SCA Tool Admin Business Intelligence App

MASTER THESIS

Maximilian Krug

Submitted on 30 June 2026

Friedrich-Alexander-Universität Erlangen-Nürnberg
Faculty of Engineering, Department Computer Science
Professorship for Open Source Software

Supervisor:
Martin Wagner M.Sc
Prof. Dr. Dirk Riehle, M.B.A.

Declaration of Originality

I confirm that the submitted thesis is original work and was written by me without further assistance. Appropriate credit has been given where reference has been made to the work of others. The thesis was not examined before, nor has it been published. The submitted electronic version of the thesis matches the printed version.

Erlangen, 30 June 2026

License

Erlangen, 30 June 2026

Abstract

SCA Tool is a university-led project that focuses on license compliance and Software Bill of Materials (SBOM) management. The system lacks the capability to analyze usage data and product-related metrics. Such an analysis is needed for data-driven decision-making for the continuous project and product development.

This thesis addresses this issue and implements the Business Intelligence (BI) system for SCA Tool. The system utilizes Extract, Transform, Load (ETL) pipelines; these extract the data from the production environment. Then transforms the data into time-series metrics before loading the metrics into the data warehouse. Finally, the data is visualized using charts and dashboards.

The system was built using an agile iterative approach to development and is ready to be integrated into the main deployment of SCA Tool. This system is capable of calculating 23 business metrics and implements the end-to-end process from production data to visual dashboards. The current system is limited by the fact that it is not integrated into the main production system. Therefore, no metrics were calculated on the production data, and no performance analysis is conducted.

Contents

List of Figures

List of Tables

Acronyms

API Application Programming Interface

AWS Amazon Web Services

BI Business Intelligence

CSV Comma-Separated Values

ELT Extract, Load, Transform

ERP Enterprise Resource Planning

ETL Extract, Transform, Load

gRPC gRPC Remote Procedure Calls

GUI Graphical User Interface

IDE Integrated Development Environment

JVM Java Virtual Machine

K8s Kubernetes

MCDM Multi-Criteria Decision Making

NoSQL Not only SQL

OLAP Online Analytical Processing

OLTP Online Transaction Processing

OSS Open Source Software

RBAC Role-Based Access Control

REST Representational State Transfer

RLS Row-Level Security

SBOM Software Bill of Materials

SQL Structured Query Language

WSM Weighted Sum Model

UUID Universally Unique Identifier

1 Introduction

1.1 SCA Tool

'Software is not an asset; software is a liability' is a core principle in modern software development. Today, this includes more than just the source code. It also includes licenses for the software components used, which are provided by other entities and vulnerabilities.

SCA Tool is a university project run by the Open Source Software (OSS) chair of computer science at the FAU Erlangen-Nuremberg. SCA Tool is tackling these arising challenges by providing users with license compliance, SBOM management, vulnerability scanning, and open source governance [1].

1.2 Rationale: To Measure is to Know

The truism 'To measure is to know' accredited to Lord Kelvin, says that measuring is the basis for all decision-making. The current production system of SCA Tool has no measuring or reporting system implemented to gather business and customer behavior. In practice no statistics are being collected on what features users are using and to what degree users are interacting with the current functionalities. This includes the usage of the newly created features such as license curation, as well as overall statistics such as user number over time. Furthermore, SCA Tool currently has no central data-collection system for business intelligence data and visualization system. This leads to decision-making that is not data-driven.

1.3 Objectives for the Thesis

The working result of this thesis is the BI system for SCA Tool. The system contains three subsystems. Firstly, it includes building a storage solution for

[1] https://scatool.com/

the BI data, which centrally stores data from all the operational data sources. Secondly, a data transformation tool is integrated to transform the data. Thirdly, a BI visualization tool is deployed to assess the newly gathered information. This thesis implements these three tools and their functionalities for SCA Tool.

1.4 Research question

This section covers the metrics that are being analyzed in this work in order to improve the decision-making in SCA Tool. The metrics are divided into four groups. User-based metrics cover the user creation and user behavior within SCA Tool. Package-based metrics are the second group. Packages are the dependencies of the code repositories from the users. Project-based metrics cover the projects users can create to upload their code repositories. The projects contain multiple versions of the same code repository. Organization-based metrics are the last group. For easier maintenance and access management, users are grouped in organizations. These metrics are aggregated and visualized by the BI system.

The following paragraph lists the metrics gathered for every group. The list explains the main purpose of all the gathered metrics. In order to get more insights into the user count and feature usage of users.

1. User

Users Over Time Tracks the total number of active or registered users historically.

Notifications per User Measures the volume of system alerts received by individual users. Tracking the maximum, average, and median helps assess user engagement and potential alert fatigue.

Uploads per User Quantifies the number of files uploaded by the user for the curation.

Curation Activities per User Counts the license refinement actions performed by users. Tracking the high, average, and median indicates the level of community or individual contribution to data accuracy.

File Completions per User Evaluates license file completions by tracking the high, average, and median number of files successfully processed by each user.

2. Package

Packages by License Counts the number of packages associated with each declared license. This is critical for monitoring legal compliance and open source exposure.

Package Scan Frequency Tracks how many times each specific package (ignoring version variations) was scanned on a given date to understand user and technological demand.

Most Scanned Packages Identifies the packages that undergo the most frequent scans, detailing the total count and average rates over time.

Most Failed Scans Highlights packages that experience the highest rate of scan failures. This metric is crucial for identifying corrupted, misconfigured, or highly problematic packages.

Most Successful Scans Highlights packages with the highest frequency of successful scans, indicating stable, well-formed, and reliable packages.

3. Projects

Projects Over Time Measures the cumulative count of projects tracked chronologically to gauge overall system utilization.

Version Units Over Time Measures the growth of version units over time, reflecting the volume of specific project iterations being managed.

Total Projects per Day A daily aggregate count of projects to monitor daily platform activity and growth.

Total Version Units per Day A daily aggregate count of version units processed, indicating day-to-day system load.

Vulnerabilities per Version Unit Evaluates security health by calculating the maximum, average, and median number of vulnerabilities found within version units per day.

Components per Version Unit Assesses software complexity by tracking the maximum, average, and median number of components per version unit per day.

Issues per Project Tracks the maximum, average, and median number of issues logged per project to gauge project health and maintenance burden per day.

Total Issues per Day A daily aggregate count of all issues raised across the system to monitor overall platform stability and error rates.

4. Organizations

Organizations Over Time Tracks the retention and growth of distinct organizations using the platform over time.

Curated Packages per Organization Counts the number of packages that have been actively curated by a specific organization, reflecting their engagement level.

Users per Organization Measures organizational scale and platform penetration by calculating the maximum, average, and median number of users tied to each organization per day.

Scanner Jobs per Organization Quantifies the compute workload generated by each organization by tracking the maximum, average, and median number of scanning jobs initiated per day.

Application Programming Interface (API) Keys per Organization Evaluates the scale of automation and external integrations by counting the maximum, average, and median number of API keys provisioned per organization per day.

The decision-making system is based on the numeric results of these metrics. These metrics are calculated and presented in the BI system.

1.5 Outline

Chapter 2 lays out the literature review, which provides the theoretical foundation for the BI solution, it also covers the software selection process as well as the descriptions for each of the software tools used in the system or evaluated in the selection process. Chapter 3 lays out the component selection process and reasons for the selection of Apache Superset as the BI visualization tool. Chapter 4 covers the requirements for the system. The chapter is split up into the starting requirements and the final requirements. Chapter 5, addresses the overall architecture of the system. This includes the BI visualization, ETL and Databases needed. Chapter 6 goes into further detail and elaborates the implementation of the complete system. The next chapter, Chapter 7 is Evaluation. In the evaluation the system and methodology are critically assessed. The final chapter, Chapter 8, is the conclusion.

2 Technical Background and Related Work

The chapter focuses on the scientific foundations of this work. In the first two subsections, the definition and theoretical groundwork BI and BI system architecture are covered. The scientific research on the topic of software component selection is also provided. Further subsections cover the tools used to build the BI system for SCA Tool. The final section contains the related work of similar scientific papers that built BI systems using a similar tech stack.

2.1 BI Basics

This section addresses the initial question: 'What is business intelligence?' and the relevance of business intelligence to start-ups.

BI is defined as the process of business users analyzing and monitoring the data of a business in order to turn the data into actionable knowledge. Analysis covers several tasks, such as data querying and data mining. The monitoring includes data warehouses, visualizations, and dashboards. Therefore, BI can be seen as a decision support tool. This is done with the primary goal of improving the companies' economic returns and stability (Molensky et al., 2010). In the context of BI, the term 'business' is not limited to the economic endeavors of companies but is meant in a broader context of purposeful activities, for example, in the context of research, law, and broader academia. Similarly, the term "intelligence" is more used as an umbrella term for knowledge (Luhn, 1958).

Therefore, a business intelligence system is defined as a system to carry out the different activities, such as monitoring and analyzing, in order to assist the company in the decision-making process. The BI system bundles the aforementioned aspects into one system provided to business users (Luhn, 1958).

In the context of SCA Tool, a fully digital university project, BI is of importance because start-up-like projects move in a high-paced environment. (Huang et al., 2022) find that BI improves the innovation power and network learning of

start-ups, both of which improve the financial returns of the start-up. More importantly, BI systems help start-ups to act quicker on market changes and accelerate the generation of business projections (Azeroual & Theel, 2018). In addition, BI systems can help start-ups to improve the following aspects: sort and unify company data, enhance collaboration, make processes more transparent, and improve exploratory feature development (Sharma et al., 2024).

In this work BI is defined as the process of consolidating data from operational functions in order to gain a deeper understanding of the business and the user engagement, in order to improve SCA Tool based on data driven decisions.

2.2 The BI architecture

This section elaborates further on the different aspects of BI systems, with a larger focus on the aspects implemented in this work. The main architecture is discussed in Chapter 5. Overall there are five steps, which are part of the architecture of BI systems and the BI stack. These are defined by (Azeroual & Theel, 2018) as:

1. Data Collection
2. Data Integration
3. Data Storage
4. Data Processing
5. Data Presentation

The first step, Data Collection, describes collecting the data from the production environment. Data sources for this step are internal and external from the production environment. They are not limited to Structured Query Language (SQL) databases but also include Not only SQL (NoSQL) databases, data warehouses, and data marts.

The second step, Data Integration, combines the data collected from the different operational data sources. During data integration additional processing to unify data for further storage, using methods such as ETL or Extract, Load, Transform (ELT).

The third step is Data Storage during which the business data is stored persistently. Storage solutions vary from SQL to NoSQL databases, data warehouses to data marts. One key aspect of data storage is the decoupling from the production database.

The fourth step is Data Processing, the persistent data from the data storage step is the basis for the execution. This transformation is an analytical layer

to analyze and mine the data and prepare it for the next step. Compared to the data integration step, the data is processed for the final visualization step to gather analytical insights from the data instead of unifying data.

The fifth step is Data Presentation. This step visualizes the business data to the business user via dashboards, charts, and representations.

The following subsection explains the key functions of the BI system architecture in more detail. These are the ETL system that is used to perform the data integration. The data warehouse that is used for data storage and the visualization tool, which is used to implement the data presentation. These functions are the main subject of the architecture of the BI system built for SCA Tool.

2.2.1 ETL-Tool

There are two main data transformation paradigms: ETL and ELT, which perform BI data integration. ETL extracts the data from the data source first, then transforms it before loading the data. ELT switches the order of operation. It first extracts, then loads the data into the data solution and performs the transformations on the data after storage. Overall, the advantages of ETL are that data is structured before being stored in the data warehouse. Through ETL the performance of the data warehouse is improved because the data is cleaned before the loading. Therefore, the data is not transformed in the data retrieval from the data warehouse. ELT is best suited for real-time and fast transactions of big data, as its main advantage is reduced latency and less computational overhead before loading (Machado et al., 2019). SCA Tool uses daily data extractions of highly structured operational data. This higly structured data and the clear separation of concerns between data transformation and loading are the reasons for the usage of ETL in the Data Integration step of the BI architecture stack.

During the extraction phase, the organizational data from multiple sources is gathered. These sources can be external and internal. The organizational data includes not only structured data from databases but also data from Representational State Transfer (REST) API calls or Comma-Separated Values (CSV) files. In the transformation phase, the data is unified and structured for later storage. The processing also includes quality checks and data augmentation. The final phase loads the data into the storage solution. Data repositories range from databases to data warehouses (Bansal, 2014). Additionally, the transformation step includes data cleaning, combining, and calculations as well as encoding and decoding (Pereira & Costa, 2016). The purpose of the ETL process is to prepare and restructure the data for decision-making (Mahmud & Ikbal, 2022). Concretely, the data is optimized for the data presentation step (Pereira & Costa, 2016).

2.2.2 Data Warehousing

(Han et al., 2012) define a data warehouse as a repository, storing data decoupled from the operational data with the goal of supporting multiple systems for decision-making support. There are two main data processing environments: Online Analytical Processing (OLAP) and Online Transaction Processing (OLTP) (Conn, 2005). The difference is that OLAP is used for analytical processes to support decision-making systems such as BI systems, whereas OLTP is used to support operational data (Conn, 2005). There are four characteristics of data warehouses: these are Subject oriented, Integrated, Non Volatile and Time Variant (Ramadhani et al., 2021). In detail, Subject oriented means the data is centered around subjects instead of operational processes. This is also a characteristic of OLAP systems (Conn, 2005). Subjects can be user usage metrics or suppliers. Integrated describes that the data is unified into one system and storage solution. This is also in contrast to the operational data, which contains multiple sources and modalities. Storing the business data decoupled from the operational data is a requirement for an OLAP system (Conn, 2005). Non volatility means data entering the storage solution is not subject to changes in the form of updates or deletion. Time variance is the characteristic of a data warehouse to add chronological information, for example, time stamps to data upon storage. This is necessary since some operational data might not contain time stamps. The advantage of adding timestamps is to retrieve information regarding trends, like the scan frequency of specific packages in a timeframe of five days (Ramadhani et al., 2021). Trends like these can be shown in the data presentation step by using data visualizations.

2.2.3 Data Visualization

The role of visualization is to provide the user with a more actionable and concise representation of the data. The main usage of the data visualization tool is to implement the data presentation step of the BI architecture. This representation can range from dashboards, charts, and graphs to abstract images. The visualization is based on the stored data. Visualization is done to reduce the time in the decision-making process (Eberhard, 2023).

2.3 Component Selection

In order to choose a visualization tool for the BI system for SCA Tool a methodological framework on how to choose third-party software components is needed. This section covers the theoretical base for software component selection used in this work. The standard process of component selection consists of five steps (Becker et al., 2013):

1. Define criteria
2. Search for components
3. Shortlist candidates
4. Evaluate candidates
5. Analyze results and choose component

In more detail, the first step of defining the criteria is the selection of criteria that are used to rate the software components. These are used in step two, searching for components. This step aims to search for suitable candidates broadly. The third step reduces the inital list of suitable software components into a short list of the most promising ones. Step four is the detailed analysis of the short list of candidates that are evaluated using the initially defined criteria. The difference between step two and four is the scope of the analysis. In step two a subset of critera is used to create a selection of the available components. In comparison, step four analyzes the components to get a conclusive final rating. Step five is the final step of comparing the results of the analysis and deciding on a final component. Here the component with the best result is chosen (Mohamed et al., 2007).

There are several methods of analyzing and calculating the result based around Multi-Criteria Decision Making (MCDM), where different mathematical bases can be used (Zaidan et al., 2015). The focus is laid on the WSM system within MCDM. WSM is a known and widely used method in MCDM, it is also the simplest and easiest (Zaidan et al., 2015).

$$A_i = \sum_{j=0}^{n} w_j x_{ij}, \quad \text{for } i \in \{1, \ldots, m\}$$

Figure 2.1: WSM formula for multi-criteria component evaluation

WSM works by assigning each criterion a weight. The WSM formula is shown in Figure 2.1. It assigns each possible component a value for each criterion. These two values are then multiplied, resulting in a rating for each component and criterion. In order to get the final result for the comparison, these values are summed up for each criterion (Soares & Brito, 2023). The selected component is the one with the highest overall sum of all the values (Zaidan et al., 2015). For the ratings of the individual values for each component, a five-point Likert scale is used, as (Papp & Hanussek, 2020) found that it was precise in granularity without compromising the distinctness of the results.

