.py
25      |   `-- views.py
26      |-- config
27      |   |-- __init__.py
28      |   |-- __pycache__
29      |   |-- asgi.py
30      |   |-- celery.py
31      |   |-- settings.py
32      |   |-- urls.py
33      |   `-- wsgi.py
34      |-- manage.py
35      |-- media
36      |-- pyproject.toml
37      `-- uv.lock
```

The config folder contains the base configuration for the entire project. The `asgi.py` file sets up the development server, while `celery.py` provides the base configuration for the distributed workers. Most notably, `urls.py` partially shown in Source code 6.2 acts as a URL dispatcher, mapping incoming HTTP requests to the appropriate view handlers within the application. For instance, the line 3 defines the routing to the `cleverminer_tasks` implementation, which is made available directly under the top-level domain with no URL prefix.

Source code 6.2

URL Dispatcher Configuration

```
urlpatterns = [
path("admin/", admin.site.urls),
path("", include("cleverminer_tasks.urls")),
path("api/schema/", SpectacularAPIView.as_view(), name="schema
    "),
path(
"api/docs/",
SpectacularSwaggerView.as_view(url_name="schema"),
name="swagger-ui",
),
]
```

The `cleverminer_tasks` application contains the main backend service implementation for this project. It houses two key components that will be described in detail later: the execution layer, located in the execution folder, and the internal API, referred to as service layer, located in the api folder.

Source code 6.3

cleverminer_tasks Structure

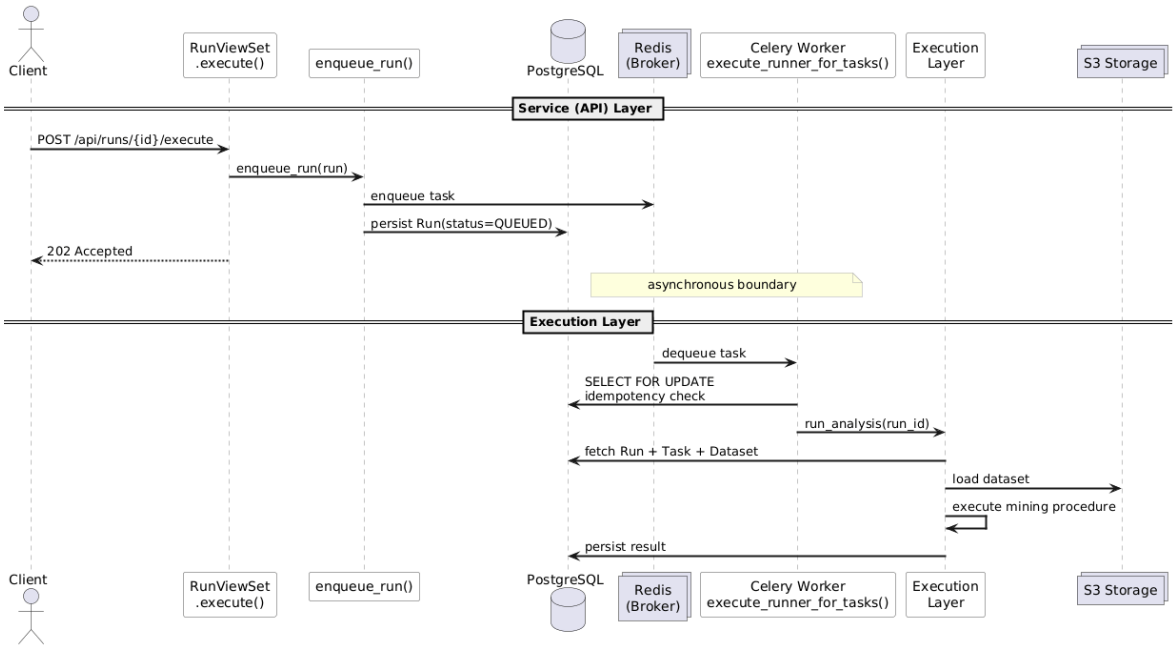
```
|-- admin.py
|-- api
|   |-- __init__.py
|   |-- __pycache__
|   |-- dataset
|   |-- execution
|   |-- procedures
|   |-- project
|   |-- runs
|   |-- tasks
|   |-- urls.py
|   `-- views.py
|-- apps.py
|-- execution
|   |-- __init__.py
|   |-- __pycache__
|   |-- datasets
|   |-- procedures
|   |-- runner.py
|   |-- shared
|   |-- tasks.py
|   `-- utils
|-- models.py
|-- registry
|   |-- __init__.py
|   |-- __pycache__
```

```
27      |   |-- analysisRegistry.py
28      |   `-- procedureConfigsRegistry.py
29      |-- signals.py
30      |-- tests.py
31      |-- urls.py
32      `-- views.py
```

A sequential diagram, presented in Figure 6.1 , gives high level overview, of how the layers communicate with each other and what each layer implements, a more detailed sequential diagrams are available in the respective sections.

Figure 6.1

Sequential Diagram - Layers Overview



Note: Created by author using Plantuml.com

To conclude this chapter, `models.py` presented in Source code 6.4 defines all the domain entities established in the system specification chapter Chapter 4. The Task and Run entities are highlighted here, as they are central to the application's functionality. These Django ORM models map directly to relational database tables and together form the task queue architecture underpinning the distributed computing model. The key field in the Task entity is `params`, a JSON field that stores the task definition. The Run entity similarly uses `run_snapshot` to capture the task definition at the moment of execution, preserving it in case the parent Task is later modified.

Source code 6.4

Task and Run ORM Models

```
1  class Task(models.Model):
```

```
2       project = models.ForeignKey(
3           "Project",
4           on_delete=models.SET_NULL,
5           null=True,
6           blank=True,
7           related_name="tasks",
8       )
9
10      name = models.CharField(max_length=200)
11
12      owner = models.ForeignKey(
13          settings.AUTH_USER_MODEL,
14          on_delete=models.SET_NULL,
15          null=True,
16          blank=True,
17          related_name="tasks",
18      )
19      dataset = models.ForeignKey(
20          Dataset, on_delete=models.CASCADE, related_name="tasks"
21      )
22
23      procedure = models.CharField(
24          max_length=32, choices=ProcedureType.choices, default=
                ProcedureType.FOUR_FT
25      )
26
27      params = models.JSONField()
28
29      created_at = models.DateTimeField(auto_now_add=True)
30      updated_at = models.DateTimeField(auto_now=True)
31
32      def __str__(self):
33          return f"{self.name} ({self.procedure})"
34
35
36  class Run(models.Model):
37      task = models.ForeignKey(Task, on_delete=models.CASCADE,
            related_name="runs")
38
39      status = models.CharField(
40          max_length=16, choices=RunStatus.choices, default=
                RunStatus.QUEUED
41      )
42
43      run_snapshot = models.JSONField(null=True, blank=True)
44      result = models.JSONField(null=True, blank=True)
45      error_log = models.TextField(null=True, blank=True)
46
47      created_at = models.DateTimeField(auto_now_add=True)
```

```
48      started_at = models.DateTimeField(null=True, blank=True)
49      finished_at = models.DateTimeField(null=True, blank=True)
50
51      celery_task_id = models.CharField(max_length=255, null=True,
            blank=True)
52
53      def __str__(self):
54          return f"Run␣#{self.id}␣of␣{self.task_id}␣({self.status})"
```

6.1.2 Execution Layer

This subchapter discusses the execution layer presented in the Source code 6.5 within the `cleverminer_tasks` application. The layer is organised into several distinct folders and is built around object-oriented principles, most notably inheritance. Since the platform integrates all four GUHA procedures available in the CleverMiner library, a clean shared foundation is established in `baseMining.py`, located in the shared folder. This abstract class exposes common methods and attributes that are then inherited by procedure-specific classes in the procedures folder.

Source code 6.5

Execution Layer Structure

```
1       |-- __init__.py
2       |-- __pycache__
3       |   |-- __init__.cpython-312.pyc
4       |   |-- __init__.cpython-314.pyc
5       |   |-- runner.cpython-312.pyc
6       |   |-- runner.cpython-314.pyc
7       |   `-- tasks.cpython-312.pyc
8       |-- datasets
9       |   |-- __pycache__
10      |   `-- service.py
11      |-- procedures
12      |   |-- __init__.py
13      |   |-- __pycache__
14      |   |-- cfMiner
15      |   |-- fourftMiner
16      |   |-- sdFourftMiner
17      |   `-- uicMiner
18      |-- runner.py
19      |-- shared
20      |   |-- __init__.py
21      |   |-- __pycache__
22      |   |-- baseConfig.py
23      |   |-- baseMining.py
24      |   `-- baseSerializer.py
25      |-- tasks.py
```

```
26      `-- utils
27      |-- __init__.py
28      |-- __pycache__
29      `-- datasetLoader.py
```

`BaseMiningService`, presented in Source code 6.6, is an abstract base class that defines a common interface and shared logic for all data mining procedures. It follows the Template Method and Strategy design patterns, ensuring that all child classes adhere to a uniform contract while maintaining a clean separation of concerns.

The class exposes several key methods, abstract methods `_mine` and `_required_attributes` delegate their implementation to child classes, as this is where procedure-specific logic diverges. The static method `add_cedent_columns` provides shared logic for extracting attribute information from a `CedentConfig` object. Finally, the `execute` method orchestrates the mining procedure by updating the run state, invoking the procedure-specific `_mine` method, and handling any exceptions, storing the result or error log accordingly before returning the completed Run instance.

Source code 6.6

BaseMiningService Abstract Class

```
1  class BaseMiningService(ABC):
2      def __init__(self, run: Run):
3          self.run_instance = run
4          self.task = run.task
5          self.dataset = self.task.dataset
6
7      @staticmethod
8      def add_cedent_columns(cedent: CedentConfig, columns: set[str
           ]):
9          if not cedent or not cedent.attributes:
10             return
11         for item in cedent.attributes:
12             columns.add(item.name)
13
14     @abstractmethod
15     def _required_attributes(self) -> list[str]:
16         raise NotImplementedError
17
18     @abstractmethod
19     def _mine(self, df: pd.DataFrame) -> Dict[str, Any]:
20         raise NotImplementedError
21
22     def execute(self) -> Run:
23         run = self.run_instance
24         run.status = RunStatus.RUNNING
25         run.started_at = timezone.now()
26         run.error_log = None
```

```
27          run.save(update_fields=["status", "started_at", "error_log
                "])
28
29          try:
30              df = load_dataset(self.dataset, columns=self.
                    _required_attributes())
31              result = self._mine(df)
32              run.result = result
33              run.status = RunStatus.DONE
34          except Exception:
35              run.status = RunStatus.FAILED
36              run.error_log = traceback.format_exc()
37
38          run.finished_at = timezone.now()
39          run.save(update_fields=["result", "status", "error_log", "
                finished_at"])
40          return run
```

6.1.3 CleverMiner Integration

4ft Mining Service

The `FourFtMiningService`[3] implementation is presented in Source code 6.7, representing a concrete implementation of `BaseMiningService` for the 4ft-Miner GUHA procedure. Upon instantiation, the class invokes the parent constructor via `super()` to resolve the context in which the new instance will operate. Since task parameters are stored in JSON format and the database does not enforce domain validity, they are deserialized and validated through the Pydantic model `FourFtConfig`, presented in Source code 6.8, which ensures the parameters are correctly structured.

Source code 6.7

FourFtMiningService Implementation

```
1  class FourFtMiningService(BaseMiningService):
2      def __init__(self, run):
3          super().__init__(run)
4          self.config = FourFtConfig(**self.task.params)
5
6      def _mine(self, df: pd.DataFrame) -> Dict[str, Any]:
7          q_dict = self.config.quantifiers.model_dump(
8              exclude_none=True, exclude={"extra_params"}
9          )
10         if self.config.quantifiers.extra_params:
11             q_dict.update(self.config.quantifiers.extra_params)
```

[3]Within the application, the 4ft-Miner procedure is referred to using a word prefix rather than the leading digit, as Python does not permit variable names to begin with a numerical character.

```
12
13          params = {
14              "df": df,
15              "proc": "4ftMiner",
16              "quantifiers": q_dict,
17              "ante": self._build_cedent(self.config.ante),
18              "succ": self._build_cedent(self.config.succ),
19          }
20
21          if self.config.cond:
22              cond_cedent = self._build_cedent(self.config.cond)
23              if cond_cedent:
24                  params["cond"] = cond_cedent
25
26          clm = self._run_miner(params)
27
28          return serialize_4ft_result(clm, self.run_instance.id)
```

The `_mine` method first prepares the quantifiers dictionary, merging any additional extra quantifiers if present. It then constructs the final params dictionary, appending the condition cedent only if one was provided in the configuration. Once assembled, the dictionary is passed to the CleverMiner library through a private `_run_miner` method inherited from the parent class. Finally, the raw result is serialized and persisted back through the execute method defined in the parent class.

Source code 6.8

FourFtConfig Pydantic Model

```
1  class FourFtConfig(BaseModel):
2      quantifiers: QuantifiersConfig
3      ante: CedentConfig
4      succ: CedentConfig
5      cond: Optional[CedentConfig] = None
```

SD4Ft Mining Service

To further fulfil the objectives of this thesis, the `SD4FtMiningService` implementation is presented in Source code 6.9. This class follows the same pattern established in Source code 6.7 for the 4ft-Miner. However, as the SD4ft-Miner procedure operates with additional attributes, the configuration is extended accordingly with set1, set2 fields supplementing the base structure. The core implementation pattern remains unchanged, ensuring code consistency and maintainability across all mining services.

Source code 6.9

Sd4FtMiningService Implementation

```python
class Sd4FtMiningService(BaseMiningService):
    def __init__(self, run):
        super().__init__(run)
        self.config = Sd4FtConfig(**self.task.params)

    def _mine(self, df: pd.DataFrame) -> Dict[str, Any]:
        q_dict = self.config.quantifiers.model_dump(
            exclude_none=True, exclude={"extra_params"}
        )
        if self.config.quantifiers.extra_params:
            q_dict.update(self.config.quantifiers.extra_params)

        params = {
            "df": df,
            "proc": "SD4ftMiner",
            "quantifiers": q_dict,
            "ante": self._build_cedent(self.config.ante),
            "succ": self._build_cedent(self.config.succ),
            "frst": self._build_cedent(self.config.set1),
            "scnd": self._build_cedent(self.config.set2),
        }

        if self.config.opts:
            params["opts"] = self.config.opts.model_dump(
                exclude_none=True)

        if self.config.cond:
            params["cond"] = self._build_cedent(self.config.cond)

        clm = self._run_miner(params)

        return serialize_sd4ft_result(clm, self.run_instance.id)
```

6.1.4 Pydantic Data Validation

This subchapter places special focus on data validation and the approach to data integrity and domain modeling introduced in previous subchapters. Each mining service defines its own configuration class, as seen in Source code 6.8, built from the most primitive domain entities, which are presented in Source code 6.10.

Source code 6.10

Base Domain Validation Models

```python
class AttributeSpec(BaseModel):
    name: str
    attr_type: AttributeType = AttributeType.SUBSET
    minlen: int = 1
```

```
5       maxlen: int = 1
6       gace: GaceType = GaceType.POSITIVE
7
8
9   class CedentConfig(BaseModel):
10      attributes: List[AttributeSpec]
11      minlen: int = 1
12      maxlen: int = 1
13      type: Literal["con", "dis"] = "con"
14
15
16  class QuantifiersConfig(BaseModel):
17      Base: Optional[int] = None
18      conf: Optional[float] = None
19      aad: Optional[float] = None
20      relbase: Optional[float] = None
21      extra_params: Dict[str, float] = Field(default_factory=dict)
22
23      class Config:
24          extra = "allow"
```

`AttributeSpec` represents a single dataset attribute, holding its name, type relative to CleverMiner settings, and the minimum and maximum length of possible values. `CedentConfig` composes one or more `AttributeSpec` instances into an antecedent or succedent for the CleverMiner call. The full list of attributes in the cedent is stored in the attributes field. Since cedents can vary in length, the corresponding `minlen` and `maxlen` fields are provided. The type field further specialises whether the cedent is a conjunction or disjunction.

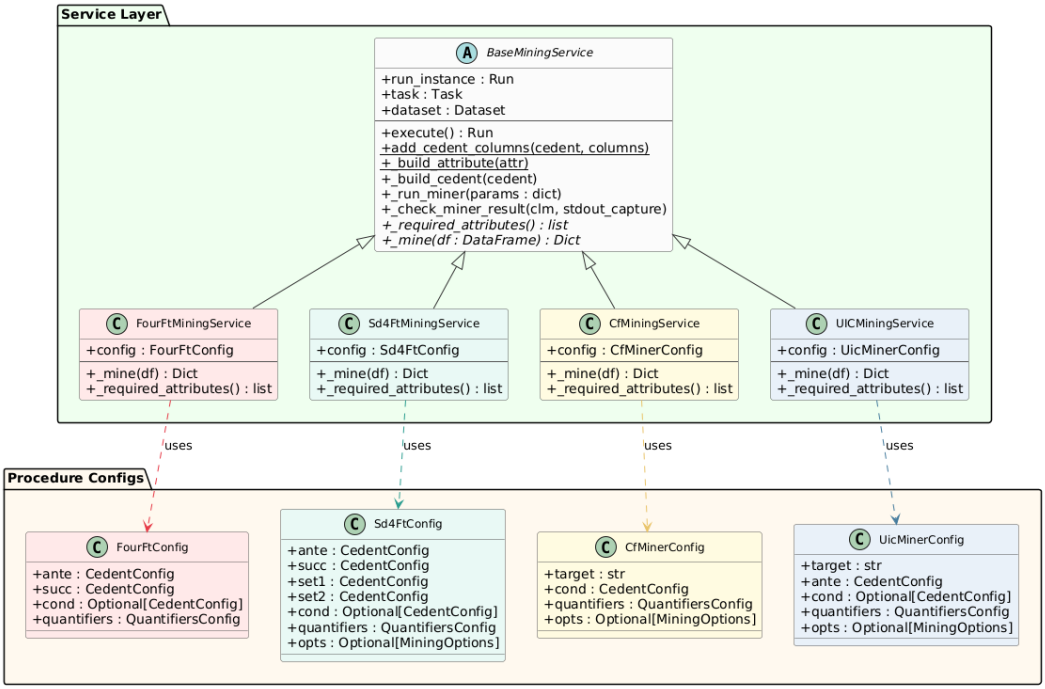
`QuantifiersConfig` manages the available quantifiers for each procedure. To accommodate procedure-specific quantifiers not covered by the base fields, additional parameters are permitted through the inner Config class.

All of these validation classes inherit from Pydantic's `BaseModel`. Upon instantiation, the class `BaseModel` automatically validates the passed data against the model's field definitions, raising a `ValidationError` before any mining logic executes if the data is malformed or missing. This lightweight validation schema avoids the overhead that would arise from modeling these entities as database tables, keeping domain validation cleanly within the execution layer.

To conclude this chapter, two class diagrams are presented in the Figure 6.2 and Figure 6.3, illustrating the full class architecture of the execution layer. The Service Layer depicts the concrete child classes inheriting from `BaseMiningService`. Their relationships with the corresponding configuration classes, which handle data validation via Pydantic, are shown in the Configuration Models section of the diagram.

Figure 6.2

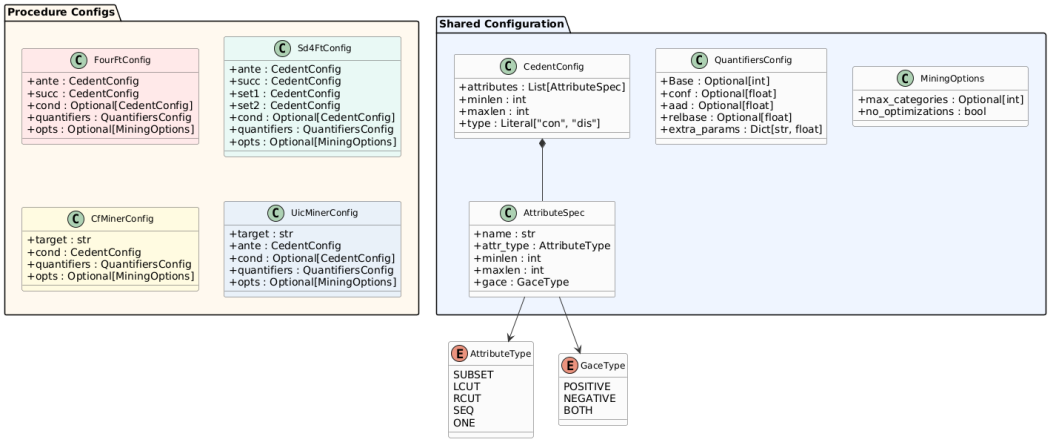
Class Diagram - Mining Classes



Note: Created by author using Plantuml.com

Figure 6.3

Class Diagram - Mining Configs



Note: Created by author using Plantuml.com

6.1.5 Distributed Architecture

This subchapter establishes a clear link between the execution layer and the service (API) communication layer by describing the flow triggered when the Service layer requests task execution.

The function `execute_runner_for_tasks`, presented in Source code 6.11, serves as the bridge between the two layers. It also acts as the Celery entry point for asynchronous task execution and concurrency control. The function is decorated with `@shared_task`, configured with automatic retry logic for `TimeoutError` and `ConnectionError` failures, with a maximum of three retries.

Inside the function, concurrency control is enforced by acquiring a database-level row lock via `select_for_update()`, ensuring no two Celery workers process the same run simultaneously. The lock is used solely to verify that the run is in a processable state. If the run has already been completed or is otherwise ineligible, the function returns early. Once this check passes, the lock is released. Notably, the run status is not updated within this function, as Celery may not have scheduled the task for immediate execution. Finally, the function delegates to `run_analysis`, passing the `run_id`.

Source code 6.11

execute_runner_for_tasks Celery Task