2.4 Tools

This section focuses on the tools used and evaluated in this work. The first subsection covers the BI visualization tools. In order to provide the information needed for Chapter 3 on software component selection method is presented. The second subsection focuses on the other tools used to build the system. These include the software handing the ETL process, data storage, data processing and visualization.

2.4.1 BI Visualization Tools

The following section covers the three visaulization tools selected for the final analysis in the software component selection process. These tools are Redash, Metabase, and Superset. These tools are used in the data presentation step of the architecture because they provide visualization of data.

Redash Redash, is an OSS solution to visualize data in dashboards. It was created in 2014. The software called Redash was created by the company Redash. After stagnation the software was reinitiated by the original founders in April 2023 as a community-led project[1]. Today it is licensed under the BSD 2-clause. The project had 70 contributions between Q1 and Q3 of 2025[2], with a total of 28k stars and 4.5k forks on GitHub[3]. However, the project was stagnant over a range of two years, from 2021 to 2023, before it was reinitiated.

From a technical perspective, Redash is built using Python and deployed with Docker images, these can also be used with Kubernetes. Redash provides Helm charts for a simpler deployment[4]. It creates visualizations in the frontend using TypeScript. The implementation in Python suggests a slower performance than comparable solutions. From reviewing the documentation, some quick start guides are very detailed, whereas searching for more specific topics is harder.

The visualization is done with dashboards that contain charts. Redash supports a variety of charts out of the box, including boxplot, line charts, bar charts, and 12 more types of charts [5]. One advantage is that new charts can be easily added since Redash has the capability to add its own charts using Angular[6]; however, the overall capability to write and use community-created plugins is missing. Charts and dashboards can be easily shared as exported images or as weblinks using the

[1]https://github.com/getredash/redash/discussions/5962
[2]https://github.com/getredash/redash/pulse
[3]https://github.com/getredash/redash/graphs/contributors
[4]https://redash.io/help/open-source/setup/#docker
[5]https://redash.io/help/user-guide/visualizations/visualization-types/
[6]https://discuss.redash.io/t/how-to-create-new-visualization-types-in-redash/86

internal API and web services[7]. Individual charts can be easily embedded into external websites using iframes[8]. In terms of data sources, the system provides drivers to connect to PostgreSQL, MySQL, Google Sheets[9], and cloud-driven data sources such as Elasticsearch and Amazon Web Services (AWS)[10]. Since Redash is using Redis to read data from SQL databases, the real-time capabilities are slower due to the computational overhead.

Metabase Metabase is a freemium BI visualization tool that is provided by the company Metabase. Freemium in this context means that the OSS version of Metabase is released under the AGPL-3 license. However, this version has a reduced feature set compared to the premium commercial version[11]. Metabase as a commercial entity collects telemetry data for Metabase inlcuing the OSS version[12]. The software was first published on GitHub in 2015 and has an active community. There were over 500 commits in November of 2025 [13]. Overall the Metabase GitHub repository has 44k stars and 6k forks[14].

From a technical perspective, Metabase is built using Clojure (a Lisp dialect) and TypeScript for the front end. Metabase does not provide a data storage or caching service, which makes the deployment easier[15]. The absence of caching makes longer reoccurring searches slower and, therefore, reduces performance[16]. Metabase has built-in support for real-time data analysis[17]. Metabase is deployed using Docker[18]. In contrast to the official Docker image release, there are no Kubernetes Helm charts released by Metabase[19]. Metabase provides documentation for the cloud and Kubernetes capabilities of the solution. However, the documentation lacks the deployment guide.

The visualization of Metabase is done using dashboards and charts. These charts use a variety of over 20 different supported graphs[20], in addition of the 20 graphs, users can create custom charts and visualizations as Metabase supports plugin

[7]https://redash.io/help/user-guide/integrations-and-api/api/
[8]https://redash.io/help/user-guide/dashboards/sharing-dashboards/
[9]https://redash.io/integrations/
[10]https://redash.io/integrations/
[11]https://www.metabase.com/license/
[12]https://www.metabase.com/docs/latest/installation-and-operation/information-collection
[13]https://github.com/metabase/metabase/graphs/contributors
[14]https://github.com/metabase/metabase/graphs/contributors
[15]https://www.metabase.com/docs/latest/installation-and-operation/running-metabase-on-docker
[16]https://www.metabase.com/learn/metabase-basics/administration/administration-and-operation/metabase-and-your-db
[17]https://www.metabase.com/dashboards/real-time-analytics
[18]https://www.metabase.com/docs/latest/installation-and-operation/running-metabase-on-docker
[19]https://medium.com/@10anupriya/deploying-metabase-on-kubernetes-02d65d07837d
[20]https://www.metabase.com/docs/latest/questions/visualizations/visualizing-results

creation natively. [21]. These dashboards can easily be created through a self-service menu that abstracts the underlying SQL query from the user, which makes the usage easier for non-technical users[22]. The system also supports SQL editing to enable more technical users to interact with the underlying data directly[23]. Metabase also supports the creation of embeddings, making embedding content into external websites easy and granular[24]. Metabase lacks user-level access management for rows and table data in the open source version[25]. Multiple data sources, including PostgreSQL and other SQL databases[26], and also Google Sheets, can be integrated. Due to the support for plugins, community-developed drivers for data sources exist or can be created[27].

Superset Superset is a BI visualization tool and part of the Apache Foundation[28]. It is released under the Apache-2.0 license. It was first hosted on GitHub in 2015 and has an active community with over 237 contributions in November 2025[29]. The repository also has over 69k stars and 16k forks on GitHub[30].

On the technical side, Superset is built using Java and a TypeScript frontend. Superset is deployed with Docker images, either via Docker Compose[31] or Kubernetes[32]. The website has sufficient documentation on the deployment, with both Docker Compose and Kubernetes. Apache Superset is deployed alongside Redis and a PostgreSQL database. This reduces performance since these additional tools are needed for the deployment[33]. Similar to Metabase, Superset supports real-time data analysis[34].

The visualization is done via datasets that are extracted from data sources. These datasets are used for charts, which are then displayed in dashboards[35]. Over 60 types of charts are included in Superset by default [36], with users being able to write and create new charts since Superset supports community-driven plugins[37]. The system currently supports data sources including PostgreSQL and Google

[21] https://www.metabase.com/data-sources/?use_case=bi
[22] https://www.metabase.com/docs/latest/questions/query-builder/editor
[23] https://www.metabase.com/docs/latest/questions/native-editor/writing-sql
[24] https://www.metabase.com/docs/latest/embedding/introductions
[25] https://www.metabase.com/pricing/?use_case=bi
[26] https://www.metabase.com/data-sources/?use_case=bi
[27] https://www.metabase.com/data-sources/?use_case=bi
[28] https://projects.apache.org/project.html?superset
[29] https://github.com/apache/superset/graphs/contributors
[30] https://github.com/apache/superset
[31] https://superset.apache.org/docs/6.0.0/installation/docker-compose
[32] https://superset.apache.org/docs/6.0.0/installation/kubernetes
[33] https://superset.apache.org/docs/6.0.0/configuration/configuring-superset
[34] https://preset.io/blog/2021-4-26-real-time-analytics-in-superset/
[35] https://superset.apache.org/docs/using-superset/creating-your-first-dashboard
[36] https://superset.datatest.ch/chart/add
[37] https://superset.apache.org/docs/contributing/howtos#creating-visualization-plugins

Sheets[38]. The dashboard can be shared with external websites via the export as iframe or the Superset API[39]. One additional feature of Superset is SQL Lab, which enables more technical users to use an advanced SQL editor to write custom SQL queries. The normal editor is no-code and therefore more accessible to non-technical users. The no-code editor also creates charts and dashboards.

2.4.2 Concrete ETL-Tool

This section covers the ETL-Tool usage in this work, as it is a backbone of modern BI tools (Mahmud & Ikbal, 2022). The usage of ETL instead of ELT was argued in Section 2.2. This section explains the choosing of Apache Hop as ETL tool for the BI system, which is needed in the data integration step of the BI architecture.

Apache Hop Apache Hop, is a modern data orchestration tool. It is part of the Apache Software Foundation and licensed under Apache-2.0 license. Hop has 1.4k stars on GitHub and 453 forks [40] as the time of writing. The repository had 51 contributions in November of 2025 [41]. Apache Hop is not only a data orchestration but also a data engineering tool. It has a Graphical User Interface (GUI) and, therefore, is visualization-based. Users create workflows consisting of pipelines. These connect data sources to extract data, then transform the data and finally load the data into a data source. Currently Hop supports not only SQL-based data sources, but also NoSQL, MongoDB, Cassandra and more Apache Databases. The developed pipelines can also run on Apache Beam or Apache Spark and are therefore not limited to Hop. Further use cases of Apache Hop are parallel processing of large datasets, including data cleansing and data migration between databases. Hop is built on the Hop engine and also provides the user with the ability to create or use plugins [42]. The GUI-based development and the features such as parallel pipeline execution are the reason why Apache Hop is chosen as the ETL tool for the BI system.

2.4.3 Additional Software Components

This section covers the additional software used for building, deploying, and maintaining the BI system. It includes the database for the data warehouse and the migration tool for database consistency, as well as the deployment tools.

[38]https://superset.apache.org/docs/configuration/databases#supported-databases-and-dependencies
[39]https://superset.apache.org/docs/6.0.0/api
[40]https://github.com/apache/hop/
[41]https://github.com/apache/hop/graphs/commit-activity
[42]https://hop.apache.org/manual/latest/getting-started/hop-what-is-hop.html

PostgreSQL

One major aspect of the BI system is the data repository needed for the data storage step in the BI architecture. The database engine used for the data warehouse and OLAP environment is PostgreSQL. PostgreSQL is an open source object-relational database management system. It is released under the PostgreSQL license, which is similar to the MIT license [43]. It builds upon SQL standard and has the following functionalities: fault tolerance, security, data parity, disaster recovery, concurrency, and extensibility[44]. One additional reason for using PostgreSQL for the operational database is that the usage of the same database engine for the production and BI data warehouse is usable if the relational database supports the SQL3 standard (Conn, 2005). PostgreSQL supports the object extension of SQL3 [45]. These advantages make PostgreSQL a better choice for BI systems than other SQL databases like MySQL or data warehouses like OracleDB (Conrad, 2021). Section 2.5 lists more examples of BI systems being built with PostgreSQL as the data repository solution.

Flyway

Flyway is a database migration tool used alongside databases like PostgreSQL. It is used to migrate the schemas of databases. Flyway allows deterministic updates of the database to a new state. Furthermore, it saves the current state of the migration in a separate table in the database to ensure clear versioning of the database. Both of these arguments combined allow Flyway to fully recreate a database. It works by sequentially applying database altering SQL migration scripts [46].

Docker

Docker is an open source container engine used to run software containers. Software containers run applications in an isolated environment, using the Docker engine. Docker also provides Docker Compose, which is used to host, start, and manage multiple Docker containers in one file [47]. In addition, Docker and, therefore, Docker Compose provide volume and networking capabilities to the containers, allowing the containers to save data persistently and connect to networks and other containers using the built-in networks stack [48]. Concisely, Docker is a container engine to deploy and run containerized applications.

[43] https://www.postgresql.org/about/license
[44] https://www.postgresql.org/about/
[45] https://www.postgresql.org/docs/current/features.html
[46] https://documentation.red-gate.com/flyway/getting-started-with-flyway/why-database-migrations
[47] https://www.docker.com/products/docker-desktop/
[48] https://docs.docker.com/engine/network/drivers/bridge/

Kubernetes

Similar to Docker, Kubernetes is also an open source container orchestration tool. It is used to deploy containers across multiple servers. In addition to container management, Kubernetes has automatic deployment built-in in order to scale containers and services up and down. Kubernetes also has these features: load balancing, storage orchestration, secret and configuration management, and self-healing. Self-healing describes the capability to restart a container that failed automatically in order to keep the predetermined numbers of containers running. [49].

2.5 Related Work

This section focuses on scientific work, building a similar system to the one in this work. The section focuses on comparing the technology stack and BI architecture of this work with industry- and academia-built systems. Technically similar systems provide theoretical and foundational evidence for the feasibility and suitability of the chosen BI architecture. The basic architecture of the BI system for SCA Tool is using Apache Hop as the ETL provider, PostgreSQL as the data storage, and Apache Superset as the BI visualization tool. A deeper explanation of the architecture is provided in Chapter 5. The four sources are: (Villafanes Lozano, 2026), (Gücük et al., 2023), (Anggrainy & Sari, 2022), and (Chumpitaz-Huarcaya & Auccahuasi, 2026).

The closest architecture in comparison is Lozano's (Villafanes Lozano, 2026) bachelor's thesis from January, 2026. The system is designed to extract data from their university portal to assess the academic performance of six different universities, using information such as the enrollment data, in order to aid decision-making. The system is built using open source off-the-shelf solutions. It uses Apache Hop as the ETL provider. As a storage solution, a PostgreSQL database is used. The visualization tool used is Apache Superset. Its core tech stack is identical to this work. There are four main differences between (Villafanes Lozano, 2026) work and the BI system for SCA Tool. Firstly, the system serves a different use case. This work is part of a university project with a start-up like environment compared to the other work's academic supporting role. Secondly, due to the difference in use case, the system for SCA Tool is deployed using both Docker Compose and Kubernetes for use in a cluster, which is needed for a customer-facing service. The reference system is only deployed using Docker Compose and only used locally. Thirdly, the data source for the academic work is CSV files from the universities, compared to the PostgreSQL-based databases used in SCA Tool. Fourthly, the BI system uses additional tooling such as Fly-

[49]https://kubernetes.io/

way for the data schema migration, which is not done in the work of (Villafanes Lozano, 2026).

Another system within the educational domain was built by (Gücük et al., 2023). The system is designed to provide a real-time capable project management tool for the universities' course management and the students' progress within the courses. The tech stack chosen is built upon the Jira API. The Jira API sends data to a dedicated Flask server, which sends the data into a Kafka message queue. These messages are processed by Apache Hop and then stored in a PostgreSQL database. Finally, as the visualization tool Metabase is used to access the stored data and generate a visual representation. This is similar to this systems approach in the usage of Apache Hop as ETL tool and PostgreSQL as the storage solution. Comparatively, Apache Superset is chosen in this work as the visualization tool and Apache Hop is directly used to access the source database without an additional message broker and web server.

The work from (Anggrainy & Sari, 2022) built a BI system to analyze the international sales data of the Olist Store. The Olist Store is an e-business selling Enterprise Resource Planning (ERP) and software solutions [50]. The system uses the following tech stack to build the BI system. It uses Pentaho as an ETL system to consolidate data from the production database. In addition to the ETL system, Python scripts are used to cleanse the data. The data is then stored in a PostgreSQL database. For the visualization Tableau is used. The system is similar to this works BI system in the use of the same processes and database provider for the data warehouse. Key differences are that Pentaho is used for the ETL instead of Apache Hop, as well as the usage of Tableau instead of Superset. Similar to other works in this section, this work does not focus on the deployment in the cloud.

A similar tech stack with a different use case is the work of (Chumpitaz-Huarcaya & Auccahuasi, 2026). The work focuses on centralizing data from different data sources in a medical setting. The problem the work describes is that different medical systems generate different reports without a central data repository to gain actionable information from. In order to build this system, all source databases are connected to Apache Hop, which is used for ETL data extraction. The data is then loaded into a centralized PostgreSQL database. The data collection, data integration, and data storage steps are similar to this work, the main difference is the lack of visualization and the economic focus of the system.

These works from academia, economics, and medical domains show that the chosen architecture is applied in a broad range of fields. This shows that the system design chosen for this work is viable and feasible.

[50] https://olist.com/o-que-e-olist/

3 Selection Process

This chapter is about the software component selection process for SCA Tool. It is the basis for the design and implementation of the BI system. The question arose whether an open source solution exists that supports the visualization needed in the data representation step.

3.1 Selection process

In Chapter 2 the WSM software component selection process was described. This model is used in the process covered in this chapter.

1. Define criteria
2. Search for components
3. Shortlist candidates
4. Evaluate candidates
5. Analyze results and choose component

3.2 Defining criteria

In the first step, the definition of the criteria, the following criteria were selected. This list is the basis for the component selection. These are listed in descending order of priority, starting with the most important and ending with the least important. The following list contains the name of the criterion, the given weight, and a short description. The weights and criteria were decided together with the core SCA Tool development team in a meeting.

OSS Business Strategy: 0.12 The OSS business strategy describes the business model behind the entity of the software provider. In the context of open source, these can be foundations, companies, or loosely affiliated groups.

OSS Maturity: 0.12 OSS maturity is the maturity of the project. It includes how old the project is and how active the maintainers and the development is.

Current Data Compatibility: 0.10 The next criterion, current data compatibility, describes the support of the system for the current database setup. In the case of SCA Tool the support for PostgreSQL databases is needed. Due to the usage of the Google Suite for internal communication, support for Google Sheets is not necessary for the operational data but preferred.

Docker Kubernetes Support: 0.10 This point is about system deployment. The current production system is deployed using Docker for local development and Kubernetes, or Kubernetes (K8s), for the cluster.

Visualization Variety: 0.09 Visualization variety is the amount of charts and the visualization options and capabilities the system provides by default.

Documentation Quality: 0.08 Documentation quality is the overall impression and quality of the documentation. This includes the user and technical documentation. Since the system is used by business and technical users.

Performance: 0.07 Performance is a criterion regarding the system's performance in the deployed production environment. SCA Tool is currently hosted on a on-premise server, which makes it difficult to get more compute power, especially as the resources between production and BI system are shared.

API and Web Service: 0.07 The API and web service capabilities describe the ability of the system to provide the visualization in different ways, either via an API to embedded dashboards in a different website or directly hosted on a web server provided by the visualization tool. The more options the BI tool provides, the more flexible the system is.

Self Service: 0.06 Self service is the extent to which the tool allows users with and without technical skills to create and build visualizations including access management.

Granular Embeddings: 0.06 Similarly to the API service, the point of granular embeddings is how granular content can be embedded into an external service. For example, the embedding of the dashboard into the admin page.

Real-time Capability: 0.05 Real-time capabilities point out the ability and speed of the system to process real-time data.

Future Proof: 0.03 Future proof is an abstract criterion, which means how likely the system will support potential changes made in the production system of SCA Tool, such as the migration to a different database.

Ready Made Dashboards: 0.03 Ready-made dashboards describe the system's premade plug-and-play ability to take data and generate dashboards without configuration.

Plugin Support: 0.02 The final criterion is plugin support, which describes the systems support for community-made plugins and add-ons to the visualization.

3.3 Search for components

In order to find a component that fits the criteria several methods are used to search for candidate components that provide visualization to BI systems. Some of the search results include Tableau, Power BI, Grafana, Redash, Superset, and Metabase. In order to search for these components, a broad internet search was conducted. This search included querying Google Scholar and GitHub for repositories under the topics of BI and visualization.

3.4 Shortlist candidates

The list of components is narrowed down for the detailed comparison. In order to generate a shortlist of the components, focus was laid on two of the aforementioned criteria. These are open source business strategy and open source maturity. Since these are the highest-weighted criteria, they will have the biggest impact on the final score, which makes them a good initial metric to reduce the list.

From the list of potential candidates in the last step Redash, Metabase, and Superset were chosen. Since they are the ones that have the most stars on GitHub, which is an indicator of the maturity and usage of the software components. They were chosen because of their license and distribution model as all of them are open source.

In contrast, Tableau and Power BI are commercial, non open source solutions, and Grafana is a visualization tool for observability and system monitoring, not BI.

3.5 Evaluate candidates

The following evaluation was conducted from the shortlist of candidates which was selected in the previous steps. The short list of candidates consists of these three software components: Redash, Metabase, and Superset.

In the following paragraphs, each component gets a number on the Likert scale attributed. The points range from one to five, where a higher number indicates a better match to the criterion. The basis for the attribution is the description of the BI visualization tools from Chapter 2. The table contains the name of the criterion, the rating as a number, and a short explanation for the rating. Each ranking is attributed in the following way: A ranking of one means the criterion is not supported. Whereas a ranking of two means the feature is indirectly implemented. The ranking of three shows that the feature is present; however, it lacks some necessary details. In contrast, a ranking of four means the feature is present and fulfills the selected criteria completely. Five points are given when the criteria are overfilled.

Redash

OSS Business Strategy: 3 Redash was originally created by the commercial entity named Redash and then re-initiated by the same team. This indicates less long-term stability.

OSS Maturity: 3 The project had a nearly two-year phase of stagnation between 2021 and 2023 and fewer contributions than the other components.

Current Data Compatibility: 5 Redash supports PostgreSQL as well as Google Sheets, which completely covers our current data sources.

Docker K8s Support: 5 Docker images and Helm charts are provided by Redash for local and server deployments. This is completely covers the deployment system of SCA Tool.

Visualization Variety: 5 Redash supports approximately 14 visualization graphs, ranging from bar to box graphs. This provides a high utility for our visualization system.

Documentation Quality: 3 The documentation covers the most common settings and questions. It is missing some precise specifications and information in more detailed subjects.

Performance: 3 Compared to Java Virtual Machine (JVM) built alternatives, the reliance on Redis and Python for the backend lead to a reduced performance.

API and Web Service: 4 Redash provides several options to access the visualizations. The web server that is deployed alongside Redash provides not only a website for visualization but also an API endpoint.

Self Service: 4 Redash provides capabilities for technical and non-technical users to create dashboards and visualizations, enabling more users to benefit from the BI system.

Granular Embeddings: 4 The RESTAPI allows to embed dashboards into external websites, providing more flexibility, using iFrames.

Realtime Capability: 3 Due to the lack of data buffering or caching, Redash is limited in real-time capabilities, especially for large data inputs.

Future Proof: 2 The temporary stagnant and slower development compared to other solutions of Redash and the lack of community-created data source drivers make Redash less future proof.

Ready Made Dashboards: 1 There are no ready-made dashboards available.

Plugin Support: 1 Redash does not provide plugin support for community created plugins.

Metabase

OSS Business Strategy: 3 Metabase is a freemium software solution, which means it is licensed by a commercial entity. This entity can change the license in future versions or stop development. However, this risk is mitigated through the active development of the open source variant.

OSS Maturity: 5 Metabase is in active development, with constant community contributions and a large framework of maintainers.

Current Data Compatibility: 5 Similar to Redash, it supports both PostgreSQL and Google Sheets, completely covering all current data sources used in SCA Tool.

Docker K8s Support: 3 Metabase provides premade Docker images for the deployment. Metabase, however, does not provide Helm charts, which enable an easier deployment on the SCA Tool server infrastructure.

Visualization Variety: 5 Metabase supports over 20 different visualization graphs, which allows for a high flexibility in data visualization.

Documentation Quality: 4 The user manual is very descriptive, covering all aspects of the user experience. The technical documentation is sufficient, but missing some smaler details.

Performance: 4 Metabase is written in in Lisp, a JVM based language which makes it performant, compared to other solutions.

API and Web Service: 4 Similar to Redash, the deployment of Metabase contains a web server, which provides the visualizations using websites and API endpoints.

Self Service: 3 Metabase makes the creation of dashboards easy for non-technical users, with the point-and-click builder system. It also supports SQL quer-

ies for more technical users. It is missing row-level access capabilities and making all data available to all users, which is a security risk.

Granular Embeddings: 4 Metabase provides granular embeddings for external websites, via the RESTAPI.

Realtime Capability: 3 Metabase supports and provides documentation for real-time data processing. It is missing a caching mechanism in the open source version. This is reducing the utility and speed.

Future Proof: 3 The uncertainty from the reliance on a single commercial entity for Metabase reduces the future-proofness of the system.

Ready Made Dashboards: 1 There are no ready-made dashboards available.

Plugin Support: 3 Metabase supports plugins partially, only the creation of database drivers are supported. The plugin and plugin creation is not well documented.

Superset

OSS Business Strategy: 5 Superset is maintained and provided by the Apache foundation, which is a reputable open source foundation.

OSS Maturity: 5 The rapid development with a stable number of contributions over the lifespan of Superset shows long term support.

Current Data Compatibility: 5 Similar to Redash and Metabase, Superset supports PostgreSQL and Google Sheets, making it compatible with all current data sources used in SCA Tool.

Docker K8s Support: 4 Superset provides Docker images and Helm charts for deploying Superset, which synergizes with the deployment system of SCA Tool.

Visualization Variety: 5 By default, Superset offers over 60 different visualizations providing more variety than the other components.

Documentation Quality: 2 The documentation of Superset is sparse. It is missing some crucial information especially for the deployment using Docker and Helm charts.

Performance: 3 Apache Superset is deployed alongside many tools, such as its own PostgreSQL database, Redis and more. This costs additional compute and reduces performance.

API and Web Service: 4 Apache Superset enables users to view dashboards using the website and RESTAPI from the web server.

Self Service: 4 Superset has two ways to generate charts, either with a GUI or via a full SQL Integrated Development Environment (IDE). The GUI is adequate for non-technical users.

Granular Embeddings: 3 The system allows charts and dashboards to be embedded using iFrames. The embedding is not well documented.

Realtime Capability: 4 Due to the deployment alongside PostgreSQL and Redis, Superset becomes real-time capable, as it is able to store large data amounts quickly.

Future Proof: 4 Superset has an active community and support for multiple data sources, for local and cloud environment, in addition to the active feature development.

Ready Made Dashboards: 1 There are no ready-made dashboards available.

Plugin Support: 4 Superset has plugin support for visualizations and drivers.

3.5.1 Analyze results and choose a component

Table 3.1 combines the results from the previous sections. It adds the individual rating of each component for each criterion with the respective weight. Finally, the sum of all columns is calculated.

Table 3.1: MCDM for OSS Software Comparison

Criteria	Weight	Superset	Metabase	Redash
OSS Business Strategy	0.12	5	3	3
OSS Maturity	0.12	5	5	3
Current Data compatibility	0.10	5	5	5
Docker K8s support	0.10	4	3	5
Visualization variety	0.09	5	5	5
Documentation quality	0.08	2	4	3
Performance	0.07	3	4	3
API and Web service	0.07	4	4	4
Self service	0.06	4	3	4
Granular embeddings	0.06	3	4	4
Realtime capability	0.05	4	3	3
Future proof	0.03	4	3	2
Ready made dashboards	0.03	1	1	1
Plugin support	0.02	4	3	1
Total / Weighted Sum	**1.00**	**4.05**	**3.84**	**3.64**

From this result the conclusion is drawn, that Superset is the best ranking component and fits the criteria and weights the best. The result for Superset shows that Superset fulfills the requirements on average perfectly and in some aspects, even exceeds expectations. Superset is chosen for the architecture and implementation of this work. Due to the high average score of Superset a custom visualization tool for SCA Tool is not implemented. The BI system is build using Apache Hop, PostgreSQL, Flyway and Apache Superset.

4 Requirements

This chapter contains the requirements used for the implementation of the system that is build with the components selected in Chapter 2 and Chapter 3. These requirements are the basis for the architecture, implementation and measurements of the BI system for SCA Tool. The chapter is divided into two main sections, the starting and final requirements. Since an agile methodology was used for the implementation, the requirements were subject to change. The reason for the changed requirements is further elaborated on in Chapter 7.

4.1 Starting Requirements

The section on the starting requirements is split up into several sub-sections. As discussed in Chapter 2, BI systems consist of several software components. The first subsection is about the overall requirements that must be met by all the components. The latter subsections cover the individual requirements for the specific components. All subsections are internally split into the definition of functional and nonfunctional requirements.

4.1.1 Overall

Overall requirements describe the requirements that must be fulfilled by components of the BI system.

Functional Requirements

F-1: **Deployment in Docker**
The system and all of its subcomponents must be deployed via Docker for the local development.

F-2: **Deployment in Kubernetes**
The system and all of its subcomponents must be deployed via Kubernetes for the production and server environment. The deployment must also support Minikube.

F-3: **User Authentication**
Each of the subcomponents must have user authentication to ensure data integrity and access control over the subsystems.

F-4: **Administrative System Monitoring**
Administration users must have the option to observe the system and monitor the state of all actions.

F-5: **PostgreSQL Connectivity**
All subsystems must be connected to and interact with PostgreSQL databases.

Non-Functional Requirements

NF-1: **Data Integrity and Consistency**
All systems must enforce data integrity and data consistency for all data transformations and data migrations to prevent data corruption, ensuring ACID compliance.

NF-2: **Maintainability & Modular Architecture**
All systems must be modular and independent on other systems for deployment or availability.

NF-3: **Scalability**
All systems must support scalability for future expansion of SCA Tool in terms of data storage and processing.

4.1.2 ETL Tool - Apache Hop

Functional Requirements

F-6: **Data Migration**
The ETL Tool shall migrate data into one central data warehouse.

F-7: **Multi-Source Extraction**
The data migration step performed by the ETL Tool must support multiple input data sources in order to access all the production data sources.

F-8: **Automated Workflow Scheduling**
The data migration between databases must be automatically executed on a scheduled basis.

F-9: **Error Handling**
Apache Hop must provide a method to handle and observe errors.

Non-Functional Requirements

NF-4: **Execution Performance and Throughput**
All data migration steps must be performed at night.

NF-5: **Pipeline Observability**
Users must have the option to observe all pipeline runs in order to see any errors or track progress.

4.1.3 Data Warehouse - PostgreSQL database

Functional Requirements

F-10: **Centralized Data Storage**
The business data must be stored uniformly in a central data repository in the form of a data warehouse.

F-11: **SQL Query Support**
The data warehouse must provide standard SQL interfaces to allow other BI subsystems to execute queries for dashboard rendering and interactive exploration.

Non-Functional Requirements

NF-6: **Relational Data Modeling**
Data must be stored in a structured, relational format to facilitate efficient analytical querying.

NF-7: **Error recovery**
The data warehouse must be recovered after a catastrophic failure.

4.1.4 Visualization Tool - Apache Superset

Functional Requirements

F-12: **Visualization Representation**
The visualization tool must provide a visual representation of the BI data using charts and diagrams.

F-13: **Interactive Dashboard Generation**
Users with the permission must be able to generate dashboards in an interactive graphical manner.

F-14: **Embedding of Dashboards**
Dashboards must support external embedding into websites.

F-15: **Dynamic Filtering**
Users must have the option to dynamically view and filter the BI data in real time.

F-16: **Error Handling**
Similar to Apache Hop, Apache Superset must provide error handling in keeping the data consistent and indicating errors.

Non-Functional Requirements

NF-8: **Auto-refresh dashboards**
The layout must be responsive to always show the user the current state of the system.

NF-9: **Backend Result Caching**
All results and charts as well as queries must be stored in a cache to improve system speed availability.

4.2 Final Requirements

This section covers the final set of requirements. They are the result of the agile development approach. In the following paragraph, only the requirements that underwent changes in the development process are listed.

4.2.1 Overall

Functional Requirements

F-17: **Role-Based Access Control (RBAC)**
Access control needs to support the creation of multiple user groups with different permissions, as not all users are allowed access and modification permissions to the production data.

F-18: **Data Isolation**
All data must be isolated. The access to data must be limited to the components that need direct access to the data.

Non-Functional Requirements

NF-10: **Disaster Recovery**
All subsystems of the BI system must be recovered to a last working state after a disaster or failure.

NF-11: **System consistency**
In addition to data consistency, all data transformation steps must be

kept consistent to prevent dead locks.

4.2.2 ETL Tool - Apache Hop

Functional Requirements

F-8: **Automated Workflow Scheduling**
The data migration must be automatically executed on a scheduled basis. In addition to automation of the workflow scheduling, the correct execution of data extraction and transformation must be coordinated. The automation is only needed for the cluster deployment.

F-19: **Data Transformation and Cleansing**
After the data extraction step, Apache Hop must transform the data into a statistical representation, for example, combining raw inputs into abstract representations.

F-20: **Data Augmentation**
Apache Hop must add timestamps and other temporal data to the data fields missing temporal information.

Non-Functional Requirements

NF-4: **Execution Performance and Throughput**
All data migration steps and data augmentation must be performed automatically at night.