```
@shared_task(
    bind=True,
    autoretry_for=(
        TimeoutError,
        ConnectionError,
    ),
    retry_backoff=True,
    retry_kwargs={"max_retries": 3},
)
def execute_runner_for_tasks(self, run_id: int) -> None:
    # locking the row in db, so no two workers run the same task
        concurrently
    with transaction.atomic():
        run = Run.objects.select_for_update().get(pk=run_id)  #
            row locked

        # if the task is already finished, skip it
        if run.status not in (RunStatus.QUEUED, RunStatus.FAILED):
            return

    run_analysis(run_id=run_id)
```

The `run_analysis` function, presented in Source code 6.12, serves as an orchestration layer between the Celery task entry point and the concrete mining service implementations. The

appropriate mining service class is resolved dynamically through a registry that maps each procedure type to its corresponding mining class. This follows the Registry pattern (Fowler, n.d.), decoupling the orchestration logic from the concrete service classes and making the system straightforwardly extensible. New miners simply need to be implemented as a child class and registered in the registry, with no changes required to the orchestration logic itself.

Once the correct class is resolved, it is instantiated with the Run object and its `execute()` method is called, delegating full control to the `BaseMiningService` execution pipeline described in the previous subchapters.

Source code 6.12

run_analysis Orchestration Function

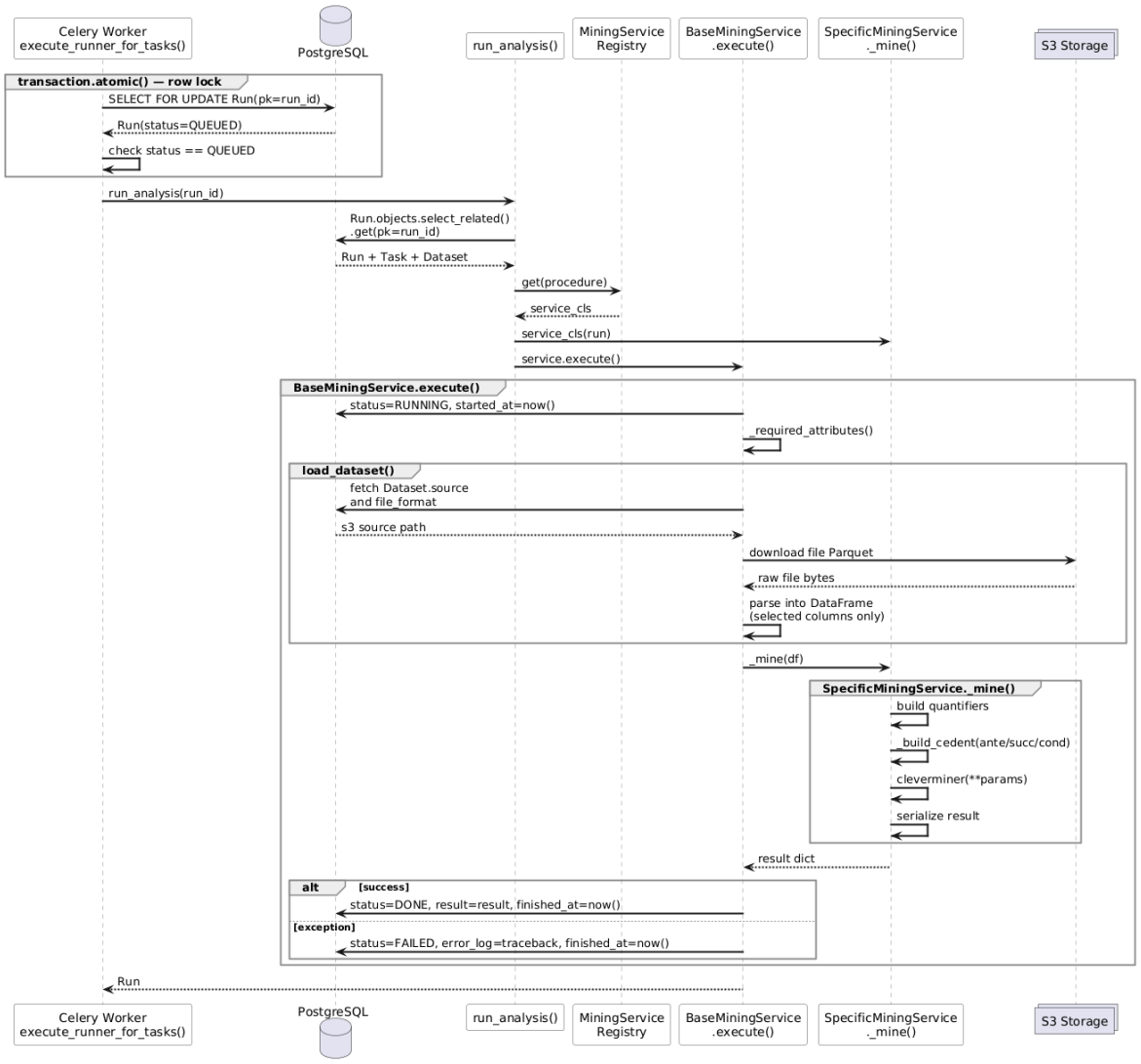
```
1  def run_analysis(*, run_id: int) -> Run:
2      run = Run.objects.select_related("task", "task__dataset").get(
           pk=run_id)
3
4      procedure = run.task.procedure
5      service_cls = MINING_SERVICE_ANALYSIS_REGISTRY.get(procedure)
6
7      if not service_cls:
8          raise ValueError(f"Unsupported␣procedure:␣{procedure}")
9
10     service = service_cls(run)
11     return service.execute()
```

As this flow may be difficult to follow in text alone, a sequence diagram is presented in the Figure 6.4 to conclude the execution layer section. The diagram intentionally omits the API layer in order to allow a deeper focus on the execution pipeline itself, though it naturally fits into the broader backend architecture that will be covered later.

Several additional details regarding data retrieval can be observed in the diagram. Dataset files are stored in S3 storage in Parquet format, meaning the mining pipeline must load only the relevant columns from the uploaded dataset before execution can begin.

Figure 6.4

Sequential Diagram - Execution Layer



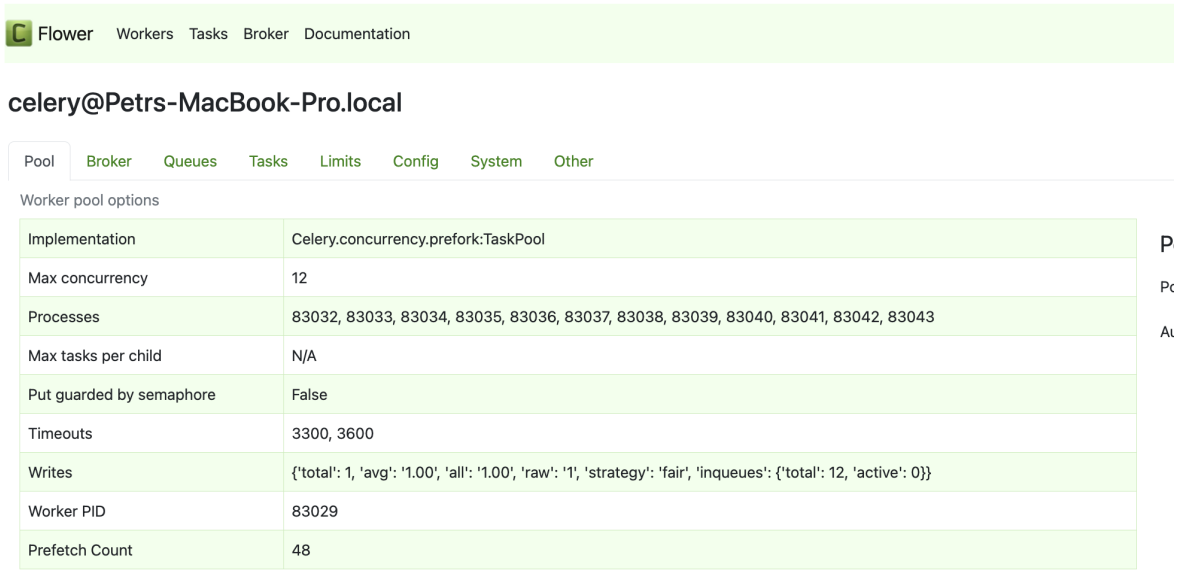
Note: Created by author using Plantuml.com

6.1.6 Celery Setup and Flower Monitoring Tool

Tasks dispatched to Celery workers can be monitored through a tool called Flower[4]. This web-based monitoring tool provides a dashboard interface exposing live metrics on worker pool configuration, task execution status, and various other runtime observables. It primarily serves as a development and debugging aid, though it can equally be deployed in a production environment for server inspection when needed.

The Figure 6.5 below shows a testing setup with maximum concurrency set to 12, meaning up to 12 workers can be spawned in parallel to process tasks simultaneously. Another notable metric is the Prefetch Count of 48, exactly four times the concurrency level. This value determines how many tasks each worker process may pre-fetch from the Redis broker queue before its current task has completed, helping to reduce idle time between task assignments.

Figure 6.5
Flower Tool - Celery Setup



Note: Screenshot by Author.

While a task is being processed by a worker, its status can be monitored in the Tasks tab, see Figure 6.6. The figure below shows a successfully completed task, in which the platform executed a UIC-Miner operation. This view provides information about the task state, its result, and the timestamps for when execution started and finished.

[4]https://github.com/mher/flower

Figure 6.6
Flower Tool - Celery Task View

Note: Screenshot by Author.

6.1.7 Service Layer

The application exposes its functionality through a RESTful API implemented using the Django REST Framework. The Service layer follows a resource-oriented design, with each endpoint corresponding to a domain entity: Task, Run, Dataset, and Project, managed through DRF's ViewSet abstraction. Only the key endpoints most relevant to the mining pipeline described in the previous subchapter are covered in this thesis.

The `create_run_and_execute` method, as its name suggests, creates a new Run and immediately dispatches it to the execution pipeline. It does so by instantiating a new Run from the task's current parameters stored in the `params` attribute, then passing it to `enqueue_run`, presented in Source code 6.13. Should the enqueueing fail, a 409 Conflict response is returned. On success, the serialized Run object is returned with a 202 Accepted status, indicating the task has been accepted for asynchronous processing.

Source code 6.13
create_run_and_execute API Action

```
@action(detail=True, methods=["post"])
def create_run_and_execute(self, request, pk=None):
    task = self.get_object()
    run = create_run(task=task)

    try:
        enqueue_run(run=run)
    except RunEnqueueError as e:
        return Response(
            {"error detail": str(e)}, status=status.
                HTTP_409_CONFLICT
        )

    return Response(RunSerializer(run).data, status=status.
        HTTP_202_ACCEPTED)
```

The `enqueue_run` function, presented in Source code 6.14, accepts a Run object as a keyword-only argument and returns an `EnqueueResult` dataclass. Before the run is enqueued, two idempotency guards are evaluated to prevent invalid or repeated submissions. The first guard checks for duplicate submission attempts by verifying whether the run is already queued with an assigned Celery task ID. The second enforces valid state transitions, ensuring the run is only dispatched from an eligible status.

If both guards pass, the Celery task is dispatched asynchronously via `.delay()`, which serializes the `run.id` and publishes it to the Redis broker without blocking the calling thread. The returned `AsyncResult` object carries the assigned Celery task identifier, which is immediately persisted to the Run record alongside a status update to QUEUED.

Source code 6.14

enqueue_run Function

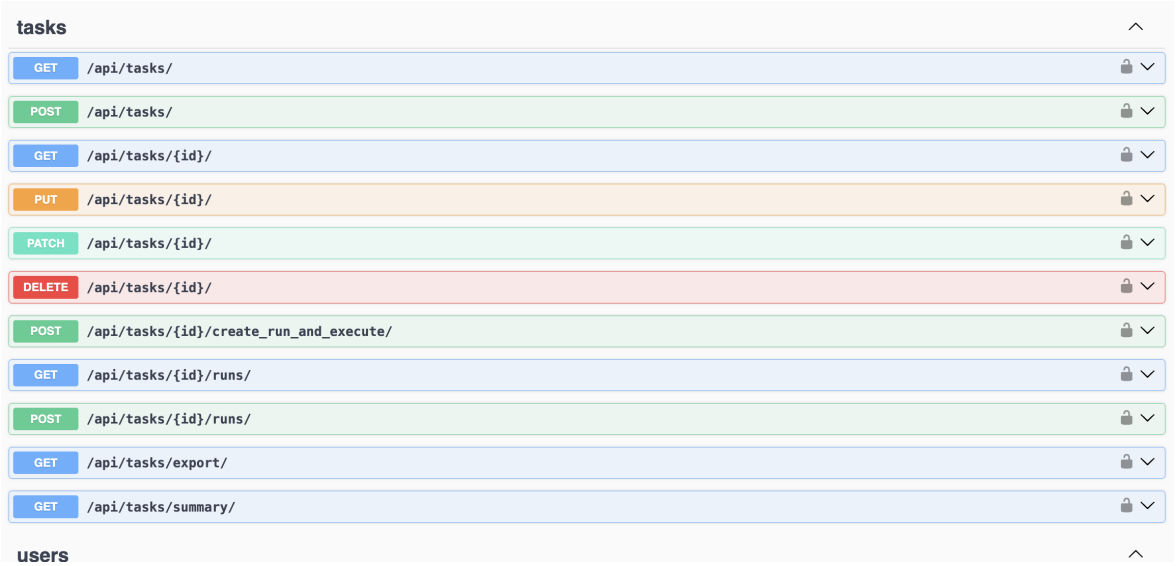
```
1  def enqueue_run(*, run: Run) -> EnqueueResult:
2      if run.status == RunStatus.QUEUED and run.celery_task_id:
3          raise RunEnqueueError("Run is already queued.")
4
5      if run.status not in (RunStatus.QUEUED, RunStatus.FAILED):
6          raise RunEnqueueError(
7              f"Run cannot be executed from status '{run.status}'."
8          )
9
10     async_result: AsyncResult = execute_runner_for_tasks.delay(run
           .id)
11
12     run.celery_task_id = async_result.id
13     run.status = RunStatus.QUEUED
14     run.save(update_fields=["celery_task_id", "status"])
15
16     return EnqueueResult(run=run, celery_task_id=async_result.id)
```

The full API is also documented and accessible through a Swagger[5] interface, available for development and testing purposes. The swagger is available on the `/api/docs` path on the backend URL. It exposes both the standard CRUD endpoints generated automatically by the `ViewSet` abstraction and any custom actions, such as the `create_run_and_execute` endpoint discussed above, which are not generated automatically and must be explicitly defined. The Swagger view for the Task entity is shown in the Figure 6.7 below.

[5] https://swagger.io/

Figure 6.7

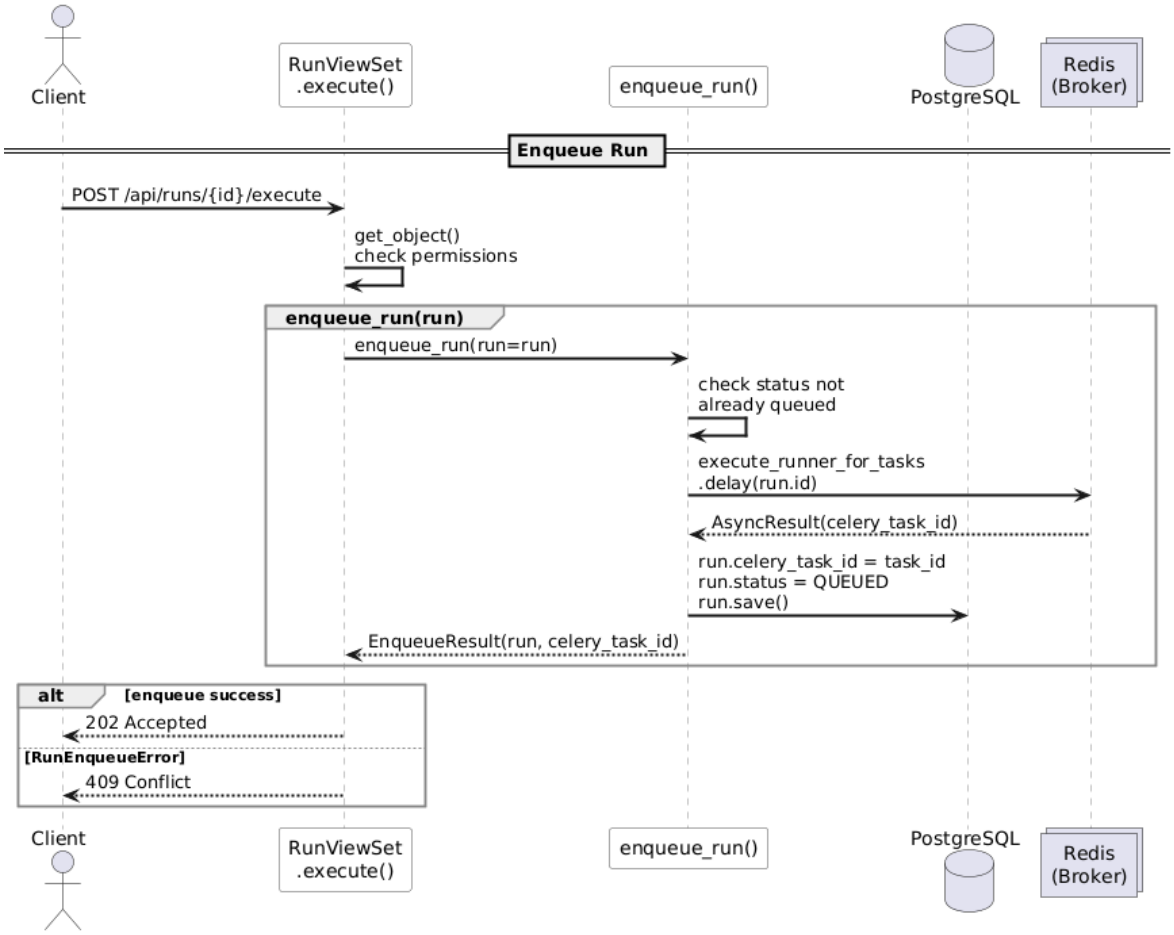
Swagger - API Documentation



Note: Screenshot by Author.

To conclude the Service layer, a sequence diagram is presented in the Figure 6.8, providing a clearer visualisation of the full call stack triggered when a user initiates a run through the frontend. The diagram traces the entire lifecycle of the request, from the initial API call through to the point where the task is published to the Redis message broker.

Figure 6.8
Sequential Diagram - Service Layer



Note: Created by author using Plantuml.com

6.2 Frontend Implementation

The frontend implementation is described through a series of screenshots illustrating how users interact with the analytical platform. Due to the scope constraints of this thesis, only the most essential parts of the application are covered in this chapter. It also includes selected code examples demonstrating how React, TanStack Query, and Axios are employed within the frontend architecture.

6.2.1 Frontend Overview

The frontend is clearly structured into feature modules, each encapsulating one primary domain of the application, see Source code 6.15. The `lib` folder hosts the `api-client.ts` file, which sets up the Axios `apiClient` that will be described further below, alongside a general `utils.ts` file. The shared folder contains code reused throughout the application, including shared API utilities, hooks, domain types, and the components folder, which also

houses the Shadcn UI component library in its `ui` subfolder.

Source code 6.15

Frontend Project Structure

```
|-- app.css
|-- lib
|   |-- api-client.ts
|   `-- utils.ts
|-- modules
|   |-- auth
|   |-- dashboard
|   |-- datasets
|   |-- homePage
|   |-- projects
|   |-- runs
|   `-- tasks
|-- root.tsx
|-- routes.ts
`-- shared
    |-- api
    |-- components
    |-- domain
    |-- hooks
    `-- utils
```

Each module follows a consistent internal structure, as shown in Source code 6.16. Modules are divided into a components folder and a pages folder. Pages represent the top-level components rendered directly to the user, while components define the smaller building blocks from which pages are composed. The components folder is further organised according to the Atomic Design principle, separating UI elements into atoms, molecules, and organisms. This clear structure enforces separation of concerns and facilitates component reuse across multiple pages.

Source code 6.16

Module Internal Structure

```
|-- api
|   `-- tasks.api.ts
|-- components
|   |-- atoms
|   |-- molecules
|   `-- organisms
|-- domain
|   |-- quantifier-definitions.ts
|   |-- task-run.type.ts
|   |-- task-schema.ts
|   `-- task.type.ts
|-- hooks
```

```
13 |   `-- tasks.hook.ts
14 |-- pages
15 |   |-- EditTask.page.tsx
16 |   |-- NewTask.page.tsx
17 |   |-- TaskDetail.page.tsx
18 |   |-- Tasks.layout.tsx
19 |   `-- Tasks.page.tsx
20 `-- utils
21     |-- logic-layout.ts
22     `-- task-validation.ts
```

6.2.2 API Layer

The frontend communicates with the Django backend through a centralised Axios client instance, configured via `axios.create()` in Source code 6.17. This approach establishes a shared base configuration reused across all API calls, eliminating repetitive setup at individual request sites and providing a single point of modification. The client is initialised with the backend URL sourced from an environment variable, JSON as the default content type, and credentials included by default to support session-based authentication.

Source code 6.17

Axios Client Configuration

```
1 export const apiClient = axios.create({
2   baseURL: import.meta.env.VITE_BACKEND_URL,
3   withCredentials: true,
4   headers: {
5     'Content-Type': 'application/json',
6   },
7 });
```

To fully leverage the Axios instance and integrate it with TanStack Query across pages and components, each module contains a dedicated `api` folder housing the typed service functions responsible for backend communication. This approach abstracts all HTTP communication behind single-purpose functions, each handling one specific API interaction. Source code 6.18 shows the `getTask` and `getDatasets` functions as implemented in the tasks module.

Source code 6.18

Task API Service Functions

```
1 export async function getTask(id: number): Promise<Task> {
2   const res = await apiClient.get(`/tasks/${id}/`);
3   return res.data;
4 }
5
6 export async function getDatasets(): Promise<DatasetType[]> {
7   const res = await apiClient.get('/datasets/');
```

```
8     return res.data;
9   }
```

This architecture enables clean integration with the TanStack Query[6] library for server state management, handling data fetching, caching, and frontend-backend synchronisation. The library significantly reduces boilerplate code and eliminates the need for manual loading and error state management.

Code in Source code 6.19 demonstrates how `useQuery` and `useMutation` are leveraged in practice. Task fetching is handled through the `useQuery` hook, which accepts a `queryKey` and a `queryFn`. The `queryKey` acts as a unique cache identifier, enabling TanStack Query to cache, deduplicate, and invalidate data automatically. Mutations are handled through the `useMutation` hook, which provides `onSuccess` and `onError` callbacks for side effect management. Additionally, by utilising `queryClient.invalidateQueries()`, cached data can be manually invalidated upon a successful mutation, triggering an automatic refetch to synchronise the frontend with the latest backend state without requiring a full page reload.

Source code 6.19

TanStack Query Usage

```
1  const { data, isLoading } = useQuery({
2    queryKey: ['tasks'],
3    queryFn: getTasks,
4  });
5
6  const { data: tasksSummaryData, isLoading: tasksSummaryLoading } =
       useQuery({
7    queryKey: ['tasks-summary'],
8    queryFn: () => getTasksSummary(),
9  });
10
11 const exportTaskMutation = useMutation({
12   mutationFn: () => exportTasks(),
13   onError: (error: any) => {
14     toast.error('Export failed:', error.message);
15   },
16 });
17
18 const deleteTaskMutation = useMutation({
19   mutationFn: (taskId: number) => deleteTask(taskId),
20   onError: (error: any) => {
21     toast.error('Delete failed:', error.message);
22   },
23   onSuccess: () => {
24     toast.success('Task deleted successfully');
25     queryClient.invalidateQueries({ queryKey: ['tasks'] });
```

[6] https://tanstack.com/query/latest

```
26     },
27   });
```

6.2.3 Frontend Presentation Layer Overview

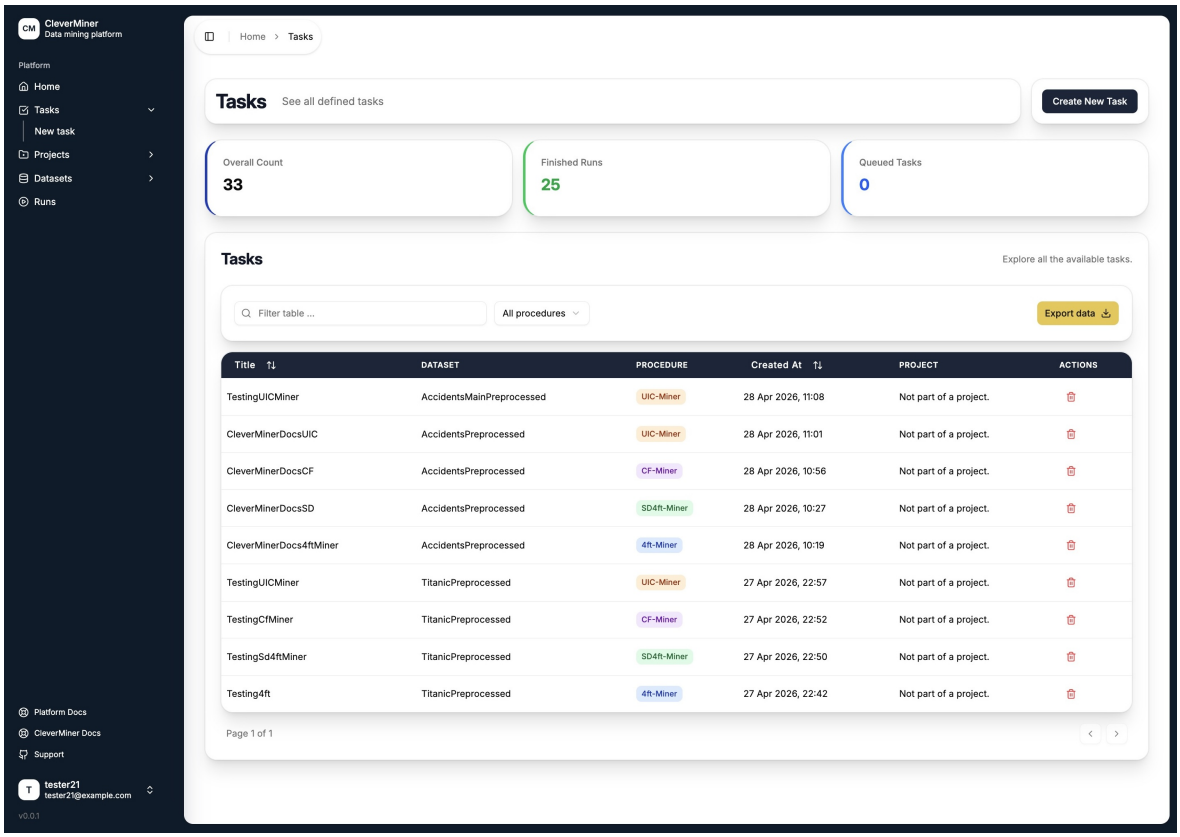
To further present the implemented platform, this chapter showcases the most important parts of the application that users will interact with during typical workflows, including task definition, dataset exploration, and mining result analysis.

As established in previous chapters, the frontend code is structured in accordance with Atomic Design principles. As a result, several key components are reused throughout the entire platform, such as the `DataTable` component, `BaseSummaryCard`, `PlatformCard`, and `ModulePagesHeader`. This approach streamlines development and ensures a consistent user experience, as design patterns and visual language remain uniform across all parts of the platform.

6.2.4 Tasks Main Page

The Tasks main page in Figure 6.9 provides an overview of all current tasks. Data is displayed primarily through the `DataTable` component and `BaseSummaryCard` components. The table presents the most relevant information for each task: its title, the selected dataset, the procedure it was defined with, the creation timestamp, and an actions column allowing the user to delete the task. Pagination is implemented to prevent the page from growing indefinitely.

Figure 6.9
Frontend - Tasks Main Page



Note: Screenshot by Author from the Platform.

Thanks to the shared component architecture, the Tasks main page is composed of just a few key elements, as shown in Source code 6.20. The `DataTable` component is shared across many pages and is built on top of the Radix UI `DataTable` component. It supports multiple table definitions within a single component through the columns property, which accepts column headers and data access keys allowing each entity to be described independently through its own column definition object.

Source code 6.20
Tasks Main Page Composition

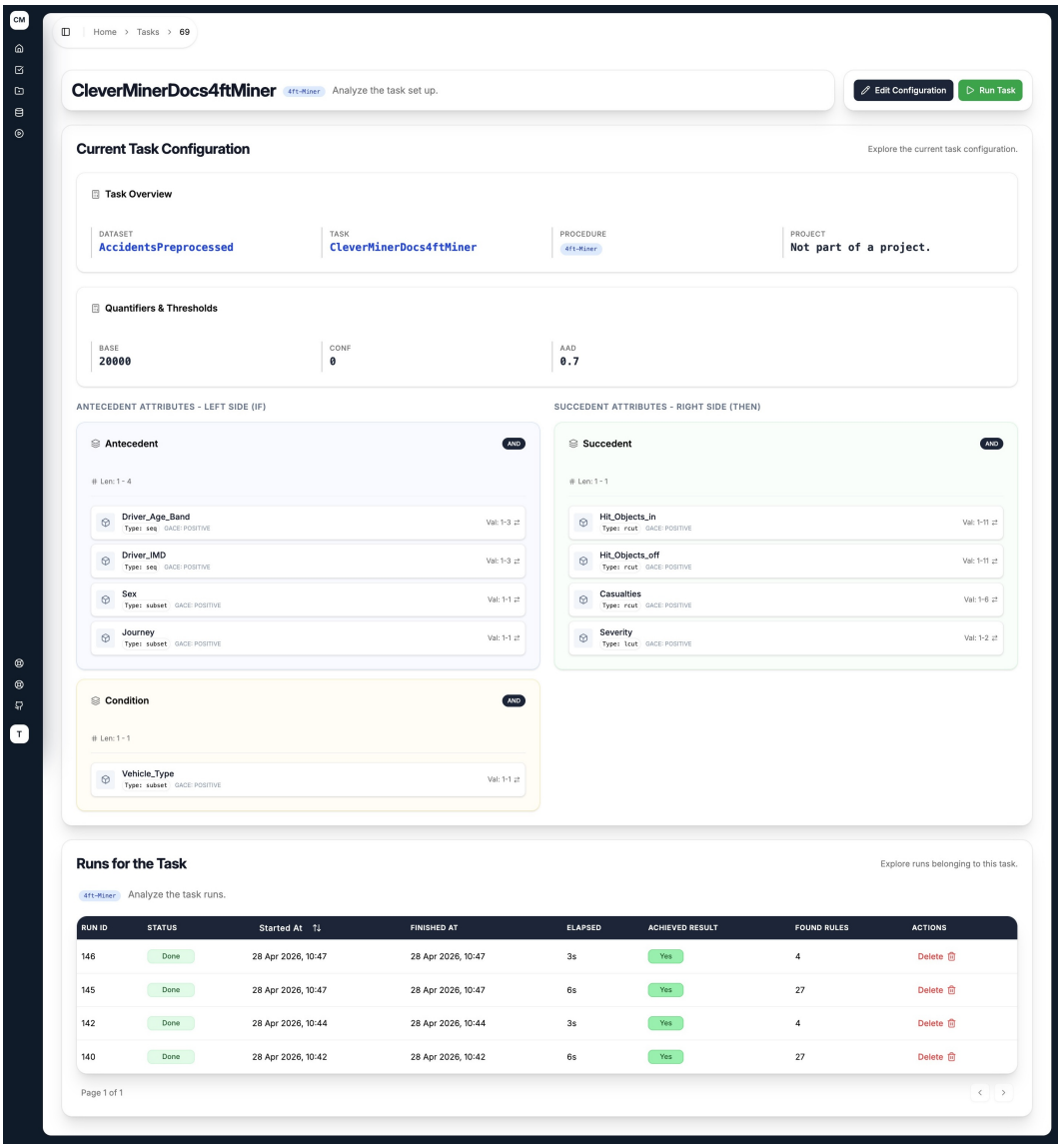
```
<div>
  <ModulePagesHeader title={'Tasks'} description={'See all defined
      tasks'}>
    <Link to={'/tasks/new-task'}>
      <Button>Create New Task</Button>
    </Link>
  </ModulePagesHeader>
  <div className="mb-6 grid gap-4 md:grid-cols-3">
    <BaseSummaryCard
      title={'Overall Count'}
      value={tasksSummaryData.total}
```

```
11          variant={'default'}
12        />
13        <BaseSummaryCard
14          title={'Finished Runs'}
15          value={tasksSummaryData.done}
16          variant={'success'}
17        />
18        <BaseSummaryCard
19          title={'Queued Tasks'}
20          value={tasksSummaryData.queued}
21          variant={'running'}
22        />
23      </div>
24      <div className="space-y-5">
25        <PlatformCard
26          cardTitle={'Tasks'}
27          cardDescription={'Explore all the available tasks.'}
28          titleClassName={'text-2xl font-bold tracking-tight text-gray
              -900'}
29        >
30          <DataTable
31            columns={TaskColumns}
32            data={data}
33            showSearch={true}
34            onRowClick={(row) => navigate('/tasks/' + row.id)}
35            exportData={exportTaskMutation.mutate}
36            initialSorting={[{ id: 'created_at', desc: true }]}
37          />
38        </PlatformCard>
39      </div>
40    </div>
```

6.2.5 Task Detail - 4ft-Miner and SD4ft-Miner

The Task detail page presents relevant information about a defined task, see Figure 6.10. It displays the current task configuration rendered through the shared `CedentDetail` and `QuantifiersDetail` components. Below the configuration, a `DataTable` component lists all runs associated with the task, showing key information such as whether the run completed successfully, whether any rules were returned, and if so, their count. The table also includes runs that are currently queued and awaiting execution by a Celery worker. In addition to the data view, the page exposes two primary actions: executing the task with its current configuration, and editing the configuration through the setup wizard.

Figure 6.10
Frontend - 4ft-Miner Task Detail



Note: Screenshot by Author from the Platform. This is a full-screen screenshot, that is the reason why the sidebar is malformed.

Since this page is conceptually identical across all procedures but each procedure requires a different configuration view, a Render Function pattern is employed to dynamically determine what the page should display based on the procedure type. As shown in Source code 6.21, the correct view is rendered simply by passing the return value of the renderer function as a child component.

Source code 6.21
Render Function Pattern Usage

```
1 <PlatformCard
2   cardTitle={'Current Task Configuration'}
3   cardDescription={'Explore the current task configuration.'}
```

```
4    titleClassName={'text-2xl font-bold tracking-tight text-gray
       -900'}
5  >
6    <div className="animate-in fade-in duration-500">
7      {renderProcedureDetails(task)}
8    </div>
9  </PlatformCard>
```

The `renderProcedureDetails` function, shown in Source code 6.22, is reused in multiple places throughout the platform. Based on the task's procedure type, it returns the appropriate detail component for that procedure. If an unrecognised procedure type is encountered, the function returns an Alert indicating that the procedure render failed. In practice, however, this fallback should never be reached, as the application enforces a strict set of permitted procedure types throughout.

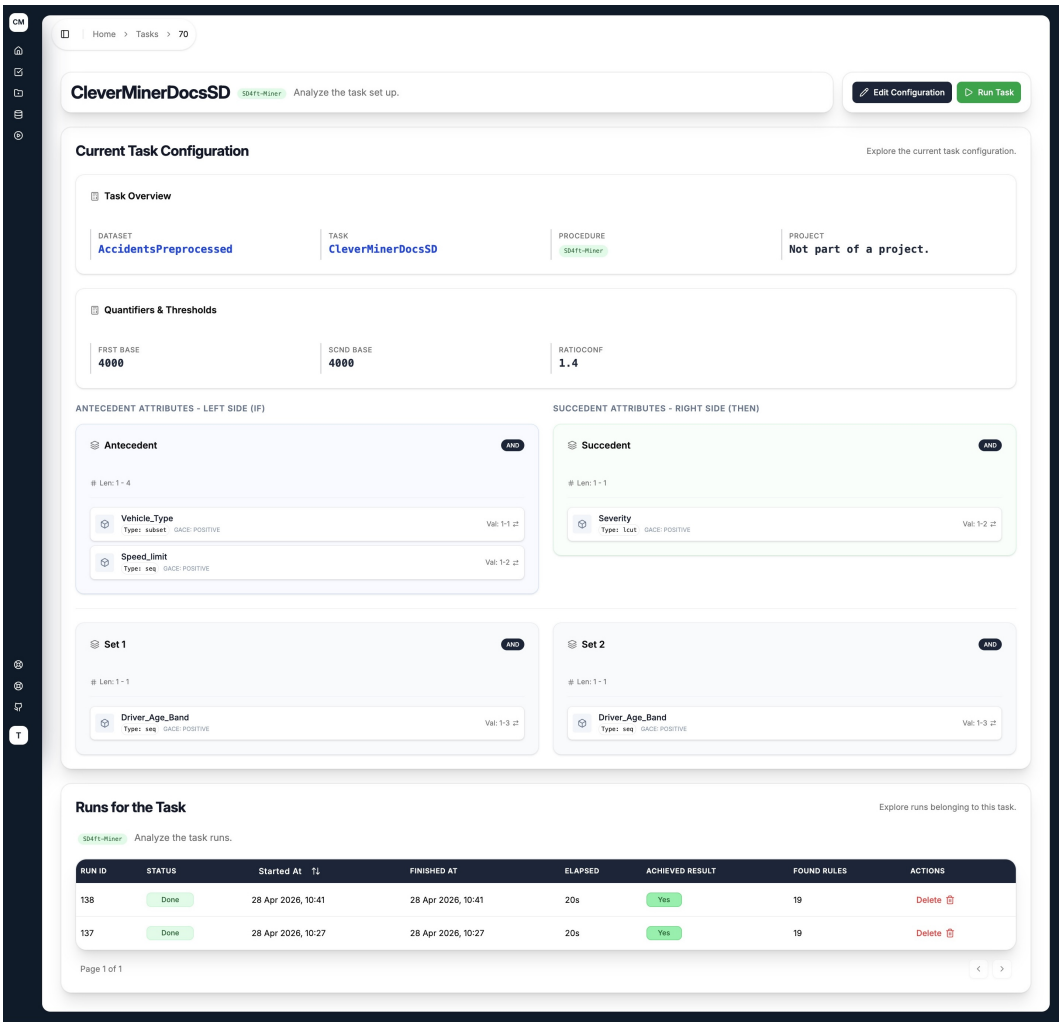
Source code 6.22

renderProcedureDetails Function