NF-12: **Data Pipeline Idempotency**
Pipelines must be idempotent. This means that consecutive executions of the same pipeline must not change the returning result.

4.2.3 Data Warehouse - PostgreSQL database

Functional Requirements

F-21: **Schema Versioning**
Changes to the database schema must be tracked and versioned using a database migration tool (e.g., Flyway or Liquibase) to ensure that the table schema remains perfectly synchronized across local, staging, and production environments.

Non-Functional Requirements

NF-13: **High Availability**
In the production Kubernetes environment, the PostgreSQL instance

must be configured for high availability utilizing a primary-replica architecture with automated failover to minimize potential system downtime.

4.2.4 Visualization Tool - Apache Superset

Functional Requirements

F-22: **Ad-Hoc SQL Querying (SQL Lab)**
In addition to the interactive dashboard creation, users must have the option to use advanced queries written in SQL to get more detailed insights on the data.

F-23: **Granular Row-Level Security (RLS)**
Access control of the BI data must be on the row level to have fine-grained control over knowledge.

Non-Functional Requirements

None of the nonfunctional requirements for the visualization tool changed in the development process.

The final list concludes the requirements used in the design and implementation of the BI system. The list is used in Chapter 7 in the evaluation of the system.

5 Architecture

This chapter covers the architecture of the BI system built for SCA Tool. The architecture combines the design, the selected components, and the requirements from Chapter 4. The first subsection briefly covers the existing architecture to provide the context for the change of the production system and the data sources used in the BI system. The remaining sections of the chapter focus on the BI system and its components and how they interact with each other.

5.1 Existing Architecture

The existing production system of SCA Tool is built around a Java BootSpring module with several services. The core of the production system is a Modulith, which is a modular monolith. All of the systems are part of a microservice architecture.

5.1.1 Modulith

This subsection provides more insight into the Modulith in order to elaborate on the changes made to production system for the data collection step. The Modulith handles all major business functions by either performing the action directly or delegating them to the microservices.

Functionality

The Java BootSpring Modulith is a modular monolith. This means that all business functions are separate modules within the Modulith. These modules are distinctly separated into packages, which makes communication only possible via cross-package accessible interfaces and events.

One of these functionalities is the source code analysis. As established in Chapter 1, one of SCA Tools' main functions is to scan code repositories for their licenses.

The scanning is started with the input of a code repository by the user. This repository is sent to the Analyzer and then the Scanner both of them are using

queued jobs. Both of these functions are not part of the Modulith, they are individual microservices. The results from the Analyzer contain a list of all found dependencies from the code repository the user gave as input. This result is further processed in the Modulith. The packages, which are part of the user input are stored with the license in the database table called ”package.” One issue with this system is the built-in caching mechanism. If a package is already in the package cache, the package is not scanned. This, in return, means the system does not create a ScanJob for the package and therefore does not save any information on the package scan request. The non-creation of ScanJobs for each package makes tracking the number of overall scans and scans per package impossible.

Another issue of the current system is data persistency. Users are able to delete VerUnits, users, and projects. VerUnits describe input code repositories combined with a specific version. This leads to the inability to calculate historic data accurately without storing intermediate data. In the case that a user is created, the overall user statistic is increased by one. If the user is deleted, the overall user statistic is decreased by one. However, if the users per day metric is calculated on the current state of the database, the calculations do not account for the users that were created and deleted before the calculation was started. This makes the statistic unreliable in the current state. In order to mitigate this, we need to store user information even after deletion while ensuring that no personal user data is stored.

Deployment

The Modulith has two different deployment methods.

The first is for local development, where the Scanner is started using Gradle. This is a prerequisite to start the Modulith and Scanner, which are also started with Gradle. On startup, the Modulith BootSpring application checks if the necessary microservices are running. If this is not the case, the module executes a Docker Compose file to start all the services needed, including the database and access control system.

The second development method is using Kubernetes and Helm charts. This deploys a production-ready system including all microservices needed for the system. After this, the images are built and and run to start the Modulith, Scanner and Analyzer.

Data Sources

The main data source for the Modulith is the main PostgreSQL database, which is further discussed in the next subsection on microservices.

5.1.2 Microservices

Alongside the Modulith, SCA Tool uses several microservices that handle critical functions outside the core business logic.

Functionality

The two core microservices are the Scanner and the Analyzer, which contain the business logic of scanning and analyzing the user repositories. These were explained in the last section.

Further microservices include:
Ory Kratos, which is an access control provider that is used to handle API and login security. Ory uses an internal PostgreSQL database to persistently store data used to manage the access control. This data includes the API keys, users, and sessions.

The PostgreSQL database that stores the production data from the Modulith, Scanner, and Analyzer. The production data, including separated user data, organizations the user is part of, the code bases, and Scanner and Analyzer results.

5.2 Changes in the Modulith

This section is about the changes made to the Modulith. The first step in creating a BI system, as shown in Chapter 2, is data collection. Currently, most modules collect all the operational data in the main PostgreSQL database. One of the missing key metrics, as mentioned before, is the collection of scanned packages metrics.

5.2.1 Package-Scan-Stats

The main change to the Modulith is done by adding more functionality to the package called ‘packages’ and ‘scans’ inside the Modulith. The goal is to gather statistics on the number of scans per package. The overall change to the production system adds the functionality that ScanJobs are also created for cached entries. This is done in the ‘scans’ package. The main logic is set up in ‘packages’, containing the package service. Here an automatically scheduled cron job is created. This job loops through all the created ScanJobs and gathers the package and version and stores it in the main database. In the execution the job stores the number of SUCCEEDED, FAILED, SKIPPED and CACHED scans. The sum of these is the total number of scans. The name of the new service is package-scan-stats.

5.3 The BI architecture overview

The main architecture of the BI system provides the design for the systems performing the last four steps in the architecture. These steps are as mentioned in Chapter 2: Data Integration, Data Storage, Data Processing, and Data Presentation. This section explains the overall architecture and the architecture of the individual components in greater detail.

Apache Hop is used in the step of Data Integration. For Data Storage a PostgreSQL database is used as a data warehouse. The data processing of the business data is done in Apache Superset. Superset also uses to visualize the data and generate charts and dashboards.

5.3.1 Data Flow

This section describes the architecture in greater detail by explaining how the data flows between components and how the different systems are communicating with each other.

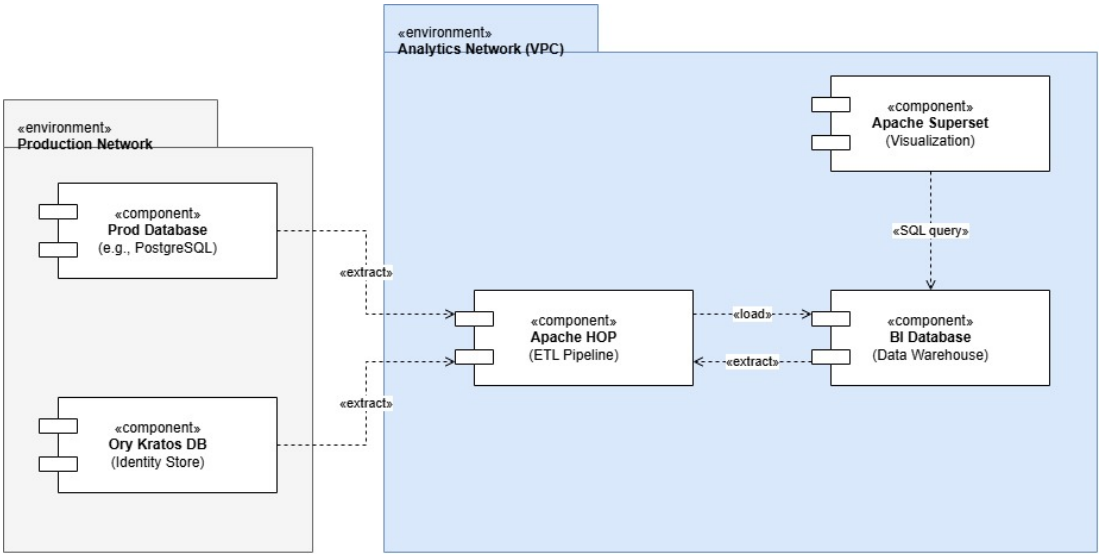

Figure 5.1: Data flow from and to the BI System

Figure 5.1 shows that Apache Hop is extracting the data from the different production PostgreSQL databases. The extracted data is transformed, augmented, and cleaned in Apache Hop. For the data that is subject to deletion, such as user data, the data is first extracted from the production database and then stored in the BI database. This is the case since Apache Hop also calculates statistics such as 'users over time' with the data stored within the BI database and it stores the resulting statistics back to the BI database.

The main workflow is run once per day. The main reason for the time frame is the computational demand of the BI system. The performance need is reduced by calculating the metrics once per day at night, in times of low overall load.

The access to the data is clearly structured. Apache Hop has read-level access to the production and Ory databases. The transformed data is stored in the BI PostgreSQL database, which results in Apache Hop having read and write access to the BI database. Apache Superset, as the visualization tool, only has read access to the BI database. This limits access to the data for the components, which adds security to the BI system. The write access to the data is even further limited, as the BI system besides Apache Hop does not have write access to the production environment.

Another advantage of this architecture is the modularity. This enables adding or changing the system easily. The following sections explain the architecture of the components in greater detail.

5.3.2 ETL system

For the extract, transform, and load operations, workflows and pipelines are used. Workflows consist of either workflows or pipelines. Pipelines consist of transforms, which contain the actual instructions performed on the data. Each pipeline consists of three parts of the ETL process: the first part is the data extraction, the second part is the data transformation, and the third part is the data loading. The main principle used in the pipeline creation is to separate the business metrics into individual pipelines in order to keep the ETL process decoupled, modular, and maintainable. In addition, the pipelines are built to keep the data persistent and idempotent.

These pipelines are part of workflows that are grouped by topics. There are five workflows in total. The first four of them process either user, project, package, or organization-based statistics. These four workflows are bundled into one main workflow that is scheduled to run at night in order to perform data processing once per day.

In order to execute the pipeline once per day, a Kubernetes CronJob is created to start the Apache Hop Runner. This execution schedule is only implemented in the cluster deployment. Overall two versions of Apache Hop are deployed. Apache Hop has two Docker images that are deployed. One of them starts the Apache Hop web server, and the other starts the Apache Hop runner. The runner deployment has the advantage of being started with a reference to the main workflow. The container is started, which triggers the run of the configured workflow, and terminates after the execution. The web server is used as a GUI based pipeline builder. The scheduling of workflows is only used in the cluster development, because the local deployment is mainly used for development.

5.3.3 Data warehouse

The data warehouse is built using a PostgreSQL database. For this the same application version is used as the database for the BI system. The reason for the usage of one database instance is the technical parity to the production system. The database system stores each metric in a separate table. For example, the metric on scans per package has its own table, which contains only the index, the package name, the time stamp of the BI data aggregation, and the number of scans. The data decoupling further ensures that the database is ACID compliant and reduces the blast radius of errors and failures in pipeline runs. In addition to this, PostgreSQL is error- and transaction-safe. This ensures that in case a transaction fails, the data is reverted and no corrupted state is written into the database.

Alongside the PostgreSQL database, a second instance of Flyway is deployed for the database migration. This ensures that changes to the database schema get stored and applied every time the deployment is started, using a Kubernetes init-container. It keeps the system up-to-date in regard to schema changes.

5.3.4 Data visualization and Data processing

The data visualization is done using Apache Superset. As mentioned in Section 5.3.1, Apache Superset only accesses the data from the BI database. This data is integrated into datasets. Superset accesses the data via a semantic layer and executes SQL queries dynamically to keep the datasets current. Therefore, Apache Superset does not only serve as a data visualization tool, but also as a tool for data processing. These datasets are the basis for charts. Charts are the visual representation of the data. These charts are aggregated into dashboards. Superset also provides storage and caching for the SQL queries and charts.

Similar to the ETL system, the data is grouped into four dashboards, one for each of the topics: users, projects, packages, and organizations. These dashboards contain the individual charts, each visualizing each of the metrics for each topic. This continues the separation of concerns and the modularity started in the ETL system. Another advantage is the ease of creating dashboards or adding new charts to the respective dashboards. This streamlines the development.

In addition, Apache Superset allows the administration of row-level access. This enhanced security is augmented by the feature to add roles to users, which limit their ability to read and write to the data by giving more fine-grained access.

5.3.5 Deployment

The architecture of the BI system deployment is similar to the deployment of the production system. The deployment is split in the development and cluster

deployment.

Local deployment All components of the BI system are deployed using Docker images and Docker Compose. This Docker Compose is separated from the Compose used to deploy the microservices for the Modulith. The BI Docker Compose starts all the services in the correct order. These are the BI database, Flyway, Apache Hop, and Apache Superset. The same Compose setup is also used to start all the subcomponents needed for Apache Superset. These include the web server, Redis caching, metadata PostgreSQL database, and query runners.

Cluster deployment Similar to the cluster deployment of the production system, Helm charts and Kubernetes are used as container orchestration tools. These Helm charts are integrated into the deployment structure of the production system, inside the default namespace. The Helm charts deploy to the Kubernetes cluster, which ensures a constant number of components are running, as the system automatically detects errors and redeploys services to keep all systems healthy. In addition, Helm charts abstract and unify the deployment of all components, which makes the system more modular and easier to maintain. The deployment expands the existing architecture and utilizes the same features to maintain maintainability, scalability, and error mitigation.

6 Implementation

This chapter further elaborates on the implementation done to realize the BI system built for SCA Tool. The chapter is divided into several subsections, identical to Chapter 5. The first subsection discusses the changes in the Modulith, which is part of the data collection step of the BI architecture. The second subsection covers the data integration step of the architecture. The third subsection covers the data warehouse, which is the data storage step. The fourth subsection explains the data processing and data representation step. The final subsection describes the implementation of the different deployment setups in more detail.

6.1 Package Scan Stats

The package_scan_stats service is the main change done to the Modulith and the production system. As described in Section 5.1, SCA Tool is currently not able to collect metrics on scanning numbers of individual packages. This feature is added with the package scan stats service and some more minor changes to the Modulith.

The main workflow of the package_scan_stats service is to derive the analytics for the package scans from the ScanJobs. These are stored in the production database and contain the package_url of the target package, which is to be scanned, and a scan status.

The first change was adding a new ScanJobStatus called Cached. This status is used in the ScannerServiceImpl. Both are part of the Scanner package. Provisioning of the functionality to create ScanJobs for all non-cached packages is the main focus of the ScannerServiceImpl. It filters out the cached jobs from the list of ScanJobs before the database entries are created. In addition, ScanJobs are added by saving the cached ScanJobs, instead of removing them from the original request list, into a new list, which is then used to create ScanJobs with the status set to Cached.

The collection of the analytics of scans is done in the package_scan_stats service, which is part of the package called package. This service is scheduled to run

every night at 2am using the cronJob scheduler built into Java BootSpring. The service stores the final result in a database table called package_scan_stats, the table has the package_id as the index and stores the count for failed, succeeded, cached, and skipped in separate columns. In addition, it also stores the name of the package in the same table. The package_scan_stats service has three functionalities.

The first functionality is reading the ScanJobs from the database. This access and sorting of the ScanJobs is done via the ScanJobService. The retrieved jobs are then parsed individually using the ScanJobs' package_id as index for the package_scan_stats table. Increasing the status count is by one depending on the status of the ScanJob. For example if the ScanJob succeeded, the value for succeeded_scans is increased by one.

The second functionality is storing the id for the ScanJobs that have the status of queued or running. The final status of a ScanJob is set by the scanner after the component is finished. The main issue is that when the package_scan_stats service retrieves the list of ScanJobs, some of the Jobs might be in the state of running or queued. Therefore these ScanJobs are stored in a second table called package_scan_unfinished, where only the id is stored. The advantage of this is, that on the next execution of the service these unfinished jobs can be incorporated into the statistics.

The third functionality is making the PackageScanStatsService more performant by storing the time stamp of the last analyzed ScanJob. This is done using a third table called package_scan_stats_metadata. It stores an index, the timestamp of the most recent ScanJob that was analyzed, the starting and finishing times of the PackageScanStatsService and the number of ScanJobs processed. This enables only processing the new ScanJobs, which were not analyzed on every consecutive run of the scheduled service. Furthermore, this table stores useful information on the execution time and processing numbers which allow for system monitoring and observability.