```
1  export const renderProcedureDetails = (task: Task) => {
2    switch (task.procedure) {
3      case ProceduresType.SD4FTMINER:
4        return <SD4ftMinerDetails params={task.params} />;
5      case ProceduresType.CFMINER:
6        return <CFMinerDetails params={task.params} />;
7      case ProceduresType.FOURFTMINER:
8        return <FourFtMinerDetails params={task.params} />;
9      case ProceduresType.UICMINER:
10       return <UICMinerDetails params={task.params} />;
11     default:
12       return (
13         <Alert>
14           <Terminal className="h-4 w-4" />
15           <AlertTitle>Unknown Procedure</AlertTitle>
16           <AlertDescription>
17             Procedure renderer failed.
18             <pre className="mt-2 w-full overflow-x-auto rounded
19               bg-slate-950 p-4 text-xs text-slate-50">
20               {JSON.stringify(task.params, null, 2)}
21             </pre>
22           </AlertDescription>
23         </Alert>
24       );
25   }
26 };
```

Thanks to the Render Function pattern, all procedures share the same page structure, with procedure-specific data handled entirely within the rendering function. The Figure 6.11 below shows an example of an SD4ft-Miner task definition as displayed on the Task detail page.

Figure 6.11
Frontend - SD4ft-Miner Task Detail



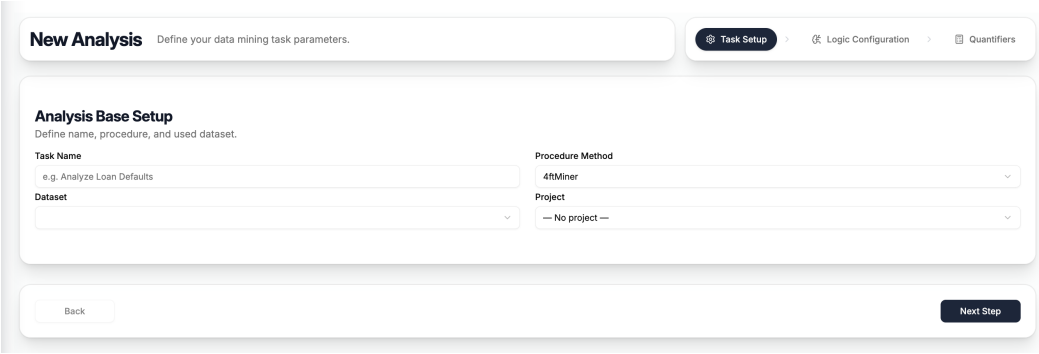
Note: Screenshot by Author from the Platform. This is a full-screen screenshot, that is the reason why the sidebar is malformed.

6.2.6 Task Wizard - Creating New Analysis

New tasks are defined through the New Analysis form wizard, which guides the user through the task definition process in three steps. The first step, see Figure 6.12, captures the basic task information: the task name, the procedure to be used, the dataset selected from a dropdown menu, and an optional Project field, listing the projects the user is already a member of. The entire form is wrapped with react-hook-form's[7] `FormProvider`, with input validation handled through the Zod library[8].

[7] https://react-hook-form.com/
[8] https://zod.dev/

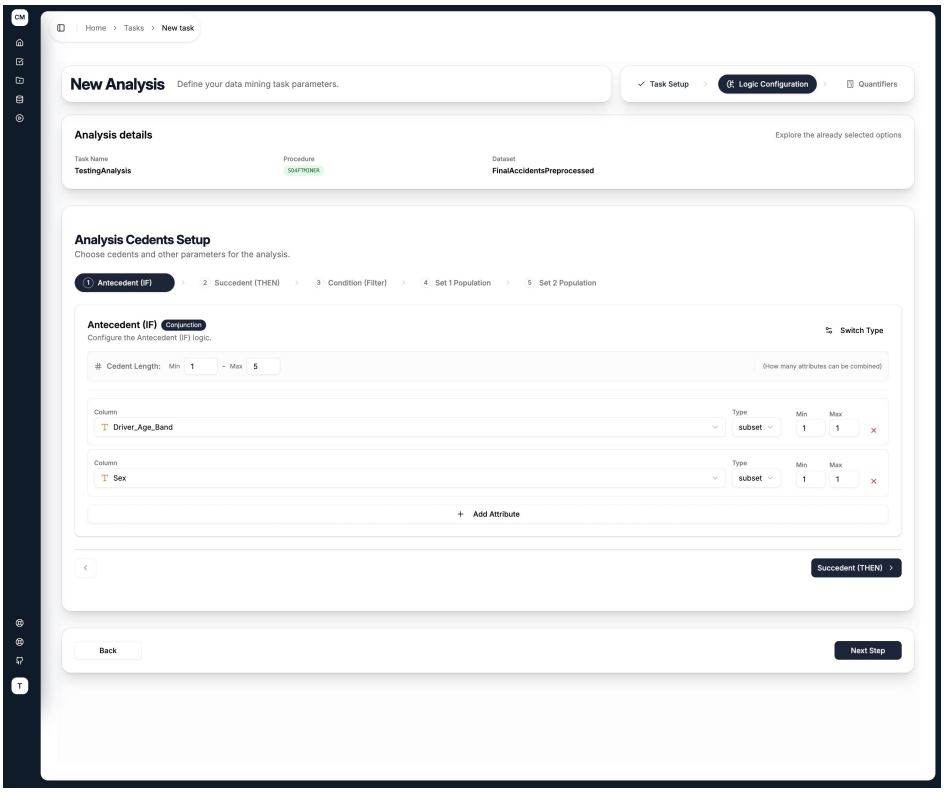
Figure 6.12
Frontend - Task Wizard - First Step



Note: Screenshot by Author from the Platform.

The second step handles the core configuration of the GUHA mining procedure. As shown in the Figure 6.13, where an SD4ft-Miner procedure is being configured, the available options are presented to the user as a series of tabs.

Figure 6.13
Frontend - Task Wizard - Second Step



Note: Screenshot by Author from the Platform. This is a full-screen screenshot, that is the reason why the sidebar is malformed.

As described in the Chapter 2, the SD4ft-Miner requires an antecedent, succedent, condition,

and two additional sets, each of which is rendered as a separate tab with its own form. The visible tabs are determined by the `LOGIC_LAYOUTS` constant, which maps each procedure type to its required sections, allowing the form to dynamically render only the tabs relevant to the selected procedure.

Source code 6.23
visibleSections and LOGIC_LAYOUTS

```
const visibleSections = LOGIC_LAYOUTS[procedure] || [];

export const LOGIC_LAYOUTS: Record<string, LogicSection[]> = {
  SD4ftMiner: ['ante', 'succ', 'cond', 'set1', 'set2'],
  CFMiner: ['target', 'cond'],
  UICMiner: ['target', 'ante', 'cond'],
  fourftMiner: ['ante', 'succ', 'cond'],
};
```

The form allows users to select attributes from a dropdown populated by the dataset's columns. For each attribute within a cedent, a type selection field is provided to specify whether it should be treated as a subset, left cut or right cut and as type one. The `CedentEditor` also allows the minimum and maximum length of each attribute to be specified. Above the cedent list, two additional fields define the overall length constraints of the entire antecedent or succedent. In the top-right corner, a toggle switch determines whether the cedent should be evaluated as a conjunction or disjunction.

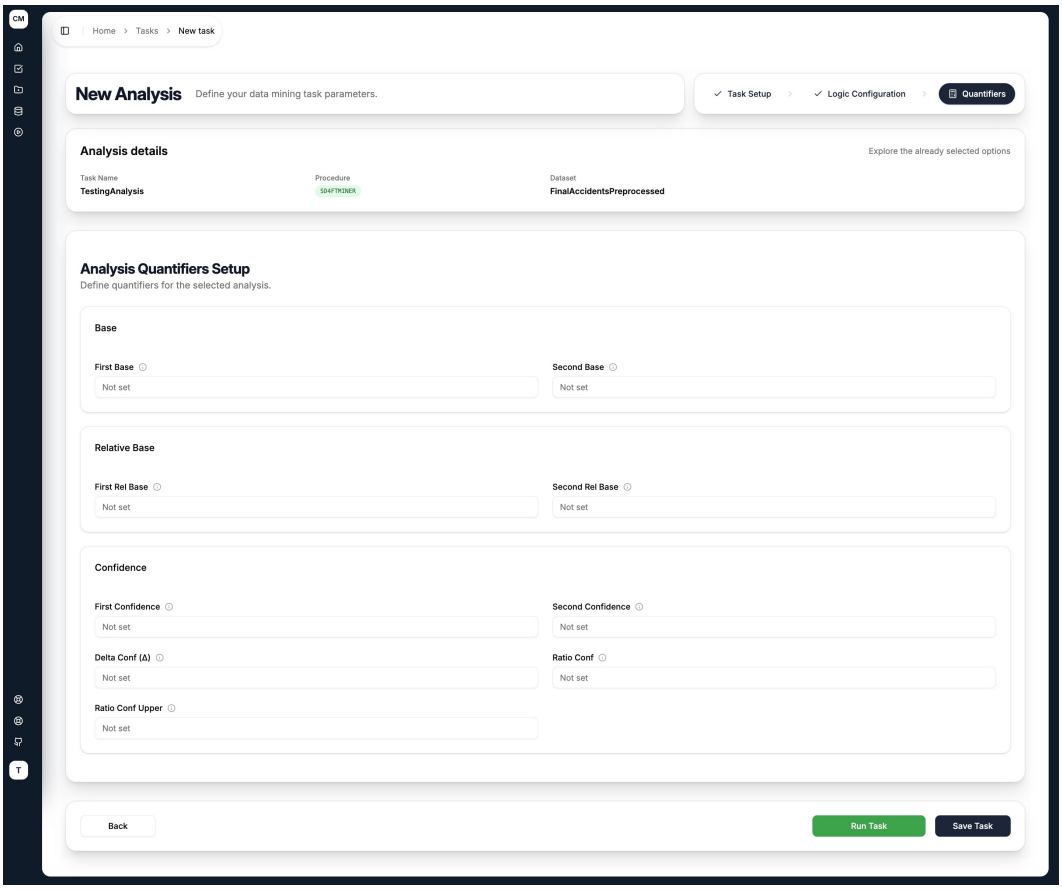
The final step of the form handles quantifier configuration, see Figure 6.14. The input fields are dynamically generated based on the selected procedure type, using the constant `QUANTIFIER_SCHEMAS`, which defines the available quantifier inputs for each procedure. At the bottom of the form, two actions are provided: saving the task for later execution, or saving it and immediately dispatching it to the execution pipeline. The latter is handled by the `useCreateTaskAndRunMutation` hook Source code 6.24, located in `tasks.hook.ts`, which orchestrates two sequential API calls: `createTask()` to persist the task definition, followed by `createAndExecuteRun()` to instantiate a run and immediately enqueue it for execution.

Source code 6.24
useCreateTaskAndRunMutation Hook

```
export function useCreateTaskAndRunMutation() {
  const navigate = useNavigate();
  return useMutation({
    mutationFn: async (data: any) => {
      const task = await createTask(data);
      await createAndExecuteRun(task.id);
      return task;
    },
    onSuccess: (task) => {
      toast.success(`Task "${task.name}" created and started!`);
```

```
11          navigate(`/tasks/${task.id}`);
12        },
13        onError: (error: any) => {
14          console.error(error);
15          toast.error(
16            'Failed to create task: ' +
17            (error.response?.data?.detail || error.message)
18          );
19          alert(
20            'Error: ' +
21            (error.response?.data?.detail || 'Something went wrong')
22          );
23        },
24      });
25    }
```

Figure 6.14
Frontend - Task Wizard - Third Step



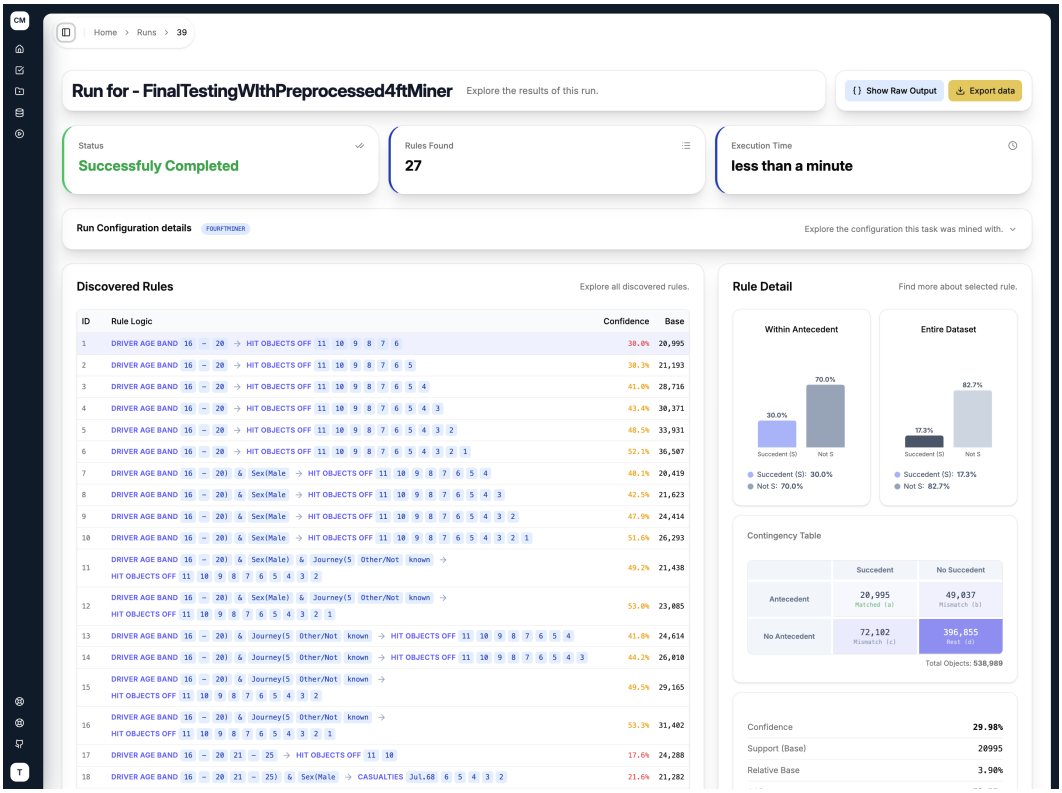
Note: Screenshot by Author from the Platform. This is a full-screen screenshot, that is the reason why the sidebar is malformed.

6.2.7 Run Details

The Run detail page displays the results of successfully completed runs. Since the platform implements all four available CleverMiner procedures, the page dynamically adjusts its components based on the procedure type returned by the API, as each procedure produces a slightly different result structure.

One of the core requirements and goals of this thesis is that results are easily interpretable and accessible to domain users without prior knowledge of data mining. The results page is therefore structured around several key components: a rules list accompanied by statistical indicators such as confidence and support for each discovered rule, and a set of visual aids, including histograms, four-fold tables, and additional statistical data, to assist in interpreting selected rules. The page also provides an option to display the configuration under which the run was executed, sparing the user from having to navigate back to the task.

Figure 6.15
Frontend - 4ft-Miner - Results



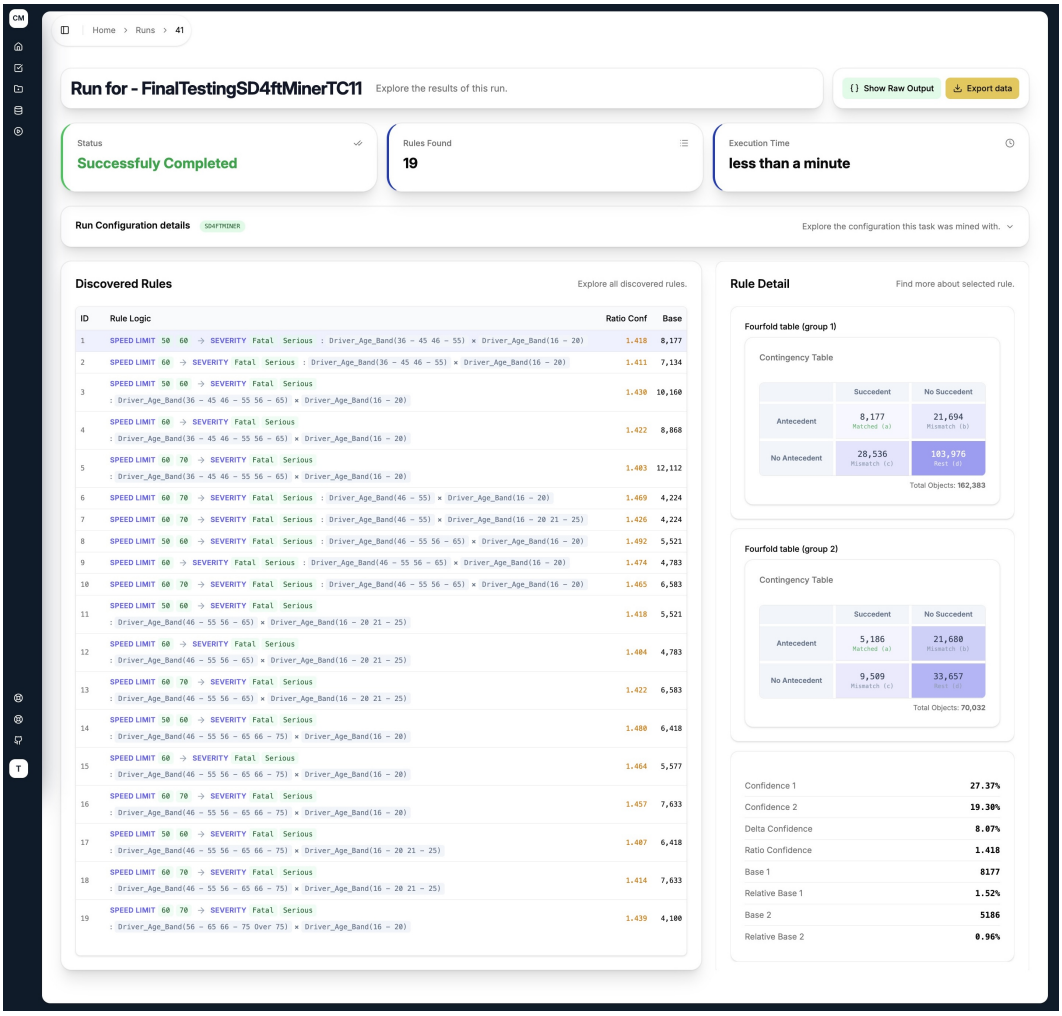
Note: Screenshot by Author from the Platform.

The Figure 6.15 shows the results page for the 4ft-Miner. Three execution statistics are immediately visible at the top: whether the run completed successfully, how many rules were discovered, and how long the mining process took. Below this summary, a collapsible component allows the user to inspect the `run_snapshot` data, rendered through the

`RunConfigurationDetails` component, which reuses the same configuration display components as the Task detail page. The main section consists of two parts: the Discovered Rules section, which presents all discovered rules in a scrollable container with colour-coded attribute values, with colour coding specific to each miner type for improved clarity and the Rule Detail section, which updates dynamically upon rule selection to display statistical data, in the case of the 4ft-Miner comprising two histograms and one four-fold table.

To illustrate the differences between miners, the SD4ft-Miner results page is presented in the Figure 6.16. This miner produces two histograms, two four-fold tables along with additional statistical data, and applies a distinct colour coding scheme compared to the 4ft-Miner.

Figure 6.16
Frontend - SD4ft-Miner - Results



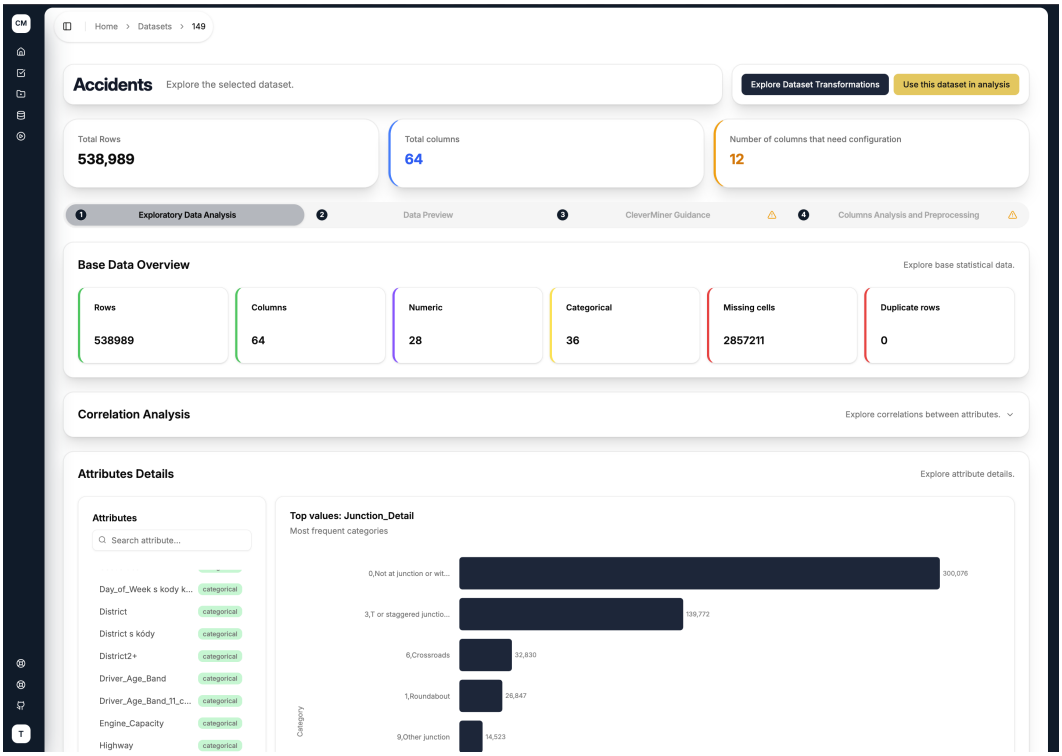
Note: Screenshot by Author from the Platform. This is a full-screen screenshot, that is the reason why the sidebar is malformed.

6.2.8 Datasets

The platform provides broad support for the CRISP-DM methodology. The final step addressed is data exploration and preparation, handled through the Datasets module in the frontend. This module offers several key capabilities: a dataset preview allowing users to view their uploaded data in its raw format. data exploration through visualisations, primarily histograms, and descriptive statistics and access to CleverMiner Guidance, a heuristic tool that analyses the data structure and suggests how it should be handled, whether discretisation is required, or whether the data is ready to be used directly with the underlying library.

In addition, the module provides a suite of data preparation tools, including attribute discretisation, column removal, and missing value imputation. Each of these operations supports multiple strategies, giving users flexibility in how they achieve the desired preprocessing outcome.

Figure 6.17
Frontend - Dataset Exploratory Analysis



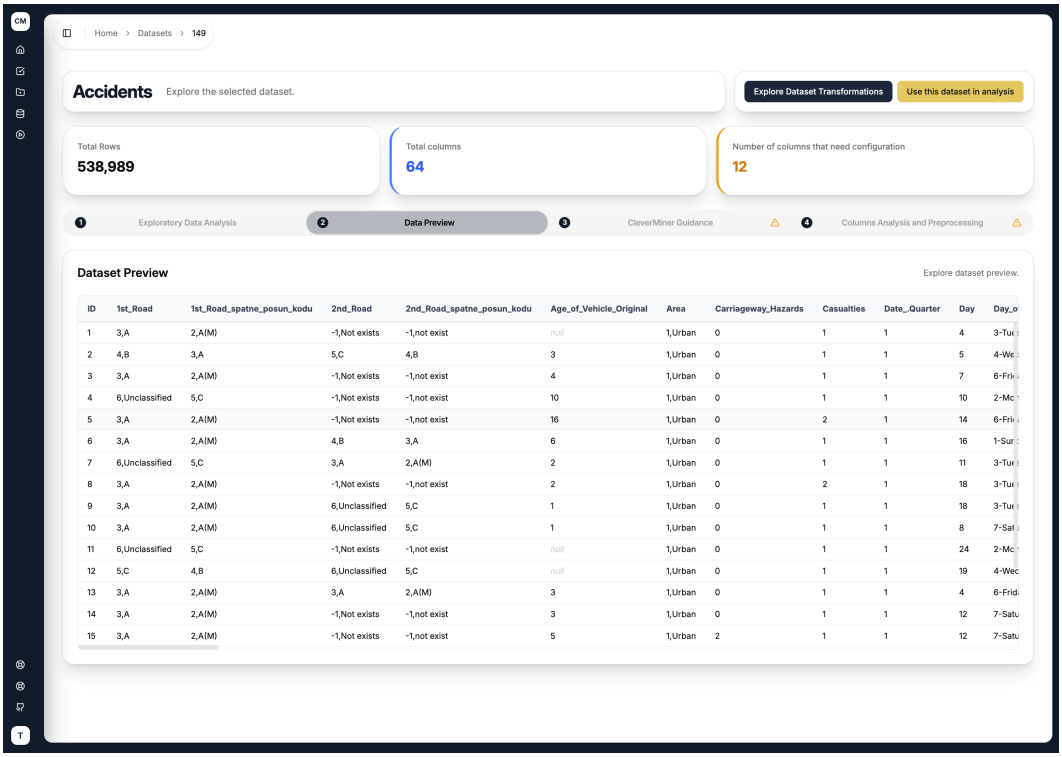
Note: Screenshot by Author from the Platform.

The dataset detail page is organised into four tabs, corresponding to the functionalities described in the previous paragraph. The Exploratory Data Analysis tab, presented in Figure 6.17, provides an overview of the dataset's base statistical properties and allows users to examine each column in detail. The Base Data Overview section summarises key metrics such

as the counts of numeric and categorical columns and the number of missing values across the dataset. The following section presents a correlation matrix, illustrating how each attribute relates to the others, enabling users to identify potentially redundant columns. The final section offers an interactive column explorer, where users can select any column to view either its category distribution for categorical attributes or an automatically binned histogram for numerical ones.

The dataset preview in Figure 6.18 displays a table of 20 rows sampled from the dataset. Horizontal scrolling is supported, allowing users to explore all available attributes regardless of the dataset's width.

Figure 6.18
Frontend - Dataset Preview

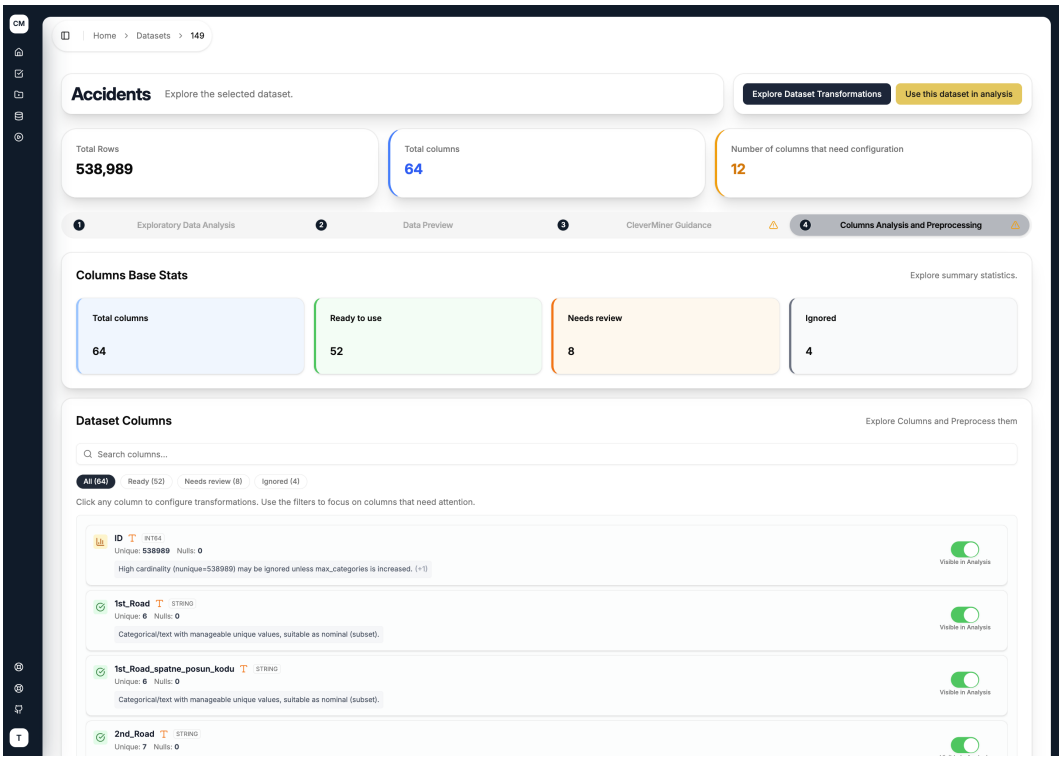


Note: Screenshot by Author from the Platform.

The Columns Analysis and Preprocessing tab presented in Figure 6.19 begins with a Columns Base Stats summary, giving users an at a glance indication of how many columns likely require preparation. This tab applies stricter criteria when classifying columns as either ignored or requiring review. Unlike the CleverMiner Guidance tab, which focuses on identifying attributes that are immediately usable, this tab evaluates all columns in the dataset and flags those that may need preprocessing before mining can begin.

The main Dataset Columns section presents a list of all columns present in the dataset. The Visible in Analysis switch can hide the attributes from the inputs in the already presented

Figure 6.19
Frontend - Dataset Preprocessing Page

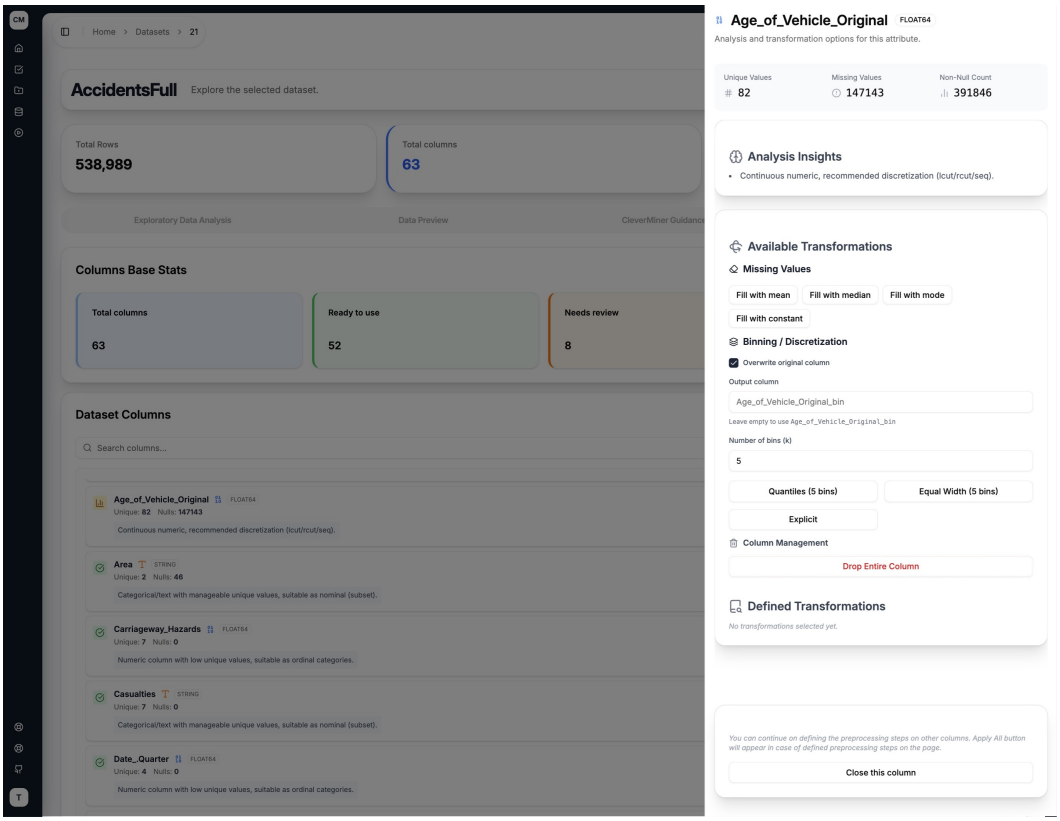


Note: Screenshot by Author from the Platform.

wizard, see Figure 6.13, for new analysis creation. Clicking on a column opens a drawer component, presented in Figure 6.20, from the right side of the page, exposing the available preprocessing options for the selected column. The drawer summarises the column's key statistics and then lists the Available Transformations. Users can fill in missing values using one of four strategies: mean, median, mode, or a user-specified constant. Discretisation is also available, with multiple binning strategies supported and the option to either create a new column or overwrite the existing one. Finally, the drawer provides the option to permanently delete the column from the dataset.

Figure 6.20

Frontend - Dataset Preprocessing Drawer

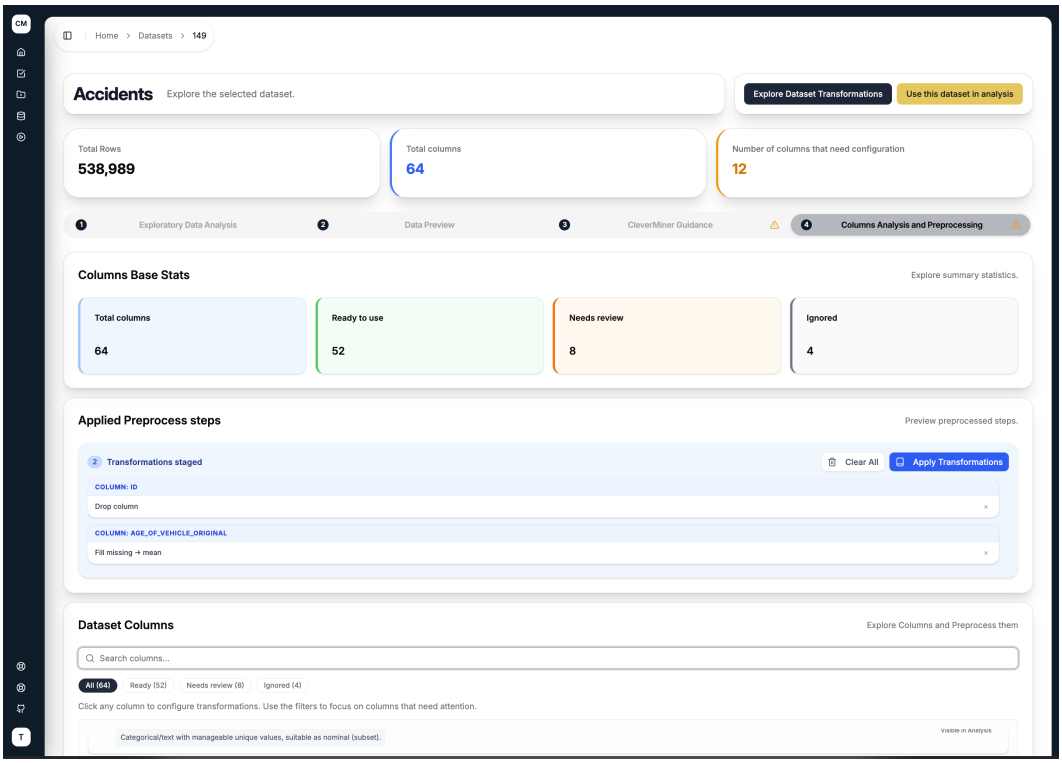


Note: Screenshot by Author from the Platform.

Users can iterate through all columns and apply preprocessing steps to each as needed. These operations are dispatched to the Celery workers in the backend, ensuring the web server remains free to handle other requests concurrently.

The selected preprocessing steps are accumulated in the Applied Preprocessing Steps section, see Figure 6.21, where users can review them for correctness, remove individual steps, or clear all pending operations at once. Once satisfied, user can continue with the transformation, by clicking on Apply Transformations button, in a opened modal and input name of the dataset, that will be created from these transformations.

Figure 6.21
Frontend - Dataset Preprocessing Steps Preview



Note: Screenshot by Author from the Platform.

6.3 Documentation

The documentation, implemented using Docusaurus, serves end-users by explaining what the platform offers, how to install it on their preferred environment, and how to use it in practice through a prepared case study based on the official CleverMiner documentation[9].

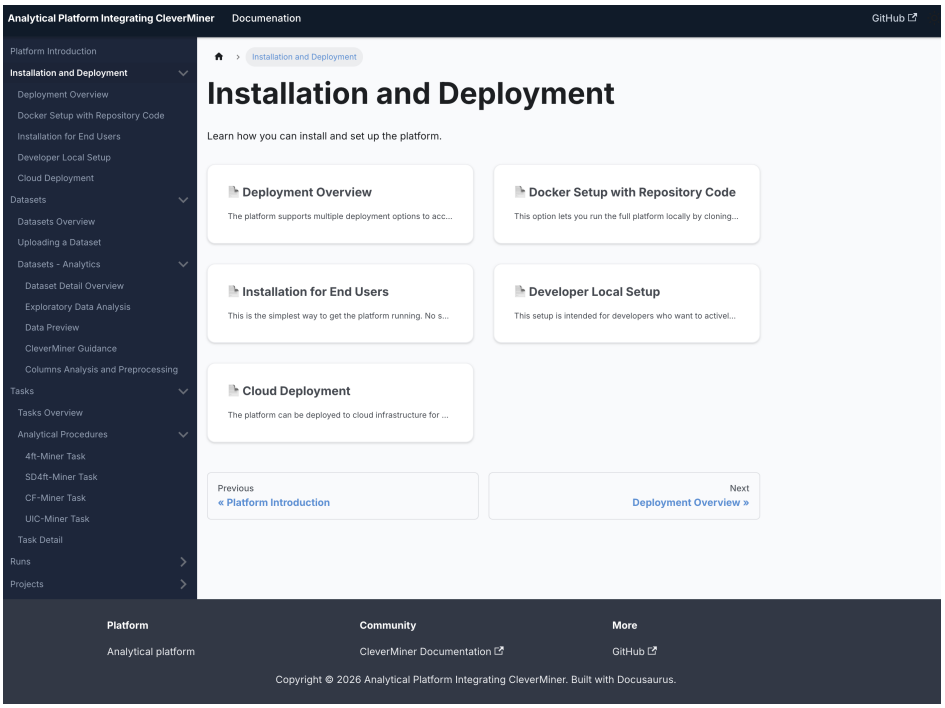
The documentation begins with the Installation and Deployment section, shown in Figure 6.22, where users can find instructions for setting up the application locally as well as deploying it to a cloud provider such as Railway.

The Datasets section guides users through exploring their dataset, obtaining preprocessing guidance, and applying the available preprocessing operations. The Tasks section, shown in Figure 6.23, demonstrates how users can define each of the implemented GUHA procedures and apply them to an available dataset. The Runs section explains how users can interact with and interpret the results of their executed tasks. Finally, the Projects section describes how users can collaborate on tasks created within the platform.

[9] https://www.cleverminer.org/doc/index.html

Figure 6.22

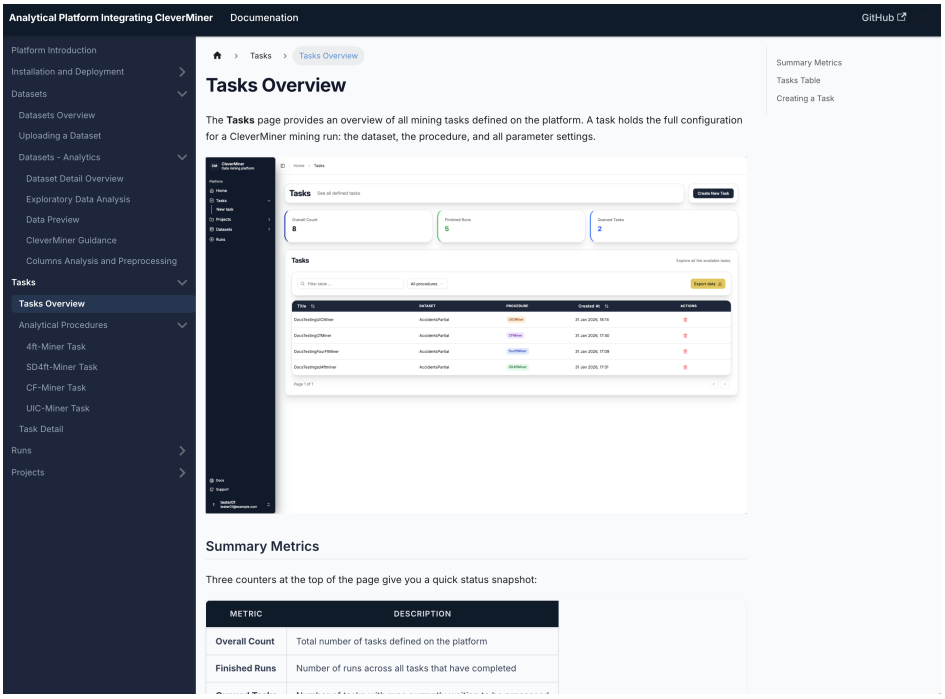
Documentation - Installation and Deployment Tab



Note: Screenshot by Author from the Docusaurus documentation.

Figure 6.23

Documentation - Tasks Overview



Note: Screenshot by Author from the Docusaurus documentation.

7. Application Deployment and Installation

One of the goals of this thesis, SO3, is to research and evaluate deployment options for the implemented platform, with the requirement that it remains accessible without a hosted cloud environment and does not demand advanced DevOps knowledge to run on personal devices. This chapter builds closely on the approaches previously explored by Vala (Vala, 2025) and Zedníčková (Zedníčková, 2025). This chaper implements the non-functional supportability requirements presented in Section 4.3.2. Additional information regarding the application setup and deployment can be found in the `README.md` file available in the repository and in the documentation for this platform, both these resources are available in Appendix A.

Zedníčková (Zedníčková, 2025) explored deployment through the Electron framework, which allows a web application to be packaged and launched as a desktop application. Vala (Vala, 2025), on the other hand, stayed within the boundaries of browser and created a Docker setup capable of starting the platform with a single terminal command.

Both approaches are valid, however, given that the platform implemented in this thesis has a considerably heavier backend, comprising Celery workers, a Redis message broker, and the Django backend itself, either approach would introduce significant setup overhead without meaningfully shielding users from the underlying complexity of Docker. The author concluded in Section 5.6, that the most appropriate solution for managing an application of this complexity is Docker, leveraging its compose capabilities for multi-container orchestration.

Some basic technical familiarity is expected from users running the application locally, but the setup has been designed to be as straightforward as possible, relying on Docker Compose and Docker Hub for image distribution.

For entirely non-technical users, a publicly accessible hosted deployment of the platform is in the time of writing this thesis available, see Appendix A. This deploy is built on the same technical stack as the local versions and is extended with S3 storage for dataset management. However, maintaining such a deployment would require a paid subscription to cover the hosting costs associated with cloud infrastructure, but is now for free.

7.1 Deployment and Installation Overview

There are four possibilities on how to deploy and use the application. All of these approaches are presented in the following sub-chapters and are also available in the documentation available in Appendix A.

- Docker setup with repository code, presented in Section 7.3

- Installation for End Users, presented in Section 7.4
- Developer Local Setup, presented in Section 7.5
- Cloud Deployment, presented in Section 7.6

The Section 7.2 introduces common ground on how the Section 7.3 and Section 7.4 work with Docker and dockerfiles, by showing base dockerfile from which the deployment implementations come from. Section 7.3 presents how users can use repository code to run the application with Docker, on the other hand the Section 7.4 describes the most user-friendly deployment, the whole application can be deployed by installing Docker and then running one command which launches the application on the local machine. Section 7.5 clearly describes how developers can run the services in the application separately, while developing on the application. Finally, the Section 7.6 section demonstrates how the application can be deployed on cloud infrastructure.

7.2 Containerization with Docker

Docker enables applications to run in isolated environments that are independent of the host machine, bundling everything the application needs to operate. These isolated environments are referred to as containers (Docker, n.d.-b).

The containerisation setup is managed by multiple dockerfiles located in the root of the project as well as in the root directories of the frontend and backend services. All services are then orchestrated by Docker Compose, which coordinates the five components that make up the platform: Redis, the Django backend, the frontend, the Celery workers, and the PostgreSQL database.

The `docker-compose.yml` file defines and orchestrates all of these services. Part of this file is presented in Source code 7.1. Since some services depend on others being available before they can start, the `depends_on` directive is used to enforce the correct startup order and prevent crashes caused by missing dependencies. As this is a local distribution of the platform, users are not required to configure any environment variables or secrets. These are preconfigured within the `docker-compose.yml` file itself, keeping the setup as frictionless as possible. However, `SECRET_KEY` variable in the environment is recommended to set up explicitly.

Source code 7.1

Backend and Worker Service Configuration