The full execution workflow of the package_scan_stats service is shown in Figure 6.1 and explained in the following paragraph.

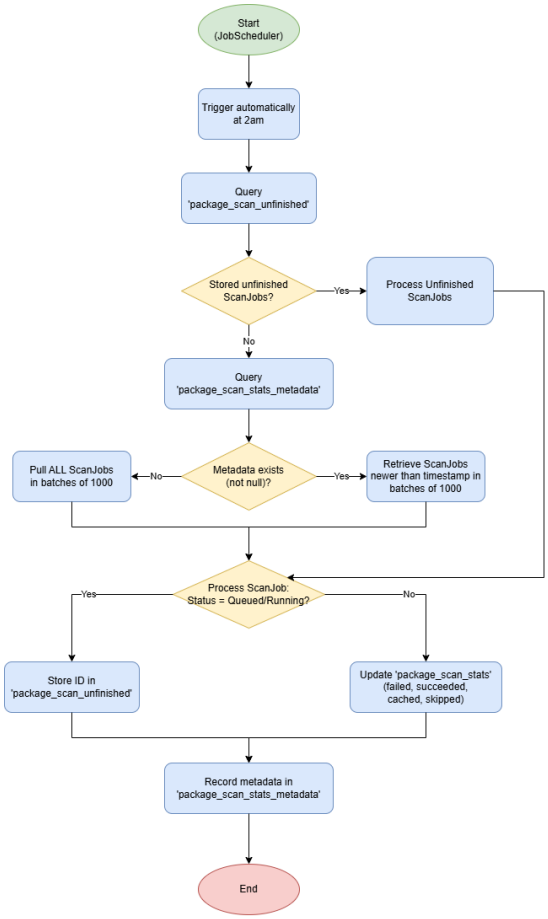

Figure 6.1: package_scan_stats service Flowchart

The scheduling of the service executes automatically every night at 2 am. Firstly, the table of unfinished ScanJobs from the previous day is analyzed. If the table is empty, the service continues. In case there are stored unfinished ScanJobs, the system retrieves them and processes them. The id from the table package_scan_unfinished is used to retrieve the ScanJobs from the main production table containing the complete ScanJob. This result is used to update the package statistic, and the original entry in the table package_scan_unfinished is deleted. If the package was not scanned before, a new entry in the package_scan_stats table is created for this package. Secondly, the information of the last analysis done by the service from the package_scan_stats_metadata table is retrieved. In case the metadata is null, the service pulls all ScanJobs in batches of 1000 entries to process. Otherwise, if the metadata entry exists, only the ScanJobs newer than the timestamp of the last execution are retrieved. Thirdly, the retrieved ScanJobs are processed. For running or queued ScanJobs the id is stored in the table package_scan_unfinished. Else, the package scan statistics are updated by increasing the counter of the relative scan result. After all ScanJobs are processed,

either by analyzing the result or storing them in the package_scan_unfinished table, the metadata is updated.

The addition of the package_scan_stats service concludes the data collection step in the BI architecture, as the production system already stores all of the core metrics, which are the basis for the BI system. The next step in the architecture is the data integration step.

6.2 ETL System

The ETL system is the main part of the second step in the BI architecture, which is data integration. As mentioned in Chapter 5, Apache Hop uses workflows and pipelines to execute the ETL operations. The next subsections explain the workflows and pipelines.

6.2.1 Workflows

Workflows are used as high-level organizations of pipelines and workflows. Their primary use is adding a hierarchical structure to the pipelines and therefore the ETL operations. The highest-level workflow is named main.

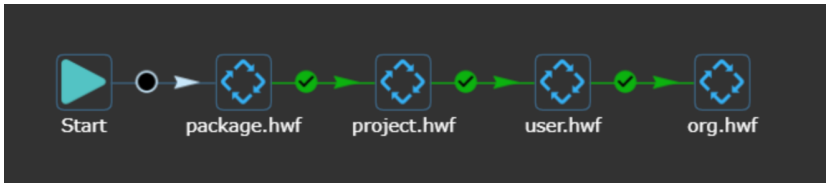

Figure 6.2: Main workflow

The main workflow executes the workflows for each of the groups of pipelines and metrics as mentioned in Chapter 1 of this work. These groups are the package, project, user, and organizations. The execution of the workflow starts with the start transform and executes each of the workflows consecutively. This is shown in Figure 6.2. The arrow with the green checkmark indicates that the next workflow is only executed if the previous workflow was successful. This is necessary, as some of the later workflows need the information from the previous workflow.

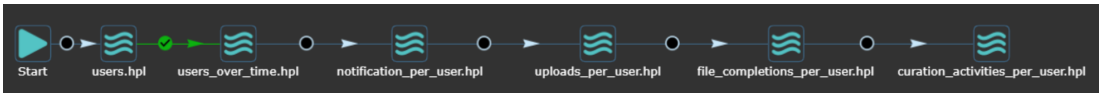

Figure 6.3: User workflow

The user workflow is shown in Figure 6.3. This workflow executes all the pipelines used to calculate the user-based metrics. The beginning is the start transform. The first pipeline that is executed is the user pipeline, as it is used to get the user data from the production database to the BI database while removing all personal data. The need for storing the users is explained in Section 5.1.1. The anonymized user data is used for all the consecutive pipelines. Therefore, the arrow with the checkmark is used to ensure that the user pipeline finished successfully before the other user-based pipelines are executed.

The same layout of the workflows per group is used for the other groups. The execution of all the workflows and pipelines within the workflows is ordered by the data need. If a pipeline uses data from a previous step, it is executed later in the setup. This ensures the data's consistency. Each metric that is collected is processed in an individual pipeline, which makes the system maintainable and modular.

6.2.2 Pipeline

This section explains the pipelines in more detail and explains how the pipelines are used for data extraction and metric calculation.

Pipelines are the core functions to execute the extract, transform, and load operations. Each pipeline consists of several transforms. These either extract input data from a data source, transform data, or load the data into a data source. The BI system for SCA Tool consists of 28 pipelines. There are three different types of pipelines, each of them used for a different type of metric.

Storage Pipeline The first type of pipelines used is the storage pipeline. These are used to store the data that is subject to deletion in the production system. These pipelines store anonymized data for projects, version units, and users.

Figure 6.4: User storage pipeline

Figure 6.4 shows a storage pipeline for the user data. The first transform is called GetUsers, it is used to extract the user data from the production environment. This is done by querying the data using SQL.

```sql
SELECT
    id,
    organization_id,
    identity_id,
    state,
    created_at,
    updated_at,
    deleted_at,
    waitlisted_at,
    state_changed_at
FROM users
```

Listing 6.1: User data extraction

This SQL statement shown in Listing 6.1 ensures that only non-personal user data is stored. This ensures data compliance. After the data extraction, the data is transformed in the second transform. The transformation maps the input, which has the datatype of Universally Unique Identifier (UUID), to the exact output datatype of varchar(255). The final transform is used to insert and update the user data into the data warehouse. This insert and update is configured to create a new entry in the user database if the tuple of the user id and created_at fields is not present in the database. In case the value tuple is present, the following data fields are updated: organization_id, identity_id, state, updated_at, deleted_at, waitlisted_at and state_changed_at. The insert and update operation has two advantages. Firstly, it ensures that each user has only one entry in the database, improving data quality. Secondly, the pipeline execution is idempotent, as the values are only updated if the data has changed.

Time-Series Pipeline The second type of pipeline is used to extract time-series data from the production or BI database. This is done for the following metrics: users over time, version units over time, total version units per day, users over time, projects over time, total projects per day, total issues per day, scans per day, projects over time, and organizations over time.

The difference between version units over time and total version units per day is the focus. Version units over time analyze the total number of version units for every day. Since version units can be deleted, this number shows trends in the total number of version units. The version units per day analyze the newly created version units for every day. The last metric indicates user behavior more directly, as the increase in newly created version units indicates days of higher activity.

An example pipeline for aggregating time-series data is the users-over-time pipeline, shown in Figure 6.5.

Figure 6.5: Users over time

This pipeline consists of two transformshe . Tfirst is of the type table input. It uses SQL to query the BI user table and extracts the time series data within the same query. This is different from the other pipelines, as the data extraction and data transformation of the ETL process are handled in one query.

```
SELECT
    calendar.metric_date,
    COUNT(u.id) AS total_count
FROM (
    -- Generates the date series from the first user ever
    created
    up to today
    SELECT generate_series(
        (SELECT MIN(created_at)::date FROM users),
        CURRENT_DATE,
        '1 day'::interval
    )::date AS metric_date
) calendar
LEFT JOIN users u ON
    u.created_at::date <= calendar.metric_date
    AND (u.deleted_at IS NULL OR u.deleted_at::date >
    calendar.metric_date)
GROUP BY calendar.metric_date
ORDER BY calendar.metric_date;
```

Listing 6.2: User time series data extraction and transformation

The SQL query seen in Listing 6.2 first generates a continuous series of dates, ranging from the creation date of the first user to the current day. This is the calendar. The left join is used to add the user metric for each day. This is done by querying the user table for every day simultaneously to see if a user was active and the deletion date of the user is null or greater than the date. The final step groups the result per day and returns the list sorted. This sorted list is used by the second transform in the pipeline to insert and update the data. The data is only inserted if the entry did not exist. Otherwise, the metrics are updated if the

value changed. For the metrics that calculate the new users or projects per day, the join operation inside the SQL query is changed. It only calculates the newly added values per day by comparing the iterating date with the creation date.

Statistical Data Pipeline The third type of pipeline is used to calculate statistical data with the addition of timestamps. These pipelines calculate the median, average, and maximum values for a given category. Statistical pipelines are used for the following metrics: API keys per organization, components per version unit, curated packages per organization, curation activities per user, file completion per user, issues per project, scanners per organization, uploads per user, users per organization, and vulnerabilities per version unit.

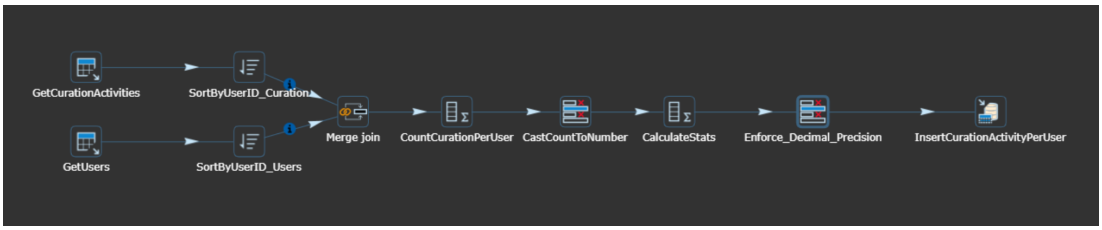

Figure 6.6: Curation activities per user

Figure 6.6 shows the main workflow of these pipelines using the curation activities per user as an example. These statistical pipelines are the most complex pipelines, as they consist of the highest number of transforms. The increase in complexity arises from the need to join the data from the user table and the curation activity table using the user ID.

The first step is the retrieval of the active users from the BI users table. This is done by only querying the users where the deleted_at field is null. The data is augmented with the reporting date. The reporting date contains the current date in order to ensure time series observation across the statistical data. The results are then sorted by the user id. In the same step all curation activities are queried from the production database. The curation activities are also sorted by the user id values.

The second step is merging the data streams together using a left outer join. This ensures that users without curation activities are not removed from the data stream. Apache Hop can only perform merge joins on sorted data, which is the reason for the sorting in the step before.

The third step is aggregating the merged and sorted data stream. The GroupBy transform calculates a new value based on this date and the curation_count per user. This stream contains the numeric value of curation activities per user. The new data stream is first transformed into the correct data type, a number with

the length of ten and a precision of two. The data stream is used to calculate the final values. The GroupBy transform calculates the maximum value, the average, and the mean curation activities per user on the data stream. The final data stream is parsed through the selected values' transform to ensure a uniform precision of the floating point values.

In the final step the data is inserted and updated into the database. In case the database contains an entry for the reporting date, the data is updated. In the case the reporting date is not present in the database, a new entry is created. This makes the pipeline execution idempotent.

This pipeline is used for most of the metrics mentioned in the beginning of the paragraph. The only two exceptions are the metrics vulnerabilities per version unit and components per version unit. As seen in Section A in the appendix, the metrics use a simpler pipeline similar to time series data pipeline. Due to the fact that the version unit table in the production database contains the number of vulnerabilities and components as a database field. This means that the number can be directly extracted from the table and augmented with the data of the data extraction before inserting. The metrics on the package statistics are calculated the same way on the package_scan_stats table from the production database. The metrics gathered for packages are package-scan-count, which sums the succeeded, failed, skipped, and cached entries for each package version up and saves it into the database. The metric package scan count without version aggregates this data across versions of the same package. As an example, the total scans for Axios 1.0 and Axios 2.0 are aggregated. The other two metrics regarding packages analyze the data for the package with the most successful and failed scans.

This concludes implementation of the workflows and pipelines, which are used to perform the ETL process of collecting data and loading it into the data warehouse. The process maintains idempotence, decoupling, and maintainability.

6.3 Data warehouse

The data warehouse used to store the BI data is implemented using PostgreSQL. Every metric is stored in a separate table to ensure modularity and reduce dependencies. The schema is fully denormalized and has a pre-aggregated architecture. One major advantage is that the schema is read-optimized, as table joins are not needed to retrieve a metric. In addition to the metrics, each table containing deletable objects is also stored in an individual table as described in Section 6.2. User accounts are used to control the access to the business data. The BI database has two user accounts that are created: one read-privileged and one write-privileged user. Apache Hop uses the write-privileged user as the

ETL process loads the data into the data repository. Apache Superset, the visualization layer, uses the read-privileged user as it queries the data for the final processing and visualization.

The table schema is updated with the utility of Flyway. For the database migration SQL scripts are applied sequentially upon startup. These scripts define the table structure. Flyway adds a table to the BI database called Flyway_schema_history, storing the name and hash value of the migrations that are applied. This ensures that migrations are only applied once in order to keep the database schema stable. One advantage of storing the hash of the migration file alongside the name is the detection of changes to the previous migrations. In the case that Flyway already applied a migration script with the same name but different hash, an error is thrown. This enhances data and database integrity and ensures that the database is consistent.

This allows the data to be stored for the visualization in graphs and dashboards.

6.4 Visualization Layer

The last two steps of the BI architecture are data processing and data presentation. Both steps are done with Apache Superset. The advantage of the storage solution, as discussed in the last section, is the read-optimization by separating every metric into one table without dependencies. Apache Superset generates dashboards to visualize the metrics. The process to generate the dashboards consists of three steps.

The first step is generating datasets from the tables in the data warehouse. Including the querying of the tables using SQL. Since every table contains the precalculated metrics. In Apache Superset, datasets are the basis for all data operations.

The second step is generating charts from the datasets. Apache Superset provides a GUI tool to generate charts. At first the user selects the graph type, such as a bar graph. Then a dataset as a basis for the content of the chart is chosen. In the final step the chart is configured to display the data in the desired manner. Figure 6.7 shows the creation of the chart for the curation activities per user. This step is the data processing step in the BI architecture.