```
1  backend:
2    build:
3      context: ./backend
4      dockerfile: Dockerfile
5    restart: unless-stopped
6    environment:
7      SECRET_KEY: local-docker-secret-key-not-for-production
```

```
 8      DEBUG: "True"
 9      DATABASE_URL: postgres://cleverminer:cleverminer@db:5432/
           cleverminer
10      REDIS_URL: redis://redis:6379/0
11    ports:
12      - "8000:8000"
13    depends_on:
14      db:
15        condition: service_healthy
16      redis:
17        condition: service_healthy
18    volumes:
19      - media_data:/app/media
20
21  worker:
22    build:
23      context: ./backend
24      dockerfile: Dockerfile
25    restart: unless-stopped
26    command: uv run python -m celery -A config worker --loglevel=
         info
27    environment:
28      SECRET_KEY: local-docker-secret-key-not-for-production
29      DEBUG: "True"
30      DATABASE_URL: postgres://cleverminer:cleverminer@db:5432/
           cleverminer
31      REDIS_URL: redis://redis:6379/0
32    depends_on:
33      db:
34        condition: service_healthy
35      redis:
36        condition: service_healthy
37    volumes:
38      - media_data:/app/media
```

To further improve the usability of the locally deployed application, user authentication is handled through a standard registration and login flow, requiring no third-party integration. This directly addresses the limitation encountered in Valá's implementation, which relied on Google Sign-In and therefore required additional configuration to function in a local environment.

7.3 Docker Set Up With Repository Code

Once the code is downloaded, the entire platform can be started with a single command presented in Source code 7.2.

Source code 7.2
Starting the Platform

```
1  docker compose up -d
```

Running this command causes Docker to pull the necessary images, such as PostgreSQL and Redis, and build and start all platform services as defined in `docker-compose.yml` from Source code 7.2. Once running, the active containers can be inspected either through the Docker Desktop application under the Containers tab, as shown in the Figure 7.1, or via the following terminal command in Source code 7.3.

Source code 7.3
Listing Running Containers

```
1  docker ps
```

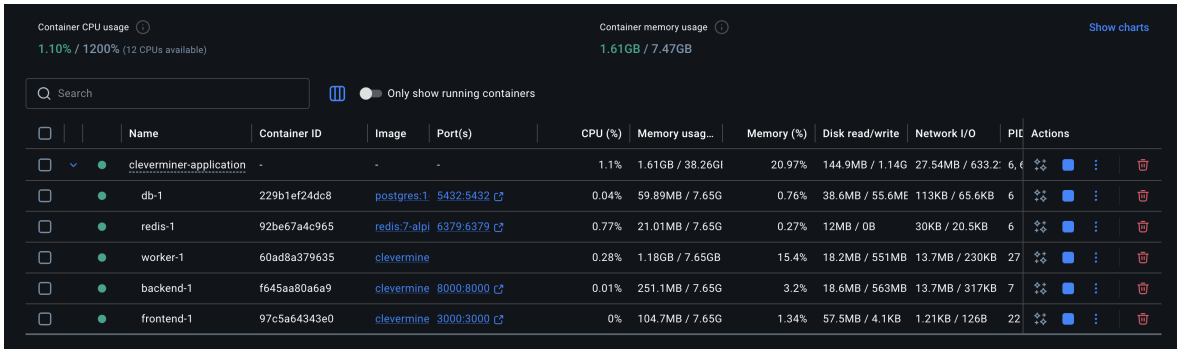
The platform can be shut down using the following command in Source code 7.4, optionally with the -v flag to also delete all persisted data.

Source code 7.4
Stopping the Platform

```
1  docker compose down
2  docker compose down -v
```

The frontend is accessible at localhost:3000, as visible in the Port(s) column in the figure.

Figure 7.1
Deployment - Docker Desktop



Note: Screenshot by Author from the Docker Desktop application.

7.4 Installation For End Users

To make installation as straightforward as possible for end users, the application images have been published to a public Docker Hub repository. With this setup, users only need to download the docker-compose.hub.yml file from the repository and run the following command Source code 7.5.

Source code 7.5

Starting the Platform From Docker Hub

```
1  docker compose -f docker-compose.hub.yml up -d
```

This command automatically pulls the pre-built images from Docker Hub and starts the application in containers, exactly as described in the previous subchapter, without requiring users to download the full source code or build the images locally.

7.5 Developer Local Setup

As this application is distributed under the MIT license, developers are free to build upon and extend it. Given that the platform relies on multiple services, the local development setup is described below for each component.

7.5.1 Frontend Setup

The frontend is started in development mode using the standard Node.js command Source code 7.6.

Source code 7.6

Starting the Frontend

```
1  npm run dev
```

The VITE_BACKEND_URL environment variable must be configured to point at the backend service.

7.5.2 Backend Setup

The backend is started with the following command Source code 7.7, which installs dependencies from `pyproject.toml` and `uv.lock` before launching the server.

Source code 7.7

Starting the Backend

```
1  uv run python manage.py runserver
```

The settings.py file contains all configurable settings. Notably, the ALLOWED_HOSTS array should include the frontend URL. By default, the application uses SQLite as its local database, however, if a `DATABASE_URL` environment variable is provided, the application automatically connects to it instead. S3 storage support can be enabled by uncommenting the relevant section in settings.py and populating the environment variables as specified in the `.env.example` file.

The Redis broker URL is configured as follows Source code 7.8, falling back to the default local address if no environment variable is set.

Source code 7.8
Celery broker URL Configuration

```
1  CELERY_BROKER_URL = os.getenv("REDIS_URL", "redis://localhost
       :6379/0")
```

7.5.3 Redis and Celery

On macOS, Redis can be managed through the Homebrew[1] package manager as is shown in Source code 7.9.

Source code 7.9
Managing Redis with Homebrew

```
1  brew services start redis
2  brew services stop redis
```

Celery is started and stopped with the following commands Source code 7.10.

Source code 7.10
Starting Celery Worker

```
1  celery -A config worker -l INFO
```

Celery automatically discovers tasks by scanning the applications listed in the Django backend INSTALLED_APPS array in `settings.py`. Additionally, the Flower monitoring interface can be started on port 5555 with the code below Source code 7.11.

Source code 7.11
Starting Flower Monitoring