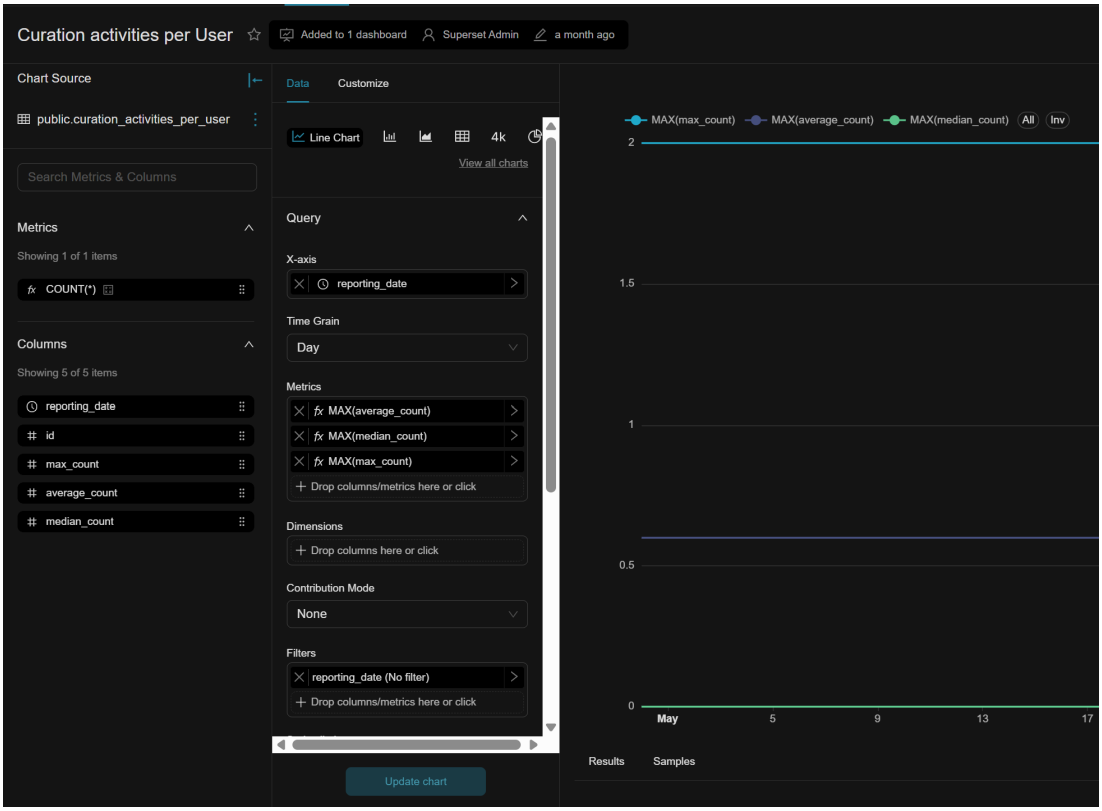

Figure 6.7: Curation activities chart

Figure 6.7 shows the different parameters set in order to generate the visualization. Testing data was used to generate the chart. First, the line chart from the top row is selected. For the x-axis, the reporting date is selected because the metric is time-series based, allowing the changes in curation activities over time to be displayed. The time grain is set to one day, as the metrics are calculated per day. The metrics displayed are the average, median and maximum count for the curation activity. A filter is set to reporting_date because the x-axis is set to show the time-series data. Finally, once the update chart button is clicked, the preview of the chart is displayed on the right. The underlying SQL query used to display the data is automatically generated by Apache Superset. Charts based on other visualizations are created in a similar way. For other chart types, the first selection of the graph type is changed to a pie or calendar chart. Similarly, the inputs for the x-axis are changed to the query dimensions and time column. The metric selection to display the data as well as optional filters remain identical. These are individualized for every metric. The created visualization for the pie chart can be seen in figure Section E and in Section D in the appendix.

In order to generate the final dashboard, all the charts generated are consolidated. In the case of the BI system for SCA Tool, four dashboards were created,

one for each group of metrics. The dashboards show the metrics grouped by users, projects, organizations, and packages. The following paragraph explains the package dashboard exemplary. The other dashboards are listed in Section H, Section G and Section F in the appendix.

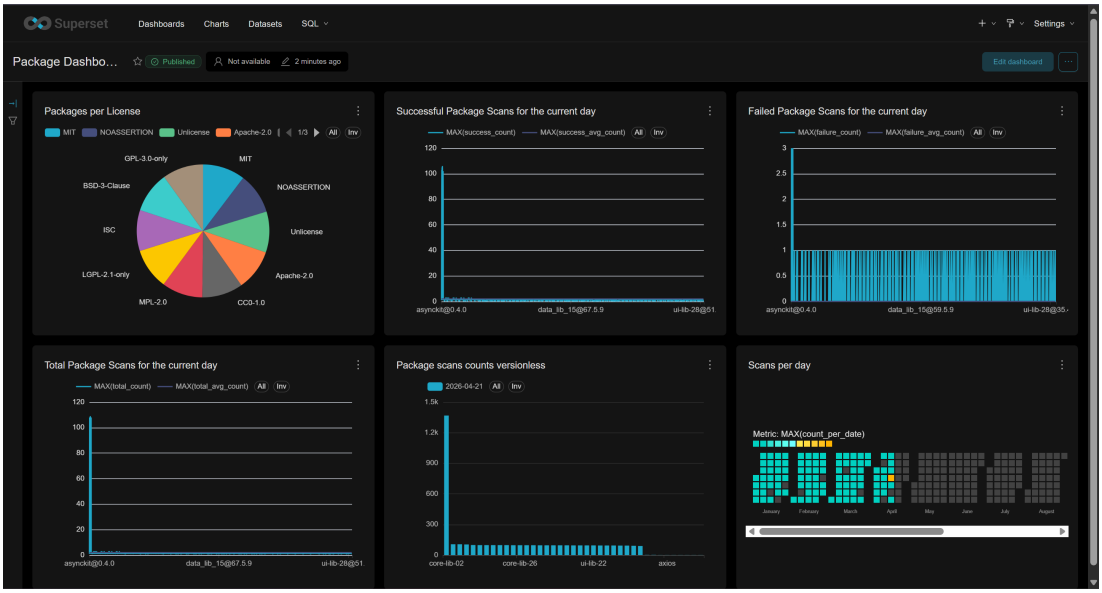

Figure 6.8: Package dashboard

The dashboard shown in Figure 6.8 shows the package dashboard. It consists of several metrics, each of them displayed in a chart that is created with a dataset. These dashboards are curated by users by selecting charts. These charts are automatically updated when new data is available in the dataset and BI database table.

This concludes the implementation's last step of the BI architecture, which started with the data collection in the Modulith and production environment. Followed by the data integration and additional processing done with Apache Hop in the ETL pipeline. The data storage in the PostgreSQL-based data warehouse is the basis for Apache Superset, which is used in the final data processing and data presentation.

6.5 Validation

The validation and testing of the system is done using end-to-end, unit, and integration tests. For this purpose, a custom SQL script was developed, which is shown in Section I in the appendix. This SQL script is executed on the main Modulith database. It populates all tables in the development database that are

accessed by the ETL pipelines. The first part of the script deletes all data entries to ensure that only the test and schema data is present in the database for the testing. The second part enters the test data into the Modulith database.

As a unit test, the user can run the pipelines manually in the Apache Hop web deployment to analyze and preview the results of every transform in each pipeline. The same check is made for the BI PostgreSQL instance and the corresponding Apache Superset chart of the same metric. In all three cases the user can either get the execution results from the container logs or the GUI, the visual validation of database is done in IntelliJ. Since every metric is independent from other metrics, the testing SQL script is used as a test fixture for the unit test for each of the pipelines. Every pipeline generates a verifiable value for each metric. The value is compared with the predetermined value. These predetermined values are added as comments into the validation script in order to document the correct results. If the result for the pipeline does not match the predetermined value in the dataset, an error has occurred. It is used as a unit test for the database because every metric is stored in a separate table. In addition, it serves as a unit test in the visualization tool because the metrics are shown as individual charts. Therefore, if the database entry for the metric is wrong or the chart displays the wrong value, an error has occurred.

Integration testing is also done with the same test fixture script because the automated pipeline execution loads the resulting metrics into the database. Apache Superset queries the data automatically from the database for the dashboards and charts. As a result, if the metric value for a statistic is correct in the database but incorrectly displayed in the visualization layer, an integration error has occurred. Similarly, if the result from a pipeline diverges from the database entry, an error has occurred. The combination of the automatic integration of all subsystems makes the test set an end-to-end test. Due to the high integration of the components and the difficulty for automatic unit testing in the visualization layer, a manual approach to testing was chosen.

6.6 Deployment

The deployment utilized multiple Docker Compose files in the setups. Docker and Docker Compose are used for the local development, whereas Helm and Kubernetes are used for the server deployment. This section explains the deployment of all components used to build the BI system.

6.6.1 Docker

The Docker deployment is done through a split-up Docker Compose setup. This setup is in a separate folder in the root of the project's directory called BI. This

folder contains the first Docker Compose file that is used to start the core BI services, such as Apache Hop web and runner. It also starts the PostgreSQL database, with Flyway as a migration tool. The Docker Compose setup ensures the use of the same network as the local main system environment in order to allow for communication between the BI and the production system. The dependencies within the docker compose file ensure that all services are started in order.

The configuration files for PostgreSQL and Apache Hop are located in the Helm charts, which are elaborated upon in the next section. The configuration folders that contain the migration scripts are mounted into the containers upon startup. Concretely, the folder containing the workflows and pipelines for Apache Hop is mounted into Hop. The SQL migration scripts are mounted into the Flyway container. The configuration files also contain the database access and authentication information. Since all services are run locally for testing and development, no secret management is needed, so all secrets are entered in plain text.

The second Docker Compose file is used to start the Apache Superset tech stack. The Superset frontend, backend workers, Redis, and PostgreSQL database services are started. The setup needs the complete Docker folder from the upstream Apache Superset repository. The folder is needed, as it contains the setup scripts for the server, database, and services. It also starts the Superset init container.

6.6.2 Kubernetes

The Kubernetes deployment of the BI stack is integrated into the deployment of the production system. Separate folders are used to store the Helm charts leveraged to deploy the BI components. These folders are located within the 'k8s/charts/', alongside the production system helm charts. The current subsection explains the deployment of the different components utilized in the BI architecture for SCA Tool. Identical functional configuration files are used for the Kubernetes deployment and Docker Compose deployment. This ensures a single source of truth.

ETL System The ETL system is deployed using Helm scripts. The main Helm script is the values.yaml. Both of the Apache Hop instances are deployed with the values.yaml file. The first is the Apache Hop web server, which is used to manually trigger workflows and debug pipelines. The second one is the Apache Hop runner. A predefined workflow is automatically executed on startup of the Apache Hop runner. After this the runner exits.

ConfigMaps are used to load the configuration files into the Apache Hop file system. ConfigMaps first load the files into the cluster and then, in a second step, these files are loaded into the container. The complete configuration is

moved into the same directory of the Apache Hop container. This is done for both the runner and web server.

The configuration contains the database management files in order to connect Apache Hop to the production and BI database. The ConfigMap contains a bash script shown in Section B in the appendix, to swap the plaintext secrets used for local development with the cluster secrets of the databases. Therefore, no cluster secrets are stored in plaintext.

Data warehouse The PostgreSQL database for the data warehouse uses the same setup as the production and Ory Kratos PostgreSQL instance. For this, a configuration file is added in the charts folder for PostgreSQL. For the production system and the BI database. Inside this chart the read-only user is created. It is also added for the production databases. The secrets are created in the secrets folder within the charts. For the cluster deployment, these charts are overwritten.

Similar to the ConfigMap used for the Apache Hop deployment are the SQL migration files loaded into the Flyway pod. Flyway reads the migration files and applies them to the database.

Visualization system Apache Superset has the most complex Kubernetes and Helm setup. The complexity roots in the adjacent services. This is similar to the Docker Compose setup. The services are rooted in the values.yaml file. The charts start the web server, database, Redis caching mechanism, worker, Celery worker, and init script.

The inclusion of the configuration files is done with the ConfigMaps. Apache Superset dashboard exports are highly nested configuration files. ConfigMaps include the files in a read-only folder. In order to edit the dashboards, the init script shown in Section C in the appendix is used to move the dashboards, including the database configuration, into the modifiable file system of the Apache Superset container. The init script does the following step. At first, it also applies the database migrations needed to update the database, which is part of the Superset deployment. Then access rule-based users are created. These can either see and modify the complete system, read the whole system or read the data they are given permission to. Secondly, the bash-based init script executes a Python script that loads the configurations into the file system. The Python script is also used to parse the database connection files and change the database access credentials, from the local passwords and ports to the cluster service IP and secrets. Then the Superset import command is called to import the folder into Superset's web server to create the datasets, charts, and dashboards. In order to prevent this import from crashing, the Python script scans the dashboard folders for orphan charts and sanitizes the data.

This concludes the setup needed for the implementation of the system, which was designed in Chapter 5, under the requirements set out in Chapter 4.

7 Evaluation

This chapter evaluates the implemented BI system build against the requirements set out in Chapter 4. Furthermore, dictionary context is provided for the changes of the requirements from the starting to finishing requirements.

7.1 Development

This section evaluates the limitation in the development process of the BI system the SCA Tool. The BI was built with the aim of being fully integrated within the main development and deployment of SCA Tool. In the development of the BI system, team communication errors lead to the parallel development of the BI system and the production system. The complete BI system is currently on a branch of the main system. The reason for the parallel development of the BI system and the production system is a complete refactoring of the Scanner and ScanJob system. One additional change in the early beginning of the development is the move from the admin app to the web server. Initially, the idea was to integrate the dashboards into the admin app used to orchestrate organizations. This plan changed because the integration removes the ability to search, filter and update data ad hoc. Therefore, the dashboards are deployed on Apache Superset's internal web server.

The refactoring of the Scanner The refactoring of the scanner updated to version 2 in the major refactoring. Scan inputs are created from version units that were created by the user. These scan inputs is sent to the scanner via the newly established gRPC Remote Procedure Calls (gRPC) API. The scanner then creates and processes the scans on the file level. After the scanner finishes the scan, the results are stored in a dedicated scanner database. The Modulith accesses the following data from the newly created database: the status of the scan of the version unit and the summary of the codebase. This summary contains the discovered files and licenses. The Modulith also accesses the file-based result in order to get the license per file in the codebase. For this, the scanner uses an API interface and a new dedicated PostgreSQL database.

Development conflict The main divergence from the new scanner and the development of the BI system is the implementation of the package_scan_stats service, which is used to analyze the scan results per package. The service can not be merged into the current production system, which is using the scanner v2. The first change is that the new scanner does not store the scanner-based data in the Modulith. This means that the service cannot access the data using the internal scanner service. The second change is that the new scanner does not support the concept of ScanJobs, as these were replaced by ScanInputs and findings. Each of these does not contain the package ID or package name combined with the scan result for that package. In conclusion, the package_scan_stats service can access neither the necessary data structure in the Modulith nor in the scanner, because the concept of ScanJobs was replaced. Since the package_scan_stats service depends on the usage of ScanJobs, for the analysis and the storage of the unfinished scans, the service needs a complete redevelopment. The absence of the ScanJob concept means that the functionality cannot be moved from the Modulith into the scanner.

Reimplementation of the BI system In order to integrate the BI system into the current main development branch of SCA Tool, a reimplementation was necessary. The adapted system analyzes a subset of the metrics gathered in the main implementation of the BI system. Despite the change of the scope, the main architecture remains functionally identical, ranging from the data collection to the data representation. Between the original state of the BI system and the state of the current application codebase, the new scanner is the main difference. The migration of the new BI system into the current main development branch is compatible with these changes, including the changes of the data modeling of the scanner. Due to the modularity of the microservices, less adaptation for the realignment was needed for them.

The reimplementation compatible with the current state of SCA Tool covers the ETL pipeline built with Apache Hop. Integrating also the PostgreSQL-based data warehouse and the visualization tool built with Apache into the revised version. Removing the outdated data models from the Modulith is the main change to the BI system. Therefore, all database tables that aggregated the unsupported metrics are removed. In addition, the dashboards were altered, and the charts for the metrics that were deleted were removed. The current metrics gathered are shown in Table 7.1. Other changes include adding the read-only user for the newly created Scanner database, as well as adapting to the changes in the Kubernetes deployment system. Due to the migration of the BI system into the main development, there was no quantitative analysis possible due to the time constraint of the project. Therefore, no evaluation of the BI system performance in the main production environment was conducted. The evaluation dataset for the system testing and validation was recreated and added

into the updated version. This ensures the same testing standard as for the original implementation, shown in Section 6.5.

In summary, the overall architecture is ready for integration with an open pull request as of writing. It implements the complete pipeline needed for the BI system, including all pipelines and dashboards for the subset of metrics. In addition to the implementation, a Readme.md file in the BI folder was created in order to record the changes needed for the deployment in the production system as well as keep a record of architecture decisions.