```
1  celery -A config flower --port=5555
```

[1] https://brew.sh/

7.6 Deployment on Cloud Infrastructure

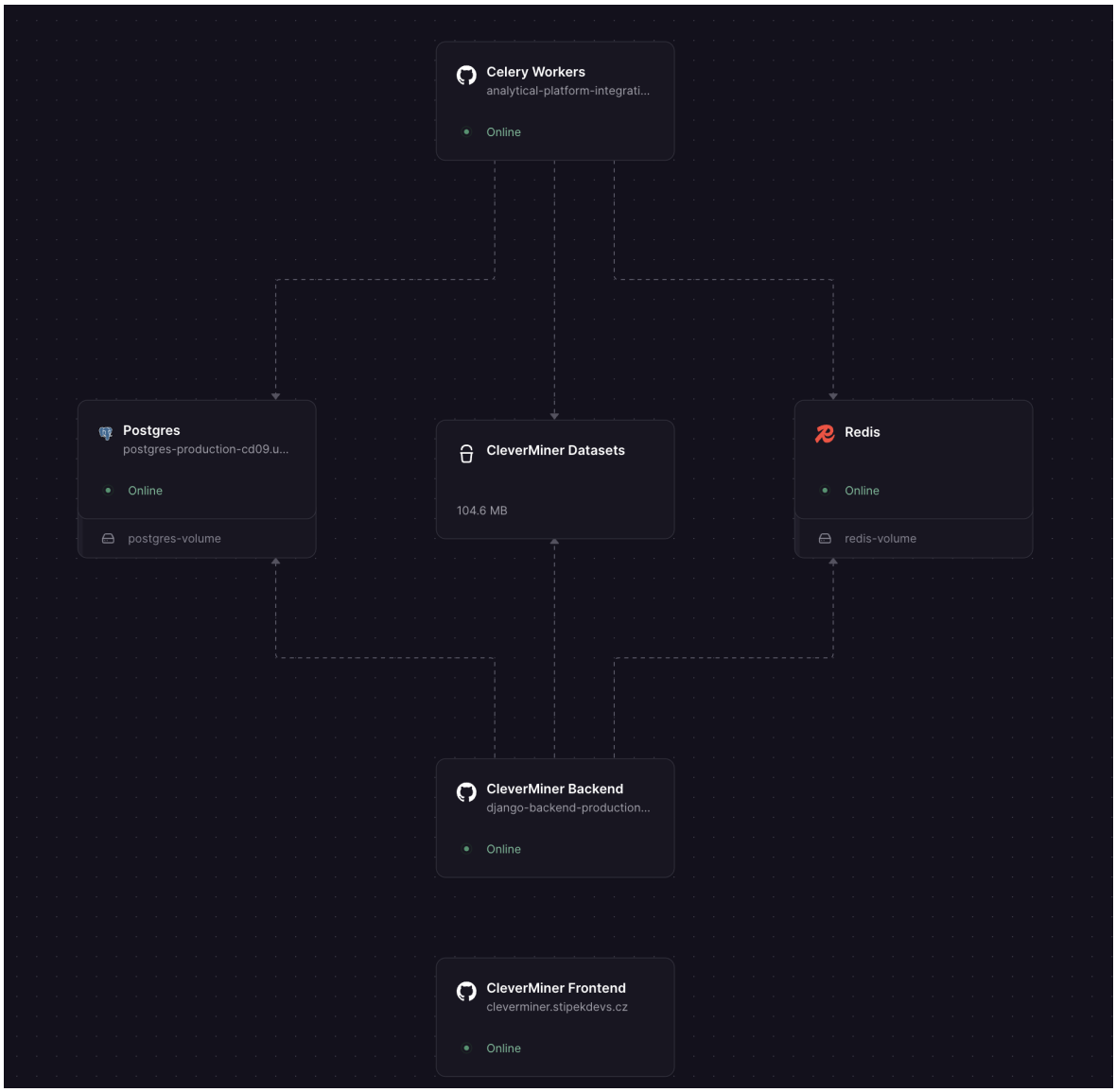
The platform can also be deployed to cloud infrastructure. Two main approaches exist: deploying using pre-built Docker images, or deploying each service individually and connecting them manually within the cloud provider. The latter approach is described in this chapter.

For the cloud deployment demonstration, the author uses Railway[2]. Railway is an accessible platform-as-a-service that supports multiple deployment configurations. The architecture of the deployed services on Railway is shown in the Figure 7.2. Compared to the local setup, the cloud deployment introduces an S3 bucket for dataset storage. The Celery workers and backend require the same environment variable configuration as in the local deployment. The Redis service provides a connection URL that must be additionally supplied to both the backend and Celery workers. The S3 bucket credentials can be conveniently injected into the relevant services directly through the Railway interface. As this is a publicly available deploy, the `SECRET_KEY` variable in the Django backend should be set up through the environment variables in Railway. The frontend requires only the `VITE_BACKEND_URL` variable to be set. It can be configured with a custom domain, with Railway handling the SSL certificate automatically. Users simply need to add a CNAME record in their domain registrar to point to the Railway-assigned domain, otherwise the application remains accessible under the automatically generated Railway subdomain.

Notably, Railway integrates with GitHub, allowing developers who fork the repository, that is available in Appendix A, to connect it directly to their Railway project. This enables automatic CI/CD pipelines whenever new code is pushed to the target branch, Railway automatically redeploys the affected service.

[2] https://railway.com/

Figure 7.2
Deployment - Railway Deployment



Note: Screenshot by Author from the Railway PaaS application.

8. Application testing and verification

The Application Testing and Verification chapter examines whether the platform fulfils the specifications and requirements established in previous chapters, and describes how the application was tested within the boundaries of the MMSP-AV methodology to implement the SO5 objective.

As Myers et al. (2011, p.1) state:

> Software testing is a process, or a series of processes, designed to make sure computer code does what it was designed to do and, conversely, that it does not do anything unintended.

The testing phase can also be considered the climax of the software development process, as it involves handing the developed application to individuals outside the development team and collecting feedback from those unaffected by confirmation bias (Yasar, 2022). Confirmation bias leads developers to favour testing the so-called happy path, the expected, error-free flow through the application, rather than actively attempting to break it. As a result, applications tested solely by their developers tend to be less robust than those evaluated by independent testers (Calikli et al., 2010).

The MMSP-AV methodology provides clear guidance on how to conduct efficient and effective testing of the implemented application. The testing procedure begins with the Test Plan document, which summarises all relevant information regarding the software testing process and clearly defines the boundaries within which testing will take place (Rejnková, 2011). The Test Plan specifies what will be tested and outlines the overall test strategy. It also describes how identified defects will be reported and tracked (Kemr, 2016a). The structure of the Test Plan presented in this thesis is derived from the `plan_testu_sablona` working product template defined by the MMSP-AV methodology.

8.1 General Approach to Testing

In this thesis, the testing approach is guided primarily by the software requirements defined in the earlier Chapter 4 and the user stories associated with those requirements. Both functional and non-functional requirements are subject to testing, with particular emphasis on performance and supportability.

The tests conducted based on this framework are user requirements tests, also known as acceptance tests, which will be carried out by external testers. Performance tests will be completed in accordance with the FURPS methodology and the non-functional requirements outlined in Subsection 4.3.2.

8.2 Application Testing

This subchapter follows the MMSP-AV methodology for testing applications and consists of the main outputs of the testing phase.

8.2.1 Test Plan

Testing will be conducted in accordance with the defined requirements and user stories, focusing on acceptance testing. In line with the FURPS methodology, particular emphasis is placed on performance and supportability requirements. Testers will use their own personal devices, and since the platform is primarily designed for desktop use, only desktop environments will be covered. The operating systems covered in tests are macOS, Windows 10, and Windows 11. Testing will be conducted asynchronously, testers will be provided with predefined test cases and asked to report their findings through a structured bug report table, included in the Appendix D.

Tests will be conducted on the available deployed application as well as on the local version installed by using Docker.

8.2.2 Test Cases

Test cases provide a structured guide for performing tests, presenting clear step-by-step instructions in which each user action corresponds to an expected system response. This approach enables systematic verification of the application's defined functionalities (Kemr, 2016a).

In this subchapter, the test cases for the platform are summarised in Table 8.1. The full MMSP-AV test case tables, defining each test case in detail, are provided in Appendix B. These test cases also support the defined user stories in Section 4.4. Four additional test cases are introduced to verify the platform's ability to correctly configure and execute each supported GUHA based procedure with defined quantifiers and parameters. These test cases are guided by the validated examples provided in the official CleverMiner documentation (Little Big Company & Máša, 2026), where each procedure is accompanied by a dataset, input configuration, and expected analysis output.

Table 8.1
Overview of Test Cases

ID	Test Case title
TC-01	User Registration and Authentication
TC-02	Uploading Datasets to the Application
TC-03	Configure Analysis Task
TC-04	Visualize Analysis Results
TC-05	Execution Flexibility and Repeated Analysis
TC-06	Dataset Preparation and Analytical Guidance
TC-07	Collaborative Analysis within Projects
TC-08	Performance Testing and Running Multiple Data Mining Operations
TC-09	Local Deployment
TC-10	4ft-Miner Procedure Verification
TC-11	SD4ft-Miner Procedure Verification
TC-12	CF-Miner Procedure Verification
TC-13	UIC-Miner Procedure Verification

8.2.3 Test Sets

Test sets group individual test cases into logical sequences that reflect the natural flow of user interaction with the application. Due to the scope constraints of this thesis, four test sets have been defined, each representing a distinct logical grouping within the overall test plan and are summarized in Table 8.2, full MMSP-AV test sets tables are present in Appendix C

Table 8.2
Overview of Test Sets

ID	Test Set Title
TS-01	Task Lifecycle
TS-02	Dataset Lifecycle
TS-03	Performance Analysis
TS-04	Procedures Verification

8.2.4 Acceptance Criteria

Based on the `zaznam_vysledku_testu_sablona` working product from the MMSP-AV, an error criteria table was created. This table categorises discovered defects into four severity groups. Following the approach of (Davidík, 2024), the author defines the acceptable number of errors permitted in each group for the application to be considered accepted and ready for deployment.

Table 8.3
Overview of Severity Categories and Acceptance Criteria

Severity	Description	Max. Defects
A	Critical mistakes that limit the usability of the application.	0
B	Non-critical mistakes that can be circumvented.	3
C	Small errors that don't have a direct consequence on the application's usage.	7
D	Errors where the user is unsure whether the behaviour is an error or not.	10

Source: (Rejnková, 2010, p.67-68), (Slavev, 2024)

8.2.5 Test Results

The application, per table Table 8.4, succesfully handled all the test cases that were prepared by the author and carried out by external testers. The table in Appendix D maps in detail all the found defects, that were submitted by the external testers to author.

Table Table 8.5 summarizes the number of found defects and the categories they belong to. This table is the second summary of the Appendix D table. Some of the reported defects were immediately fixed by the author.

Based on the results presented in Table 8.4 and in Table 8.5 author considers the application finished, as it passed prepared tests and implements all the requirements specified in the Chapter 4.

Table 8.4
Test Results

ID - test set	ID - test case	Test case title	Result
TS-01	TC01	User Registration and Authentication	OK
TS-01	TC02	Uploading Datasets to the Application	OK
TS-01	TC03	Configure Analysis Task	OK
TS-01	TC04	Visualize Analysis Results	OK
TS-01	TC05	Execution Flexibility and Repeated Analysis	OK
TS-02	TC02	Uploading Datasets to the Application	OK
TS-02	TC06	Dataset Preparation and Analytical Guidance	OK
TS-03	TC08	Performance Testing, Tunning Multiple Data Mining Operations	OK
-	TC07	Collaborative Analysis within Projects	OK
-	TC09	Local Deployment	OK
TS-04	TC10	4ft-Miner Procedure Verification	OK
TS-04	TC11	SD4ft-Miner Procedure Verification	OK
TS-04	TC12	CF-Miner Procedure Verification	OK
TS-04	TC13	UIC-Miner Procedure Verification	OK

Table 8.5
Count of Found Defects in the Application

Category	Number of found defects	Result
A	0	OK
B	2	OK
C	4	OK
D	6	OK

Source: author with (Slavev, 2024)

Conclusion

The main objective of this thesis was to design and implement analytical platform for the CleverMiner library, which will enable the involvement of domain experts in the data analysis process utilising the CleverMiner package. As the objective further states, the platform should provide a frontend interface for data inspection, definitions, execution, and management of mining tasks with particular emphasis on the SD4ft-Miner method, as well as visualisations of results while building upon existing implementations and extending them with new functionalities. This main objective was fulfilled through the collective outcomes of the supporting sub-goals addressed across the thesis chapters. The final application is available in Appendix A.

SO1 was fulfilled in the Analysis of Existing Solutions chapter, see Chapter 3, where four applications were evaluated to identify their benefits, limitations, and suitability as a foundation for this thesis. The evaluation concluded that none of the existing implementations were suitable to build upon, as each covered only one of the four GUHA based procedures supported by CleverMiner.

SO2 was fulfilled in the Theoretical Framework chapter, see Chapter 2, which analysed the CleverMiner library and its API with respect to integration possibilities. The chapter also introduced the domain of association rule mining and the GUHA method, establishing the theoretical foundation for the subsequent chapters.

SO3 was fulfilled across chapters Chapter 4 and Chapter 5 in accordance with the MMSP-AV methodology. Fifteen functional requirements and eight non-functional requirements were identified in Section 4.3, with particular emphasis on performance and reliability given the long-running, CPU-bound nature of the CleverMiner tasks. The requirements were complemented by five user stories, in Section 4.4, illustrating the application's intended workflows. The system specification concluded with a conceptual database schema in Section 4.6. The Architecture Design chapter, see Chapter 5, that followed, described the three-layer architecture supported by a Redis message broker and Celery workers, ensuring that CPU-bound mining tasks and data preprocessing do not degrade the responsiveness of the application and was concluded by the architecture design showcased in the Figure 5.2. The chapter also established the deployment strategy in Section 5.6, adopting Docker Compose as the orchestration solution for both local and cloud environments.

SO4 was fulfilled in the Application Development chapter, see Chapter 6, which presented the backend and frontend implementation using the Django and React frameworks respectively. The backend execution pipeline, the API layer, and the frontend module structure were described in detail, including the three-step task creation wizard and the results page presenting discovered rules with supporting statistics and visualisations.

SO5 was fulfilled in the Application Testing and Verification chapter, see Chapter 8, where

the application was evaluated by external testers through a defined set of acceptance test cases. The application successfully passed the acceptance criteria, with only minor defects identified and reported in Appendix D.

Importance and Benefits

The primary benefit of this work is the democratisation of the CleverMiner package, making it accessible to users without knowledge of Python or data analytics libraries. The implemented platform consolidates the entire data analysis pipeline into a single application, reducing the friction of working across multiple tools. By implementing the CRISP-DM framework, users can explore and preprocess their data, define and execute mining tasks, and iteratively refine their analyses within one environment. The platform can run entirely on a personal computer, making it accessible to academic users without any paid subscription or cloud resource constraints. As a broader consequence, increased accessibility of the CleverMiner package may contribute to greater adoption of GUHA based association rule mining in the data mining community.

Limits

The application is constrained by the web browser environment in which it operates. A native desktop application architecture would in some respects be better suited to the platform's requirements, however a native approach would limit cross-platform availability and require users to install the application locally, as opposed to accessing the publicly hosted web version directly from a browser.

Further Development

The platform offers several directions for future development. The data preprocessing capabilities could be extended with more advanced strategies beyond the currently supported discretisation and missing value imputation, and the CleverMiner Guidance heuristics could be improved to provide more sophisticated dataset recommendations.

The current implementation balances local and cloud deployment, which has constrained some features that would be beneficial exclusively in a hosted environment, such as single sign-on authentication or an email notification system. Future work could separate these concerns more explicitly, developing a more cloud-native version of the platform alongside the local distribution. Additionally, the platform could be extended to connect directly to external databases, removing the need for manual file uploads by users.

As the CleverMiner library itself is extended with additional GUHA based procedures, the platform is well positioned to incorporate them, gradually closing the gap with the LISp-Miner application, which implements the full range of generalised GUHA procedures.

Finally, to ensure the long-term financial sustainability of the publicly hosted version, a subscription or pay-per-use model could be introduced, where users pay only for the computational resources they consume. This would shift the hosting costs away from the deploying party and make the platform viable as a commercially sustainable service.

References

AbdulRahman. (October 15, 2025). *Why Written Software Requirements Are Essential for Successful Projects*. Medium. Retrieved January 22, 2026, from https://rahman-codes.medium.com/why-written-software-requirements-are-essential-for-successful-projects-e3acf3463a57

Agrawal, R., Imieliński, T., & Swami, A. (June 1, 1993). Mining association rules between sets of items in large databases. *SIGMOD Rec.*, *22*(2), 207–216. https://doi.org/10.1145/170036.170072

Agrawal, R., & Srikant, R. (September 12, 1994). Fast Algorithms for Mining Association Rules in Large Databases. *Proceedings of the 20th International Conference on Very Large Data Bases*, 487–499.

Ajitsaria, A. (N.d.). *What Is the Python Global Interpreter Lock (GIL)? – Real Python*. Retrieved February 3, 2026, from https://realpython.com/python-gil/

AltairEngineering. (N.d.). *Create Association Rules - Altair RapidMiner Documentation*. Retrieved February 15, 2026, from https://docs.rapidminer.com/2025.0/studio/operators/modeling/associations/create_association_rules.html

Apache. (N.d.). *Parquet*. Parquet. Retrieved February 2, 2026, from https://parquet.apache.org/

Axios. (N.d.). *Getting Started | Axios Docs*. Retrieved February 1, 2026, from https://axios-http.com/docs/intro

Behringer, M. (December 5, 2023). *Interactive, Explorative and User-Centric Data Analysis: Concepts, Systems, Evaluations*. BoD – Books on Demand.

Bhosale, P. (April 10, 2024). Parquet's Columnar Storage Advantage: A Case Study in Big Data Analytics. *IJSAT - International Journal on Science and Technology*, *15*(2). https://doi.org/10.5281/zenodo.14631461

Calikli, G., A. Uzundağ, B., & Basar, A. (September 19, 2010). Confirmation Bias in Software Development and Testing: An Analysis of the Effects of Company Size, Experience and Reasoning Skills.

Cohn, M. (N.d.). *User Stories and User Story Examples by Mike Cohn*. Mountain Goat Software. Retrieved January 23, 2026, from https://www.mountaingoatsoftware.com/agile/user-stories

Coolors. (N.d.). *Create a Palette - Coolors*. Coolors.co. Retrieved January 28, 2026, from https://coolors.co/generate

Creswell, J. W. (2009). *Research design: Qualitative, quantitative, and mixed methods approaches, 3rd ed.* Sage Publications, Inc.

Databricks. (October 30, 2018). *What is Apache Parquet?* Databricks. Retrieved February 2, 2026, from https://www.databricks.com/glossary/what-is-parquet

Davidík, D. (2024). *Mobilní aplikace pro podporu praxe mindfulness: Dechová a meditační cvičení* [Diplomová práce]. Vysoká škola ekonomická v Praze. Retrieved March 14, 2026, from https://theses.cz/id/vabyvm/?lang=cs

DeWitt & Stonebraker. (January 17, 2008). MapReduce: A major step backwards. https://dsf.berkeley.edu/cs286/papers/backwards-vertica2008.pdf

Django. (N.d.). *Django*. Django Project. Retrieved January 30, 2026, from https://www.djangoproject.com/

Docker. (N.d.-a). *Docker Compose*. Docker Documentation. Retrieved February 4, 2026, from https://docs.docker.com/compose/

Docker. (N.d.-b). *What is Docker?* Retrieved March 9, 2026, from https://docs.docker.com/get-started/docker-overview/

Eeles, P. (November 1, 2001). Capturing Architectural Requirements.

Encode. (N.d.). *Home - Django REST framework*. Retrieved February 2, 2026, from https://www.django-rest-framework.org/

Fabián, O. (2012). *Elektronické informační zdroje*. Retrieved February 14, 2026, from https://www.mlp.cz/katalog/titul/elektronicke-informacni-zdroje/4233707

Fayyad, U., Piatetsky-Shapiro, G., & Smyth, P. (March 15, 1996). From Data Mining to Knowledge Discovery in Databases. *AI Magazine*, *17*(3), 37–37. https://doi.org/10.1609/aimag.v17i3.1230

Feng, Y., Wu, Z., Zhou, X., Zhou, Z., & Fan, W. (November 1, 2006). Knowledge discovery in traditional Chinese medicine: State of the art and perspectives. *Artificial Intelligence in Medicine*, *38*(3), 219–236. https://doi.org/10.1016/j.artmed.2006.07.005

Fernandez, S. T., Tomas. (September 5, 2023). *Building Scalable Applications Using Redis as a Message Broker*. Semaphore. Retrieved February 3, 2026, from https://semaphore.io/blog/redis-message-broker

Figma. (N.d.). *What is Wireframing? The Complete Guide [Free Checklist]*. Figma. Retrieved January 28, 2026, from https://www.figma.com/resource-library/what-is-wireframing/

Foundatin, O. (N.d.). *Introduction | Electron*. Retrieved February 4, 2026, from https://electronjs.org/docs/latest/

Fowler, M. (N.d.). *Registry*. martinfowler.com. Retrieved March 3, 2026, from https://martinfowler.com/eaaCatalog/registry.html

Giacomelli, Š. (N.d.). *Django REST Framework Basics*. Retrieved January 30, 2026, from https://testdriven.io/blog/drf-basics/

Group, T. P. G. D. (N.d.). *PostgreSQL: About*. Retrieved February 2, 2026, from https://www.postgresql.org/about/

Hájek, P., Havel, I., & Chytil, M. (December 1, 1966). The GUHA method of automatic hypotheses determination. *Computing*, *1*(4), 293–308. https://doi.org/10.1007/BF02345483

Hájek, P., Holeňa, M., & Rauch, J. (February 1, 2010). The GUHA method and its meaning for data mining. *Journal of Computer and System Sciences*, *76*(1), 34–48. https://doi.org/10.1016/j.jcss.2009.05.004

Hatwalne, M. (November 28, 2025). *You Want Microservices—But Do You Need Them? | Docker*. Retrieved January 30, 2026, from https://www.docker.com/blog/do-you-really-need-microservices/

Hotz, N. (September 10, 2018). *What is CRISP DM?* Data Science PM. Retrieved November 20, 2025, from https://www.datascience-pm.com/crisp-dm-2/

Houtsma, M. A. W., & Swami, A. N. (March 6, 1995). Set-Oriented Mining for Association Rules in Relational Databases. *Proceedings of the Eleventh International Conference on Data Engineering*, 25–33.

Chapman, P. (2000). CRISP-DM 1.0: Step-by-step data mining guide. Retrieved February 11, 2026, from https://www.semanticscholar.org/paper/CRISP-DM-1.0%3A-Step-by-step-data-mining-guide-Chapman/54bad20bbc7938991bf34f86dde0babfbd2d5a72

Chlapek, D., Kučera, J., & Palovská, H. (2019). *Datové modelování a návrh relační databáze – Sbírka řešených úloh* (1st ed.). Retrieved January 25, 2026, from https://oeconomica.vse.cz/publikace/datove-modelovani-a-navrh-relacni-databaze-sbirka-resenych-uloh/

Chung, H. M., & Gray, P. (June 1, 1999). Special Section: Data Mining. *Journal of Management Information Systems*, *16*(1), 11–16. https://doi.org/10.1080/07421222.1999.11518231

Jackson, J. (March 22, 2002). Data Mining; A Conceptual Overview. *Communications of the Association for Information Systems*, *8*(1). https://doi.org/10.17705/1CAIS.00819

Karch, M. (January 22, 2021). *How to Do a Boolean Search on Google*. Lifewire. Retrieved February 14, 2026, from https://www.lifewire.com/boolean-search-terms-google-1616810

Kemr, J. (2016a). *Metodika MMSP-AV*. Retrieved November 11, 2025, from https://chi.vse.cz/mmsp-av/

Kemr, J. (2016b). *Návrh a implementace metodiky pro testování v agilních projektech* [Diplomová práce]. Prague University of Business and Economics. Retrieved April 28, 2026, from https://vskp.vse.cz/68661

Kollegger, E. (May 14, 2018). *What is Axios.js and why should I care?* Medium. Retrieved February 1, 2026, from https://medium.com/@MinimalGhost/what-is-axios-js-and-why-should-i-care-7eb72b111dc0

Kumar, K., Jain, A. K., Tiwari, R. G., Jain, N., Gautam, V., & Trivedi, N. K. (April 2023). Analysis of API Architecture: A Detailed Report. *2023 IEEE 12th International Conference on Communication Systems and Network Technologies (CSNT)*, 880–884. https://doi.org/10.1109/CSNT57126.2023.10134658

Kumar, R. (August 8, 2024). *Understanding Apache Parquet: A Detailed Guide*. Medium. Retrieved February 3, 2026, from https://medium.com/@reetesh043/understanding-apache-parquet-a-detailed-guide-5cd7e1b30e2e

Labs, T. (N.d.). *Tailwind CSS - Rapidly build modern websites without ever leaving your HTML*. Retrieved January 28, 2026, from https://tailwindcss.com/

Laplante, P. A., Laplante, P. A., Kassab, M., & Kassab, M. (June 7, 2022). *Requirements Engineering for Software and Systems* (4th ed.). Auerbach Publications. https://doi.org/10.1201/9781003129509

Little Big Company, s., & Máša, P. (2026). *ClemerMiner. Beyond apriori. Advanced ARM (association rule mining) ML engine.* Retrieved May 17, 2025, from https://www.cleverminer.org/

MetaPlatforms. (N.d.). *React.* Retrieved February 1, 2026, from https://react.dev/

Microsoft. (N.d.). *Common web application architectures - .NET.* Retrieved January 23, 2026, from https://learn.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/common-web-application-architectures

Myers, G. J., Sandler, C., & Badgett, T. (November 8, 2011). *The Art of Software Testing: | Guide books | ACM Digital Library.* Wiley Publishing. Retrieved March 14, 2026, from https://dl.acm.org/doi/book/10.5555/2161638

NumFOCUS. (N.d.). *Pandas - Python Data Analysis Library.* Retrieved January 30, 2026, from https://pandas.pydata.org/

Padilla-Rascón, M. A., González, P., & Carmona, C. J. (December 1, 2024). SDRDPy: An application to graphically visualize the knowledge obtained with supervised descriptive rule algorithms. *SoftwareX*, *28*, 101939. https://doi.org/10.1016/j.softx.2024.101939

Plego. (January 18, 2018). *Why User Interface Design is Crucial for Your Business.* Retrieved January 28, 2026, from https://plego.com/blog/importance-of-user-interface-design/

Pydantic. (N.d.). *Welcome to Pydantic - Pydantic Validation.* Retrieved January 30, 2026, from https://docs.pydantic.dev/latest/

Rane, N. L., Mallick, S. K., Kaya, Ö., & Rane, J. (October 13, 2024). *Explainable and trustworthy artificial intelligence, machine learning, and deep learning.* Deep Science Publishing. https://doi.org/10.70593/978-81-981271-4-3

Rauch, J. (2015). *LISp-Miner.* Retrieved February 15, 2026, from https://lispminer.github.io/wiki/lmbase/start.html

Rauch, J., Berka, P., & Šimůnek, M. (N.d.). LISp-Miner: Systém pro získávání znalostí z dat. Retrieved February 13, 2026, from https://www.academia.edu/53070972/LISp_Miner_syst%C3%A9m_pro_z%C3%ADsk%C3%A1v%C3%A1n%C3%AD_znalost%C3%AD_z_dat

ReactRouter. (N.d.). *React Router Home.* Retrieved February 1, 2026, from https://reactrouter.com/home

Read the Docs, I.

bibinitperiod c. (N.d.). *Popular documentation tools.* Read the Docs Documentation. Retrieved March 28, 2026, from https://docs.readthedocs.com/platform/stable/intro/doctools.html

Rejnková, P. (2010). *Lokalizace a přizpůsobení metodiky OpenUP* [Diplomová práce]. Vysoká škola ekonomická v Praze. Retrieved March 14, 2026, from https://theses.cz/id/s1ctdp/

Rejnková, P. (2011). *Metodika MMSP.* spicenter.vse.cz. Retrieved March 13, 2024, from https://spicenter.vse.cz/wp-content/uploads/metodiky/publikovana_MMSP/index.htm

Richardson, C. (N.d.). *What are microservices?* microservices.io. Retrieved January 30, 2026, from http://microservices.io/index.html

Salari, E. (October 4, 2024). *Understanding Association Rules: Uncovering Hidden Patterns*. Medium. Retrieved February 13, 2026, from https://medium.com/@ehsan.sepahsalari/understanding-association-rules-uncovering-hidden-patterns-8bd1a275b500

shadcn.io. (N.d.). *The AI-Native shadcn/ui Component Library for React*. shadcn.io. Retrieved January 28, 2026, from https://www.shadcn.io

Scholz, S. (August 2023). Towards a Democratization of Data Knowledge in Companies: The Concept of the Citizen Data Scientist. *ResearchGate*. https://doi.org/10.19080/ASM.2023.09.555757

Schröer, C., Kruse, F., & Gómez, J. M. (January 1, 2021). A Systematic Literature Review on Applying CRISP-DM Process Model. *Procedia Computer Science*, *181*, 526–534. https://doi.org/10.1016/j.procs.2021.01.199

Slavev, M. (2024). *Mobilní aplikace pro podporu praxe mindfulness: Indikátory stresu a angažovanost uživatelů* [Diplomová práce]. Vysoká škola ekonomická v Praze. Retrieved March 14, 2026, from https://theses.cz/id/g9tr3j/?isshlret=Michael%3BSlavev%3B;zpet=%2Fvyhledavani%2F%3Fsearch%3Dmichael%20slavev%26start%3D1

Soegaard, M. (January 21, 2026). *UI Color Palette 2026: Best Practices, Tips, and Tricks for Designers*. The Interaction Design Foundation. Retrieved January 28, 2026, from https://www.interaction-design.org/literature/article/ui-color-palette

Solem, A., & contributors. (N.d.). *Celery*. Retrieved February 3, 2026, from https://www.fullstackpython.com/celery.html

Stone, D., Jarrett, C., Woodroffe, M., & Minocha, S. (April 29, 2005). *User Interface Design and Evaluation*. Elsevier.

Studio, O. (N.d.). *Obra shadcn/ui*. Figma. Retrieved January 28, 2026, from https://www.figma.com/community/file/1514746685758799870/obra-shadcn-ui

Tan, P.-N., Kumar, V., & Srivastava, J. (July 23, 2002). Selecting the right interestingness measure for association patterns. *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 32–41. https://doi.org/10.1145/775047.775053

Tan, P.-N., Steinbach, M., & Kumar, V. (January 4, 2018). *Introduction to Data Mining* (2nd ed.). Pearson.

TanStack. (N.d.). *TanStack Query*. Retrieved February 2, 2026, from https://tanstack.com/query/latest

Team, T. P. (November 9, 2025). *Types of APIs: A Complete Guide to API Architectures*. Postman Blog. Retrieved February 3, 2026, from https://blog.postman.com/different-types-of-apis/

Tuychiev, B. (N.d.). *Pydantic: A Guide With Practical Examples*. Retrieved February 2, 2026, from https://www.datacamp.com/tutorial/pydantic

Vala, L. (June 2025). *Web frontend for CleverMiner*. Retrieved November 20, 2025, from https://vskp.vse.cz/98124_webovy-a-uzivatelsky-frontend-pro-clevermineer??page=72

Varshney, B. S. (2023). *Data Democratization Tops List of Data-Centric Trends for 2023*. TDWI. Retrieved November 18, 2025, from https://tdwi.org/Articles/2022/12/12/ADV-ALL-Data-Democratization-Top-Data-Centric-Trend-2023.aspx

VisualSiteMaps. (N.d.). *Why Create a Visual Sitemap? - VisualSitemaps*. Retrieved January 27, 2026, from https://visualsitemaps.com/why-create-a-visual-sitemap/

W3Schools. (N.d.-a). *Introduction to Django*. Retrieved January 30, 2026, from https://www.w3schools.com/django/django_intro.php

W3Schools. (N.d.-b). *Pandas Introduction*. Retrieved January 30, 2026, from https://www.w3schools.com/python/pandas/pandas_intro.asp

Wirth, R., & Hipp, J. (2000). CRISP-DM: Towards a Standard Process Model for Data Mining. https://www.cs.unibo.it/~danilo.montesi/CBD/Beatriz/10.1.1.198.5133.pdf

Yasar, K. (August 12, 2022). *What is Software Testing? Definition, Types and Importance*. WhatIs. Retrieved March 14, 2026, from https://www.techtarget.com/whatis/definition/software-testing

Zedníčková, M. (September 2025). *Web and user frontend for CleverMiner*. Retrieved November 20, 2025, from https://vskp.vse.cz/96360_webovy-frontend-pro-clevermi ner??page=64

Appendices

A. Application Code and Deployed Application

The code is available in public repository on GitHub service at:
https://github.com/petrstipek/Analytical-Platform-Integrating-CleverMiner

There is a Master Thesis Release with tag v1.0 identifying the state of the application in the time of submission of this thesis.

The app is available at: cleverminer.stipekdevs.cz

Documentation is available at: cleverminer-docs.stipekdevs.cz/docs/platform-introduction

The test credentials with testing data are in Table A.1.

Table A.1
Application Test Credentials

Account email	**Password**
`tester01@vse.cz`	fasbam-9hAchy-xybrap

B. Test Cases

ID	TC01
Title	User Registration and Authentication
Description	Testing that user can create new account and then log in with it.
prerequisites	Application has to be running.
Notes	Refers to FR-01

	Test data	Tester Action	System Reaction	Result	Notes
1.		User goes to the registration page.	System presents the registration page.	OK	
2.	tester10x @example.com Password123	User fills in the test data into inputs and clicks register.	System shows notification and redirects to login page.	OK	
3.	tester01	User fills in email and password and logs in.	System will redirect to home page and shows notification.	OK	
			Result	OK	

ID	TC02
Title	Uploading Datasets to the application
Description	Testing whether user is able to upload dataset and then do initial exploration.
prerequisites	Downloaded dataset from test data. User is logged in.
Notes	Implements FR-02

	Test data	Tester Action	System Reaction	Result	Notes
1.		User goes to the Dataset page.	System presents Datasets page and loads available datasets.	OK	
2.		User clicks on the Upload New Dataset button.	User is redirected to new page with dataset upload form.	OK	
3.	Dataset provided to testers.	User types in the name of the dataset, specifies delimiter and uses the upload area to select the dataset.	-	OK	
4.		User clicks on the upload dataset button.	Systems starts loading the dataset.	OK	
5.		-	Systems redirects to the detail of the dataset with loaded profile.	OK	
			Result	OK	

ID	TC03
Title	Configure Analysis Task
Description	Testing that user can create new data mining task and start it.
prerequisites	Uploaded and prepared dataset.
Notes	Refers to US-01. This case represents multiple steps, user should follow the official wiki to create the task.

	Test data	Tester Action	System Reaction	Result	Notes
1.		User goes to the tasks page.	System presents the tasks page.	OK	
2.		User clicks new task button.	Systems show the first wizard step.	OK	
3.	Wiki for SD4ft-Miner task creation.	User fills in all the data and selects attributes inputs quantifiers.	System accepts all the inputs.	OK	
4.		In the last step, user clicks on the button to save the task.	Systems saves the task and redirects to the task detail page.	OK	
5.		User clicks on the button to run the task.	System returns notification of the action status and updates the runs for the task table to contain the newly created run.	OK	
			Result	OK	

ID	TC04
Title	Visualize analysis results
Description	User explores the result of the run executed from a task.
prerequisites	Successffuly finished task with found rules.
Notes	Refers to US-02.

	Test data	Tester Action	System Reaction	Result	Notes
1.		User goes to the tasks page.	System presents the tasks page.	OK	
2.		User clicks on the last task that was created.	System shows the detail page of a task with current configuration and runs table.	OK	
3.		User clicks on the last run, that was executed for the task.	Systems shows the run detail page with rules list, configuration collapsible element and detailed statistics about selected rule.	OK	
4.		User can select rules.	System updates the details based on the selected rule.	OK	
			Result	OK	

ID	TC05
Title	Execution flexibility and Repeated analysis
Description	User can change task configuration and run multiple analysis.
prerequisites	Created task. User is already on task detail page.
Notes	Refers to US-05.

	Test data	Tester Action	System Reaction	Result	Notes
1.		User selects edit configuration	System redirects to page with second step of the configuration wizard.	OK	
2.		User updates the task configuration, by adding random attribute.	System allows for updating the attribute set.	OK	
3.		User continues to the next step.	System presents the quantifiers step of the wizard	OK	
4.		User updates the quantifiers to higher values.	System allows for the changes and form is validated.	OK	
5.		User selects save.	System updates the task and user is redirected.	OK	
			Result	OK	

ID	TC06
Title	Dataset preparation and analytical guidance
Description	Testing that user can do data preparation and get analytical guidance.
prerequisites	There is a dataset uploaded to the application.
Notes	Refers to US-03

	Test data	Tester Action	System Reaction	Result	Notes
1.		User goes to the datasets page.	System presents the datasets page.	OK	
2.		User selects the Accidents dataset and clicks on it from the second table	System shows the dataset detail page on the first exploratory data analysis tab.	OK	
3.		User selects the CleverMiner Guidance.	System shows guidance on how to prepare the attributes in the dataset.	OK	
4.		User selects the data preparation tab.	System shows the dataset preparation tab with all attributes in a list.	OK	
4.		User selects Road attribute.	System opens a drawer component from the right side of the screen with all available attribute preparations.	OK	
5.		User selects to impute null values with mean, and to categorize attribute by quantiles.	System adds data preparation steps to preparation batch visible on top of all the attributes.	OK	
6.		User runs the transformations.	System runs the operation and returns notification of the action	OK	
			Result	OK	

ID	TC07
Title	Collaborative analysis within projects
Description	Testing that user can collaborate on projects with other users.
prerequisites	There are multiple users registered in the application.
Notes	Refers to US-04

	Test data	Tester Action	System Reaction	Result	Notes
1.		User goes to the projects page.	System shows the current overview of all projects.	OK	
2.		User selects Create New Project button.	System shows form for creating new projects.	OK	
3.		User inputs project name and creates project.	System creates the new project and redirects to it.	OK	
4.	tester02 @exam- ple.com	User invites new user to the app.	System adds new user (tester02) to the project.	OK	
			Result	OK	

ID	TC08
Title	Performance testing, running multiple data mining operations on the server.
Description	Testing that platform can handle multiple requests for data mining.
prerequisites	User is logged in and there is already defined task in the app.
Notes	Refers to NFR-04, NFR-06

	Test data	Tester Action	System Reaction	Result	Notes
1.	Task 01	User goes to the task detail page.	System shows the task detail page.	OK	
2.		User starts multiple runs from the task detail page.	Systems queues multiple runs in the same task and executes them.	OK	
3.		User reviews the runs table in the task and verifies that all runs are completed and the app is still responsive.	-	OK	
			Result	OK	

ID	TC09
Title	Local deployment
Description	Testing that user can use the application locally.
prerequisites	User has donwloaded Docker on their device and docker-compose.hub.yml file from project repository.
Notes	Refers to NFR-08

	Test data	Tester Action	System Reaction	Result	Notes
1.		User uses this command: `docker compose -f docker-compose.hub.yml up -d`.	Docker starts downloading images and setting up container.	OK	
2.		User goes to localhost:3000, where the app runs	Application is loaded and ready to be used.	OK	
			Result	OK	

Table B.1
TC10 – 4ft-Miner Test Data

Parameter	**Value**
Dataset	Accidents
Procedure	4ft-Miner
Antecedent attributes	Driver_Age_Band (seq, 1–3), Driver_IMD (seq, 1–3), Sex (subset, 1–1), Journey (subset, 1–1)
Succedent attributes	Hit_Objects_in (rcut, 1–11), Hit_Objects_off (rcut, 1–11), Casualties (rcut, 1–6), Severity (lcut, 1–2)
Cedent length	Antecedent (con, 1–4), Succedent (con, 1–1)
Quantifiers	Base: 2000, aad: 0.7
Expected result	27 rules found

ID	TC10
Title	4ft-Miner Procedure Verification
Description	Testing that the platform correctly executes the 4ft-Miner procedure with parameters defined in the CleverMiner documentation and returns the expected rules.
Prerequisites	Accidents dataset uploaded and preprocessed. User is logged in.
Notes	Refers to FR-05, FR-06, FR-08. Test data based on (Little Big Company & Máša, 2026).

	Test data	Tester Action	System Reaction	Result	Notes
1.		User creates a new task with the 4ft-Miner procedure and selects the Accidents dataset.	System presents the task configuration wizard.	OK	
2.	See Table B.1	User configures the antecedent and succedent attributes per the test data table.	System accepts all attribute inputs.	OK	
3.	See Table B.1	User sets quantifiers and saves the task.	System saves the task and redirects to task detail page.	OK	
4.		User runs the task.	System queues and executes the run.	OK	
5.	See Table B.1	User navigates to the run detail page and reviews the results.	System displays 27 rules with statistics and visualisations.	OK	Rule count matches CM docs.
			Result	OK	

Table B.2

TC11 - SD4ft-Miner Test Data

Parameter	**Value**
Dataset	Accidents
Procedure	SD4ft-Miner
Antecedent attributes	Vehicle_Type (subset, 1–1), Speed_limit (seq, 1–2)
Succedent attributes	Severity (lcut, 1–2)
First set attributes	Driver_Age_Band (seq, 1–3)
Second set attributes	Driver_Age_Band (seq, 1–3)
Cedent length	Antecedent (con, 1–4), Succedent (con, 1–1), Set1 (con, 1–1), Set2 (con, 1–1)
Quantifiers	Base1: 4000, Base2: 4000, Ratiopim: 1.4
Expected result	19 rules found

ID	TC11
Title	SD4ft-Miner Procedure Verification
Description	Testing that the platform correctly executes the SD4ft-Miner procedure with parameters defined in the CleverMiner documentation and returns the expected rules.
Prerequisites	Accidents dataset uploaded and preprocessed. User is logged in.
Notes	Refers to FR-05, FR-06, FR-08. Test data based on (Little Big Company & Máša, 2026).

	Test data	Tester Action	System Reaction	Result	Notes
1.		User creates a new task with the SD4ft-Miner procedure and selects the Accidents dataset.	System presents the task configuration wizard.	OK	
2.	See Table B.2	User configures all cedents including first and second set attributes per the test data table.	System accepts all attribute inputs.	OK	
3.	See Table B.2	User sets quantifiers and saves the task.	System saves the task and redirects to task detail page.	OK	
4.		User runs the task.	System queues and executes the run.	OK	
5.	See Table B.2	User navigates to the run detail page and reviews the results.	System displays 19 rules with two four-fold tables and statistics.	OK	Rule count matches CM docs.
			Result	OK	

Table B.3
TC12 - CF-Miner Test Data

Parameter	**Value**
Dataset	Accidents
Procedure	CF-Miner
Target variable	Severity
Condition attributes	Driver_Age_Band (seq, 1–3), Driver_IMD (seq, 1–3), Sex (subset, 1–1), Journey (subset, 1–1), Speed_limit (seq, 1–3), Light (subset, 1–1), Vehicle_Type (subset, 1–1)
Cedent length	Condition (con, 1–2)
Quantifiers	S_Down: 1, Base: 100
Expected result	9 rules found

ID	TC12
Title	CF-Miner Procedure Verification
Description	Testing that the platform correctly executes the CF Miner procedure with parameters defined in the CleverMiner documentation and returns the expected rules.
Prerequisites	Accidents dataset uploaded and preprocessed. User is logged in.
Notes	Refers to FR-05, FR-06, FR-08. Test data based on (Little Big Company & Máša, 2026).

	Test data	Tester Action	System Reaction	Result	Notes
1.		User creates a new task with the CF-Miner procedure, sets Severity as the target variable, and selects the Accidents dataset.	System presents the task configuration wizard.	OK	
2.	See Table B.3	User configures the condition cedent attributes and cedent length per the test data table.	System accepts all attribute inputs.	OK	
3.	See Table B.3	User sets quantifiers and saves the task.	System saves the task and redirects to task detail page.	OK	
4.		User runs the task.	System queues and executes the run.	OK	
5.	See Table B.3	User navigates to the run detail page and reviews the results.	System displays 9 rules with histogram visualisations.	OK	Rule count matches CM docs.
			Result	OK	

Table B.4
TC13 - UIC-Miner Test Data

Parameter	**Value**
Dataset	Accidents
Procedure	UIC-Miner
Target variable	Severity
Antecedent attributes	Driver_Age_Band (seq, 1–3), Driver_IMD (seq, 1–3), Sex (subset, 1–1), Area (subset, 1–2), Journey (subset, 1–2), Speed_limit (seq, 1–2), Light (subset, 1–2), Vehicle_Type (subset, 1–1)
Cedent length	1–2
Quantifiers	aad_score: 20, aad_weights: [5, 1, 0], Base: 200
Expected result	Rules with aad_score above 20 returned

ID	TC13
Title	UIC-Miner Procedure Verification
Description	Testing that the platform correctly executes the UIC Miner procedure with parameters defined in the CleverMiner documentation and returns the expected rules.
Prerequisites	Accidents dataset uploaded and preprocessed. User is logged in.
Notes	Refers to FR-05, FR-06, FR-08. Test data based on (Little Big Company & Máša, 2026).

	Test data	Tester Action	System Reaction	Result	Notes
1.		User creates a new task with the UIC-Miner procedure, sets Severity as the target variable, and selects the Accidents dataset.	System presents the task configuration wizard.	OK	
2.	See Table B.4	User configures the antecedent attributes and cedent length per the test data table.	System accepts all attribute inputs.	OK	
3.	See Table B.4	User sets quantifiers including weights and saves the task.	System saves the task and redirects to task detail page.	OK	
4.		User runs the task.	System queues and executes the run.	OK	
5.	See Table B.4	User navigates to the run detail page and reviews the results.	System displays discovered rules with histogram visualisations showing category improvements.	OK	Results consistent with CM docs.
			Result	OK	

C. Test Sets

Test Set ID	TS-1
Test set title	Task lifecycle
Description	This test set tests the applications ability to create and execute task and then view the results.
Prerequisities	Uploaded dataset in the app, logged in user.
Test Cases	TC01 - TC05
Result	OK

Test Set ID	TS-2
Test set title	Dataset Lifecycle
Description	This test set tests, whether user has the ability to upload dataset and then gain knowledge about the dataset through the application.
Prerequisities	Logged in user.
Test Cases	TC02, TC06
Result	OK

Test Set ID	TS-3
Test set title	Performance analysis
Description	This test set tests, whether the application is able to handle multiple running tasks and still be responsive to normal user requests.
Prerequisities	Logged in user. Created tasks and uploaded datasets.
Test Cases	TC08
Result	OK

Test Set ID	TS-04
Test Set Title	Procedures Verification
Description	This test set verifies that the platform correctly configures and executes each of the four supported GUHA based procedures with defined parameters and returns results consistent with the official CleverMiner documentation.
Prerequisites	User is logged in. Accidents dataset is uploaded and preprocessed in the application.
Test Cases	TC10, TC11, TC12, TC13
Result	OK

D. Overview of Identified Defects

Defect ID	Defect Title	Description	Severity	Author's statement
DF-01	Login form clears on window switch	Content of the login form is lost when the user switches to another window.	C	Acknowledged. Will be addressed in a future release.
DF-02	Discretization button label unclear	The close button on the discretization drawer does not indicate that the transformation will be queued and requires Apply All to execute.	D	Fixed, note is currenlty provided.
DF-03	No navigation to settings for attribute value limit	Attributes report too many values, but there is no link or guidance directing the user to the relevant settings.	C	Acknowledged. A direct link to settings will be added.
DF-04	Task wizard skips succedent step	After filling in the antecedent, clicking Next Step skips the succedent and navigates directly to quantifiers.	B	Fixed, new level of wizard navigation was added.
DF-05	No feedback for incomplete attribute	When a third attribute is added but left unfilled, the Next Step button is blocked without any visible error or indication.	C	Fixed.
DF-06	Unclear quantifier requirements	It is not clear which quantifier fields are required to proceed.	D	Acknowledged. Required fields will be marked explicitly.

Defect ID	**Defect Title**	**Description**	**Severity**	**Author's statement**
DF-07	Task not visible from project tab	A task assigned to a project does not appear when browsing from the project tab.	B	Fixed, there was a missing insert into DB.
DF-08	404 error on run detail	After navigating into a task run, clicking on the run results in a 404 error. Breadcrumb navigation also changes unexpectedly.	D	Leftover run from author testing, not a bug.
DF-09	Task displays internal ID instead of name	The task breadcrumb and navigation show the internal system ID rather than the user-defined task name.	D	Acknowledged.
DF-10	New project displays wrong label	The new project form shows "Project Name Task Name" instead of just the project name.	C	Acknowledged. The label will be corrected.
DF-11	Pre-processed dataset fails to load	Pre-processed datasets cannot be opened, only newly uploaded datasets load correctly.	D	Leftover from author testing, not a bug.
DF-12	CSV delimiter selection not functional	The delimiter cannot be selected during CSV upload, causing the entire dataset to load as a single column.	D	Wrong version of the application was used, the correct has a delimiter input field.

E. Use of Artificial Intelligence

E.1 Improvement of Grammar and Language Style

Artificial Intelligence was used to improve grammar and word order throughout this thesis, as it is written in the author's second language. The usage consisted of asking the AI to suggest better phrasing or verify the correct sentence structure of written paragraphs.

E.2 Generation of Parts of Source Code

E.2.1 Overall Use

Artificial Intelligence was used primarily for consultation on implementation approaches and for generating basic boilerplate code, such as simple atom components. No agentic AI tools such as Claude Code or Cursor were used during the development of this thesis.
Example usage consists of author creating a component/function/class and then asking the Claude Sonnet 4.6 model on how could this be improved or whether different approach should be taken. The author then based on the response adjusted the source code.
Another example usage is architecture consultation on how to structure the apps and modules in backend and frontend applications.

E.3 Other Use

E.3.1 Generation of Documentation Text

The author used this thesis as source material when generating the platform documentation in Docusaurus. Portions of the thesis text, together with relevant screenshots and task specifications, were provided as input to the Claude Sonnet 4.6 model with a request to generate Markdown files. The resulting output was reviewed, adjusted where necessary, and integrated into the documentation source code.