7.2 Changes in requirements

This section covers the changes in the requirements that originated from the agile approach to software development. This section refers to the explicit distinction of the requirements into the start and finish in Chapter 4. The main changes are additions or changes to the existing requirements. The first list includes the changes to the functional requirements. It is denominated as F-C, and the second list includes the changes to the non-functional requirements, which is denominated as NF-C

F-C-1: **RBAC**
This requirement was added because the simple access management is not sufficient, as it does not include the distinction between read-only and write users.

F-C-2: **Data Transformation and Cleansing**
The initial ETL pipelines were designed to only aggregate data. In order to make the data in the database read-optimized, the calculation and therefore the main data transformation and cleansing must be performed in the ETL pipeline.

F-C-3: **Data Augmentation**
Similar to the requirement before, the main calculation is done in the ETL pipeline. This includes adding the time series component to the data in the ETL pipeline.

F-C-4: **Automated Workflow Scheduling**
This requirement was further refined due to the added capabilities of data augmentation. Some pipelines depend on other pipelines, creating dependencies. The automatic scheduling must therefore ensure a correct order of execution. The workflow scheduling is only necessary for the production system, because the local development is only used for developing.

F-C-5: **Schema Versioning**

The database contains over 20 tables, which are susceptible to the changes from the production system. In case a new feature is added in the production system, a new metric and table needs to be created in the BI system. This need translates into versioning requirements that need schema updates and maintenance.

F-C-6: **Ad-Hoc SQL Querying (SQL Lab)**
In order to query the data for custom charts, Apache Superset must provide the capability for ad-hoc data exploration. This is needed because ready-made dashboards may not answer specific questions.

F-C-7: **Granular RLS**
In order to provide access to the data based on the role in the organization, row-level security is needed. In order to control the access on a fine-grained level without exposing the full project data to all employees.

NF-C-1: **Disaster Recovery**
The need for disaster recovery comes from the need for systems to be available and maintainable.

NF-C-2: **System consistency**
The need for system consistency comes from the need for idempotent operations.

NF-C-3: **Execution Performance and Throughput**
Due to the higher demand on ETL pipelines, the data execution time and throughput must be made performant in order to ensure data consistency. The consistency is improved as the production data is susceptible to change while the ETL pipeline is executed.

NF-C-4: **Data Pipeline Idempotency**
Since the computational demand of pipelines contains data augmentation and data cleansing, idempotency is needed. Pipelines must provide the same result for the same data, independent of the pipeline execution. This is especially needed in case a failed workflow is retried.

NF-C-5: **High Availability**
The BI data contains time-series data. For the correct display and analysis of the historical statistics of the data, it is needed that the data be backed up and replicated.

7.3 Evaluation of the requirements

This section of the evaluation covers final requirements and analyzes if the system fulfills the requirements or not. A requirement is either fulfilled, not fulfilled, or

partially fulfilled.

7.3.1 Overall

Functional Requirements

F-1: **Deployment in Docker**
Fulfilled, as the system is completely set up for local development in Docker. In the end-to-end test process, the validation was also performed on the components deployed with docker-compose.

F-2: **Deployment in Kubernetes**
Fulfilled, as the system is completely set up for the cluster deployment in Kubernetes. In the end-to-end test process, the validation was also performed on the Kubernetes cluster.

F-3: **User Authentication**
Partially fulfilled, users are manually created for all services. The accounts must be created by the admin for each user. The authentication was tested by logging into the system in the manual validation of the test data set.

F-4: **Administrative System Monitoring**
Partially fulfilled, Apache Hop provides detailed information on the pipeline execution. Apache Superset and PostgreSQL only provide the container logs. The Apache Superset dashboards must be manually checked for data inconsistencies. This was tested by applying the testing data set and manually observing the logs.

F-5: **PostgreSQL Connectivity**
Fulfilled, all components used in the BI system are connected to PostgreSQL instances.

F-17: **RBAC**
Fulfilled, user accounts with different access are created; namely, read-only and write-access users. This was manually tested by trying to apply the test set to the production database using the read-only user, which did not work and therefore shows that the roles-based system limits the access.

F-18: **Data Isolation**
Fulfilled, all components have only access and permissions for the data directly needed.

Non-Functional Requirements

NF-1: **Data Integrity and Consistency**
Fulfilled, all components have used the database drivers for PostgreSQL

and therefore use the built-in failure recovery. Furthermore, the ETL pipelines are idempotent.

NF-2: **Maintainability & Modular Architecture**
Fulfilled, each component is independent from other components. The system is completely modular due to the use of containers.

NF-3: **Scalability**
Not fulfilled, the ETL pipelines, due to the need for sequential execution, the pipeline runs are not fully scalable, as Apache Hop is limited in execution. In addition, one PostgreSQL instance is not scalable for large user numbers.

NF-10: **Disaster Recovery**
Partially fulfilled, as all pipelines and migrations are idempotent and the state is recoverable as a second execution after a disaster recreates the state. In case a container fails in the cluster deployment, Kubernetes is self-healing. In the case the ETL pipeline is not executed on a given day, the time series for that date is irrecoverable.

NF-11: **System consistency**
Fulfilled, each component is using idempotent executions; this includes the Apache Hop pipelines and the Flyway migration. Since the database is stored like the production system and all of the other components are stateless or recoverable, the system is consistent.

7.3.2 Apache Hop

Functional Requirements

F-6: **Data Migration**
Fulfilled, as the data is migrated from the production system into the BI system. This is tested by executing the end-to-end test.

F-7: **Multi-Source Extraction**
Fulfilled, as the data migration supports multiple data sources: these are the production system, the Ory Kratos database, and the BI database. In the test set, the data from both the Modulith database and the BI database are used and tested.

F-8: **Automated Workflow Scheduling**
Fulfilled, the data pipelines are automatically scheduled for the cluster deployment. The schedules are tested using the validation set and observing the logs in Kubernetes.

F-9: **Error Handling**

Partially fulfilled, Apache Hop indicates the system status well. However, it lacks the capability to heal a failing pipeline. If a pipeline fails, manual reruns are necessary.

F-19: **Data Transformation and Cleansing**
Fulfilled, the data is transformed into BI data and the data cleansing anonymizes the user data, which is tested by executing the user pipeline in the test SQL set.

F-20: **Data Augmentation**
Fulfilled, since each metric is augmented by the timestamp of the reporting date. This is done for every metric and tested using the validation set. The test augments the metrics with the reporting data, which is then used as x-axis of the charts.

Non-Functional Requirements

NF-4: **Execution Performance and Throughput**
Not fulfilled, the execution time of the pipeline is related to the data transformed. Due to this dependence, no execution time span can be guaranteed.

NF-5: **Pipeline Observability**
Fulfilled, Apache Hop executes the pipeline and logs the process and operations in the database. This includes failures, errors, and warnings and more information, such as the number of read and write operations per transform. Setting the execution logs to detail, logs all actions performed in the transforms, this was alongside the test of the scheduled pipelines.

NF-12: **Data Pipeline Idempotency**
Fulfilled, as the pipelines use insert and update operations. The data for a specific date is only updated if the data changes. In order to test idempotency, the same pipelines are executed multiple times using the test set, which always show the same result.

7.3.3 PostgreSQL Data Warehouse

Functional Requirements

F-10: **Centralized Data Storage**
Fulfilled, as the BI data and every metric are stored in one BI data warehouse instance. The results of all tested pipelines are stored in the data warehouse therefore, it is tested.

F-11: **SQL Query Support**
Fulfilled, PostgreSQL is a SQL database.

F-21: **Schema Versioning**
Fulfilled, Flyway migrates the database schema upon startup or manual execution. The test set indirectly tests the schema versioning, because the the pipelines load the data into the data repository, which fails if the schema is not updated.

Non-Functional Requirements

NF-6: **Relational Data Modeling**
Partially fulfilled, the final metrics are stored in the data warehouse, but the data is not optimized for large queries within the data warehouse.

NF-7: **Error recovery**
Not fulfilled, as the PostgreSQL instance relies on the same mechanism of storage as the production database. In case the backup fails for one database, the other also fails.

NF-13: **High Availability**
Fulfilled, as the PostgreSQL instance relies on the same mechanism of storage as the production database.

7.3.4 Apache Superset

Functional Requirements

F-12: **Visualization Representation**
Fulfilled, Apache Superset provides visual representations for every metric. This is tested in the manual verification of the dashboards and charts using the validation dataset.

F-13: **Interactive Dashboard Generation**
Fulfilled, Superset is used to generate dashboards using an interactive GUI. It was used in the initial implementation of the dashboards and therefore tested.

F-14: **Embedding of Dashboards**
Partially fulfilled, dashboards can be embedded into external websites. The feature is not used in the BI system.

F-15: **Dynamic Filtering**
Fulfilled, users can access datasets and apply filters on the data. This is tested in the dashboard creation process and can be seen in Figure 6.7.

F-16: **Error Handling**
Partially fulfilled, dashboards indicate errors in the dashboard. Superset, however, has no error-handling capabilities for errors in the database.

F-22: **Ad-Hoc SQL Querying (SQL Lab)**
Fulfilled, Apache Superset has the SQL Lab enabled, which allows users to query data ad hoc, with SQL syntax highlighting.

F-23: **Granular RLS**
Fulfilled, admin users can control the access to data and charts on a per-row level for other users.

Non-Functional Requirements

NF-8: **Auto-refresh dashboards**
Fulfilled, the dashboards are automatically updated periodically, which is validated by applying the test dataset and waiting for the dashboard to automatically update. The refresh time is five minutes.

NF-9: **Backend Result Caching**
Fulfilled, Apache Superset is deployed with a built-in PostgreSQL for storage of metadata and Redis to cache results and data. The test of the auto-refresh dashboard shows that the results are cached.

7.4 Evaluation of the metrics

This final section of the evaluation, compares the metrics from Chapter 1 with the implementation. The following table shows the metric set out to implement, if it was implemented, the visualization used and if it is part of the subset developed for the main version of SCA Tool.

The Table 7.1 shows that all metrics are implemented in the initial version of the system. The reimplementation implements 15 of the 23 metrics for the main deployment of SCA Tool. In addition, the majority of requirements were fulfilled.

Metric Name	Implemented	Chart Type	Part of Reimplementation
1. User			
Users Over Time	Yes	Line Chart	Yes
Notifications per User	Yes	Line Chart	Yes
Uploads per User	Yes	Line Chart	Yes
Curation Activities per User	Yes	Line Chart	Yes
File Completions per User	Yes	Line Chart	Yes
2. Package			
Packages by License	Yes	Pie Chart	No
Package Scan Frequency	Yes	Calender Chart	No
Most Scanned Packages	Yes	Line Chart	No
Most Failed Scans	Yes	Line Chart	No
Most Successful Scans	Yes	Line Chart	No
3. Projects			
Projects Over Time	Yes	Line Chart	Yes
Version Units Over Time	Yes	Line Chart	Yes
Total Projects per Day	Yes	Line Chart	Yes
Total Version Units per Day	Yes	Calender Chart	Yes
Vulnerabilities per Version Unit	Yes	Line Chart	Yes
Components per Version Unit	Yes	Line Chart	Yes
Issues per Project	Yes	Line Chart	Yes
Total Issues per Day	Yes	Calender Chart	Yes
4. Organizations			
Organizations Over Time	Yes	Line Chart	Yes
Curated Packages per Org	Yes	Line Chart	Yes
Users per Organization	Yes	Line Chart	Yes
Scanner Jobs per Organization	Yes	Line Chart	No
API Keys per Organization	Yes	Line Chart	Yes

Table 7.1: Overview of Implemented Metrics

8 Conclusion

This chapter is the conclusion on the BI system build for SCA Tool. The chapter includes a final summary of the system that was built, the limitations of the system, and the future work.

8.1 Summary of the functionalities

The objective of this work was to design and implement a BI system capable of extracting, transforming, and visualizing the data from the production system of SCA Tool. The system successfully completes all five steps of the BI architecture. The data collection step is completed with the aggregation of the package data with the integration of the package_scan_stats_service. The data integration step is performed using Apache Hop ETL workflows and pipelines, which run independently from each other. The data storage step is fully integrated using a data warehouse built with PostgreSQL. The last two steps, the data processing and data presentation, are both achieved in Apache Superset, which uses the generated metrics to visualize them in charts and dashboards.

The system fulfills most of the requirements, including access control, error recovery, and deployment. The deployment is done with Docker Compose for local development and Kubernetes for the cluster, while maintaining one single source of truth.

8.2 Limitations

The main limitation is the concurrent development of the production system and the BI system. The major refactoring of the scanner leads to incompatibility of the full system with the production system. This is the reason for the integration of a subset of the functionalities of the BI system into the current development environment of SCA Tool.

Due to the concurrent development, the system was not used to process production business data. Therefore, no dashboards or charts were created on customer

data. This limits the evaluation of the system.

Further limitations are the lack of scalability, as currently only one PostgreSQL instance is used to store the BI data. The second bottleneck can be the ETL pipeline in case of a high demand. The same reason also limits the throughput and performance of the BI system. The ETL pipeline is especially performance-limited. Other limitations are the manual access management, which relies on manual user and access control, as well as error recovery, which is limited in the self-healing of the container state upon failure.

8.3 Future Work

The future work of the SCA Tool BI system is split into two different aspects.

System improvement The first part of future work is the resolving of the limitations. This includes creating a more scalable solution that is truly parallel in order to use several containers that run the ETL pipelines. This includes the introduction of dynamic horizontal scaling of containers to perform the data orchestration.

Secondly, the data aggregation can be implemented hourly instead of on a daily basis, with an analysis of whether the increase in computational demand justifies the information gain.

Thirdly, the access and role-based management of Apache Superset and the database can be integrated into the Ory Kratos access system to automate the access management. This makes the access management independent of manual interaction done by admin users

Finally, the testing of the system needs to be automated and integrated into a separate GitHub workflow in order to provide automated testing for every subsequent system change. In addition, more test cases need to be added.

System compatibility The integration of the BI system is the integration of a decision support system. This means that changes in the production system need to be reflected in the BI system. This needs to be done in order to verify and measure the usage of new features and understand the customer and changing customer needs.

The introduction of this work started with the quote: ‘To measure is to know’, which means that measuring is an ongoing task in order to keep knowing.

Appendices

A Vulnerability per Version Unit Pipeline

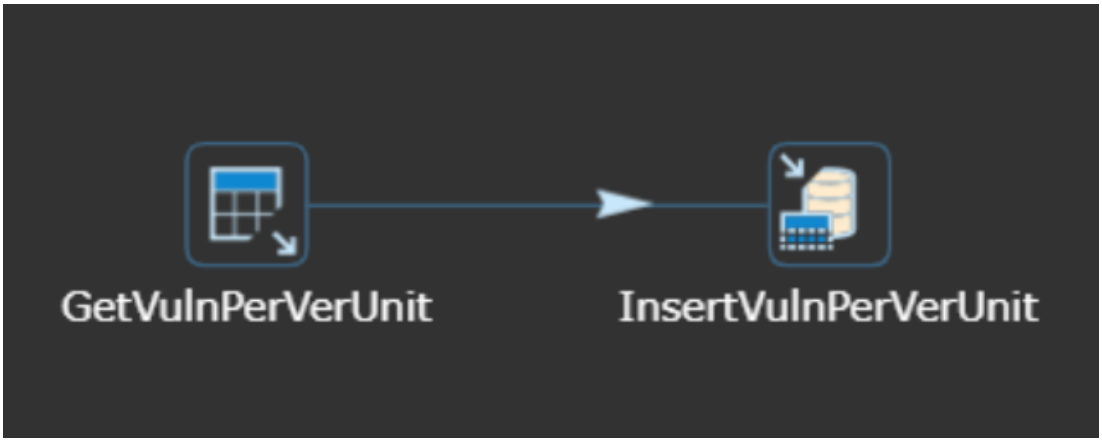

Figure 1: Vulnerabilities per Version Unit

B Hop init script

```
- sh
- -c
- |
  set -x
  # Wait for volumes to settle
  echo "Waiting 2 seconds for filesystem mount..."
  sleep 2
  cp -rL /configmap-projects/. /merged-projects/
  mkdir -p /merged-projects/metadata/rdbms
  cp /configmap-db/*.json /merged-projects/metadata/rdbms/
  if [ -n "${BI_DB_PASSWORD:-}" ]; then sed -i "s|\"password\"
   : \".*\"|\"password\": \"$BI_DB_PASSWORD\"|" /merged-
   projects/metadata/rdbms/scatool-bi.json; fi
  if [ -n "${BI_DB_USERNAME:-}" ]; then sed -i "s|\"username\"
   : \".*\"|\"username\": \"$BI_DB_USERNAME\"|" /merged-
   projects/metadata/rdbms/scatool-bi.json; fi
  if [ -n "${MAIN_DB_PASSWORD:-}" ]; then sed -i "s|\"password
   \": \".*\"|\"password\": \"$MAIN_DB_PASSWORD\"|" /merged-
   projects/metadata/rdbms/scatool-dev.json; fi
  if [ -n "${MAIN_DB_USERNAME:-}" ]; then sed -i "s|\"username
   \": \".*\"|\"username\": \"$MAIN_DB_USERNAME\"|" /merged-
   projects/metadata/rdbms/scatool-dev.json; fi
```

```
15   if [ -n "${ORY_DB_PASSWORD:-}" ]; then sed -i "s|\"password\
      ": \".*\"|\"password\": \"$ORY_DB_PASSWORD\"|" /merged-
      projects/metadata/rdbms/scatool-ory.json; fi
16   if [ -n "${ORY_DB_USERNAME:-}" ]; then sed -i "s|\"username\
      ": \".*\"|\"username\": \"$ORY_DB_USERNAME\"|" /merged-
      projects/metadata/rdbms/scatool-ory.json; fi
17   chown -R 501:501 /merged-projects
18   chmod -R u+rwX /merged-projects
```

Listing 1: Hop init script

C Superset init script

```
1  #!/bin/sh
2  set -eu
3  ADMIN_USERNAME="${SUPERSET_ADMIN_USERNAME:-{{ .Values.init.
      adminUser.username }}}"
4  ADMIN_PASSWORD="${SUPERSET_ADMIN_PASSWORD:-{{ .Values.init.
      adminUser.password }}}"
5  DASHBOARDS_SRC_DIR="${SUPERSET_DASHBOARDS_PATH:-{{ .Values.
      dashboards.mountPath }}}"
6  DASHBOARDS_WORK_DIR="${SUPERSET_DASHBOARDS_WORK_PATH:-/tmp/
      superset-dashboards}"
7  PYTHON_BIN="${VENV_PY:-/app/.venv/bin/python}"
8  if [ ! -x "$PYTHON_BIN" ]; then
9    PYTHON_BIN="python"
10 fi
11
12 echo "Upgrading DB schema..."
13 superset db upgrade
14 echo "Initializing roles..."
15 superset init
16 {{ if .Values.init.createAdmin }}
17 echo "Creating admin user..."
18 superset fab create-admin \
19   --username "$ADMIN_USERNAME" \
20   --firstname {{ .Values.init.adminUser.firstname }} \
21   --lastname {{ .Values.init.adminUser.lastname }} \
22   --email {{ .Values.init.adminUser.email }} \
23   --password "$ADMIN_PASSWORD" \
24   || true
25 {{- end }}
26 {{ if .Values.init.loadExamples }}
27 echo "Loading examples..."
28 superset load_examples
```

```
{{- end }}

if [ -d "$DASHBOARDS_SRC_DIR" ]; then
  echo "Preparing dashboard SQLAlchemy URIs..."
  rm -rf "$DASHBOARDS_WORK_DIR"
  mkdir -p "$DASHBOARDS_WORK_DIR"
  DASHBOARDS_IMPORT_LIST_FILE="/tmp/superset-dashboard-export-dirs.txt"
  rm -f "$DASHBOARDS_IMPORT_LIST_FILE"

  export DASHBOARDS_SRC_DIR DASHBOARDS_WORK_DIR DASHBOARDS_IMPORT_LIST_FILE

  "$PYTHON_BIN" - <<'PY'
import os
import json
import shutil
from pathlib import Path
from urllib.parse import quote_plus
import yaml


def copy_dashboard_exports():
    source_dir = Path(os.environ["DASHBOARDS_SRC_DIR"])
    target_dir = Path(os.environ["DASHBOARDS_WORK_DIR"])
    copied = 0

    for child in source_dir.iterdir():
        if child.name.startswith(".."):
            continue

        destination = target_dir / child.name
        if child.is_dir():
            shutil.copytree(child, destination, symlinks=False, dirs_exist_ok=True)
            copied += 1
        elif child.is_file():
            shutil.copy2(child, destination)
            copied += 1

    print(f"Copied {copied} top-level dashboard entries into {target_dir}")

def update_uri(file_name, user_env, password_env, host_env, port_env, db_env):
```

```
    username = os.getenv(user_env)
    password = os.getenv(password_env)
    host = os.getenv(host_env)
    port = os.getenv(port_env, "5432")
    database = os.getenv(db_env)
    if not all([username, password, host, database]):
        return

    sqlalchemy_uri = (
        "postgresql+psycopg2://"
        f"{quote_plus(username)}:{quote_plus(password)}@{host}:{port}/{database}"
    )
    for root, _, files in os.walk(os.environ["DASHBOARDS_WORK_DIR"]):
        for file in files:
            if file == file_name:
                path = Path(root) / file
                lines = path.read_text(encoding="utf-8").splitlines()
                replaced = False
                for index, line in enumerate(lines):
                    if line.startswith("sqlalchemy_uri:"):
                        lines[index] = f"sqlalchemy_uri: {sqlalchemy_uri}"
                        replaced = True
                        break
                if not replaced:
                    lines.insert(1, f"sqlalchemy_uri: {sqlalchemy_uri}")
                path.write_text("\n".join(lines) + "\n", encoding="utf-8")
                print(f"Updated URI in {file}")

def update_dataset_catalog(dataset_dir, catalog_env):
    catalog = os.getenv(catalog_env)
    if not catalog:
        return

    for root, _, files in os.walk(os.environ["DASHBOARDS_WORK_DIR"]):
        if dataset_dir not in root:
            continue
        for file in files:
            if not file.endswith(".yaml"):
```

```
                continue
            path = Path(root) / file
            payload = yaml.safe_load(path.read_text(encoding="utf-8")) or {}
            if isinstance(payload, dict) and "table_name" in payload:
                payload["catalog"] = catalog
                path.write_text(yaml.safe_dump(payload, sort_keys=False), encoding="utf-8")

copy_dashboard_exports()

print("\n--- PHASE 1: SCANNING FOR DATASETS ---")
dataset_uuids = set()
for root, _, files in os.walk(os.environ["DASHBOARDS_WORK_DIR"]):
    for file in files:
        if not file.endswith(".yaml"):
            continue
        path = Path(root) / file
        try:
            payload = yaml.safe_load(path.read_text(encoding="utf-8")) or {}
            if isinstance(payload, dict) and "table_name" in payload and "uuid" in payload:
                dataset_uuids.add(str(payload["uuid"]))
                print(f"Loaded Dataset: {file} (UUID: {payload['uuid']})")
        except Exception:
            pass
print(f"Total Datasets loaded: {len(dataset_uuids)}")

print("\n--- PHASE 2: PATCHING CHARTS ---")
removed_chart_uuids = set()
for root, _, files in os.walk(os.environ["DASHBOARDS_WORK_DIR"]):
    for file in files:
        if not file.endswith(".yaml"):
            continue
        path = Path(root) / file
        try:
            payload = yaml.safe_load(path.read_text(encoding="utf-8")) or {}
            if not isinstance(payload, dict) or "slice_name" not in payload:
```

```
                continue

            chart_uuid = payload.get("uuid")
            dataset_uuid = payload.get("dataset_uuid")

            if dataset_uuid and str(dataset_uuid) not in dataset_uuids:
                path.unlink()
                if chart_uuid:
                    removed_chart_uuids.add(str(chart_uuid))
                print(f"Removed orphaned chart: {file} (Missing UUID: {dataset_uuid})")
                continue

            if dataset_uuid:
                modified = False
                if "params" in payload and isinstance(payload["params"], dict):
                    if payload["params"].get("datasource") != f"{dataset_uuid}__table":
                        payload["params"]["datasource"] = f"{dataset_uuid}__table"
                        modified = True

                if "query_context" in payload and isinstance(payload["query_context"], str):
                    try:
                        qc = json.loads(payload["query_context"])
                        qc_modified = False
                        if "datasource" in qc and qc["datasource"].get("id") != dataset_uuid:
                            qc["datasource"]["id"] = str(dataset_uuid)
                            qc_modified = True
                        if "form_data" in qc and qc["form_data"].get("datasource") != f"{dataset_uuid}__table":
                            qc["form_data"]["datasource"] = f"{dataset_uuid}__table"
                            qc_modified = True
                        if qc_modified:
                            payload["query_context"] = json.dumps(qc)
                            modified = True
                    except json.JSONDecodeError:
```

```
                        pass

                if modified:
                    path.write_text(yaml.safe_dump(payload, sort_keys=False), encoding="utf-8")
                    print(f"Patched stale IDs in: {file}")
        except Exception as e:
            print(f"Error patching {file}: {e}")

print("\n--- PHASE 3: CLEANING DASHBOARDS ---")
if removed_chart_uuids:
    for root, _, files in os.walk(os.environ["DASHBOARDS_WORK_DIR"]):
        for file in files:
            if not file.endswith(".yaml"):
                continue
            path = Path(root) / file
            try:
                payload = yaml.safe_load(path.read_text(encoding="utf-8")) or {}
                if not isinstance(payload, dict) or "dashboard_title" not in payload:
                    continue

                position = payload.get("position")
                if not isinstance(position, dict):
                    continue

                removed_layout_ids = []
                for layout_id, node in list(position.items()):
                    if isinstance(node, dict) and node.get("type") == "CHART":
                        meta = node.get("meta") or {}
                        chart_uuid = meta.get("uuid")
                        if chart_uuid and str(chart_uuid) in removed_chart_uuids:
                            position.pop(layout_id, None)
                            removed_layout_ids.append(layout_id)
                            print(f"Removed dangling chart from dashboard: {file}")

                if removed_layout_ids:
                    removed_layout_ids_set = set(removed_layout_ids)
```

```
                    for node in position.values():
                        if isinstance(node, dict) and isinstance(node.get("children"), list):
                            node["children"] = [c for c in node["children"] if c not in removed_layout_ids_set]
                    path.write_text(yaml.safe_dump(payload, sort_keys=False), encoding="utf-8")
            except Exception:
                pass

update_uri("scatool-bi.yaml", "BI_DB_USERNAME", "BI_DB_PASSWORD", "BI_DB_HOST", "BI_DB_PORT", "BI_DB_NAME")
update_uri("scatool-dev.yaml", "MAIN_DB_USERNAME", "MAIN_DB_PASSWORD", "MAIN_DB_HOST", "MAIN_DB_PORT", "MAIN_DB_NAME")
update_uri("scatool-ory.yaml", "ORY_DB_USERNAME", "ORY_DB_PASSWORD", "ORY_DB_HOST", "ORY_DB_PORT", "ORY_DB_NAME")

update_dataset_catalog("scatool-bi", "BI_DB_NAME")
update_dataset_catalog("scatool-dev", "MAIN_DB_NAME")
update_dataset_catalog("scatool-ory", "ORY_DB_NAME")

export_dirs = []
for metadata_path in sorted(Path(os.environ["DASHBOARDS_WORK_DIR"]).glob("**/metadata.yaml")):
    if any(part.startswith("..") for part in metadata_path.parts):
        continue
    export_dirs.append(str(metadata_path.parent))

export_dirs = list(dict.fromkeys(export_dirs))
if export_dirs:
    print("\n--- PHASE 4: DISCOVERED DASHBOARD EXPORTS ---")
    for export_dir in export_dirs:
        print(f"Discovered export dir: {export_dir}")
    Path(os.environ["DASHBOARDS_IMPORT_LIST_FILE"]).write_text(
        "\n".join(export_dirs) + "\n",
        encoding="utf-8",
    )
else:
    print("No dashboard export directories with metadata.yaml were discovered")
PY
```

```
244
245   if [ -s "$DASHBOARDS_IMPORT_LIST_FILE" ]; then
246     while IFS= read -r export_dir; do
247       [ -n "$export_dir" ] || continue
248       echo "Importing dashboards from $export_dir..."
249       superset import-directory -o -f "$export_dir"
250     done < "$DASHBOARDS_IMPORT_LIST_FILE"
251   else
252     echo "No dashboard export directories discovered; skipping
      import."
253   fi
254 fi
```

Listing 2: Superset init script

D Scans per day

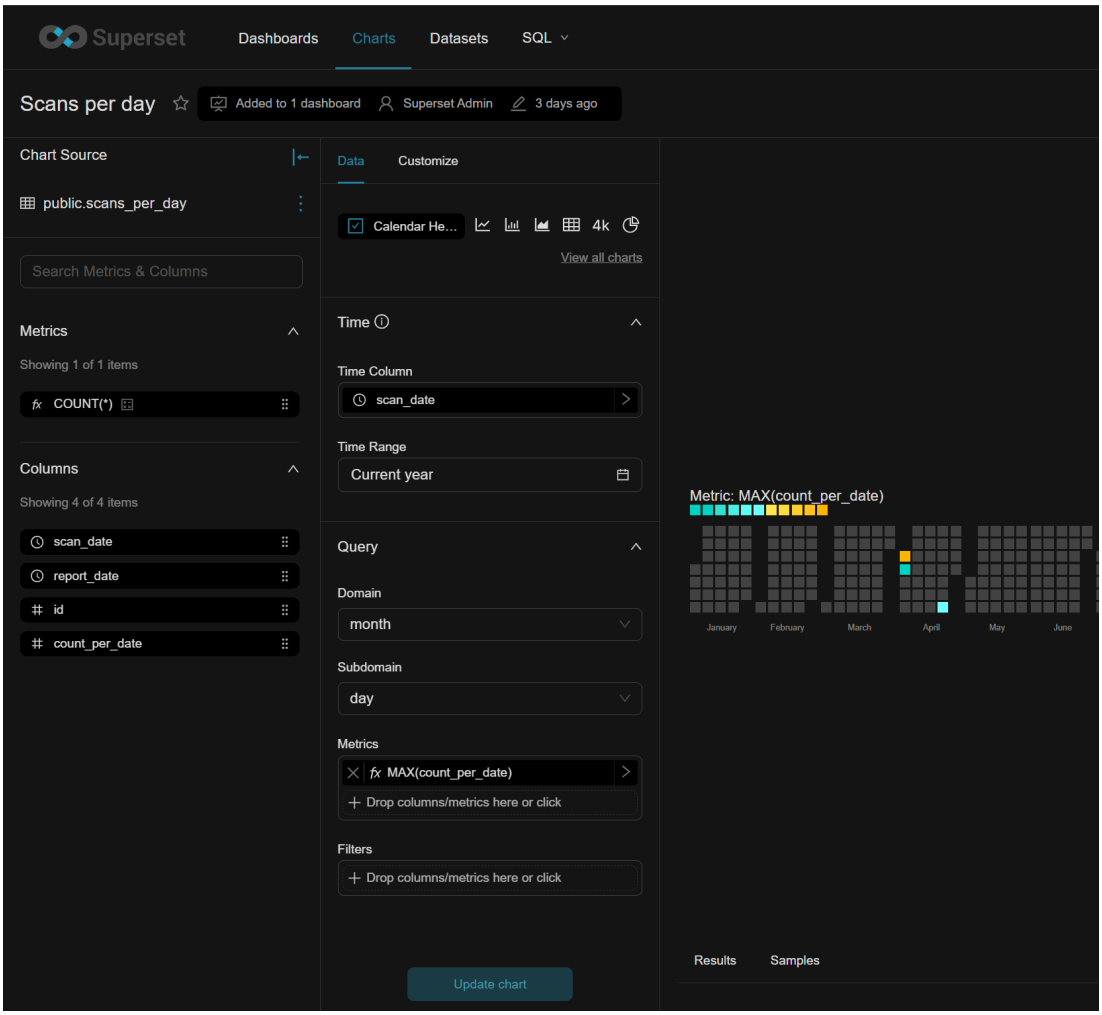

Figure 2: Scans per day

E Packages per license

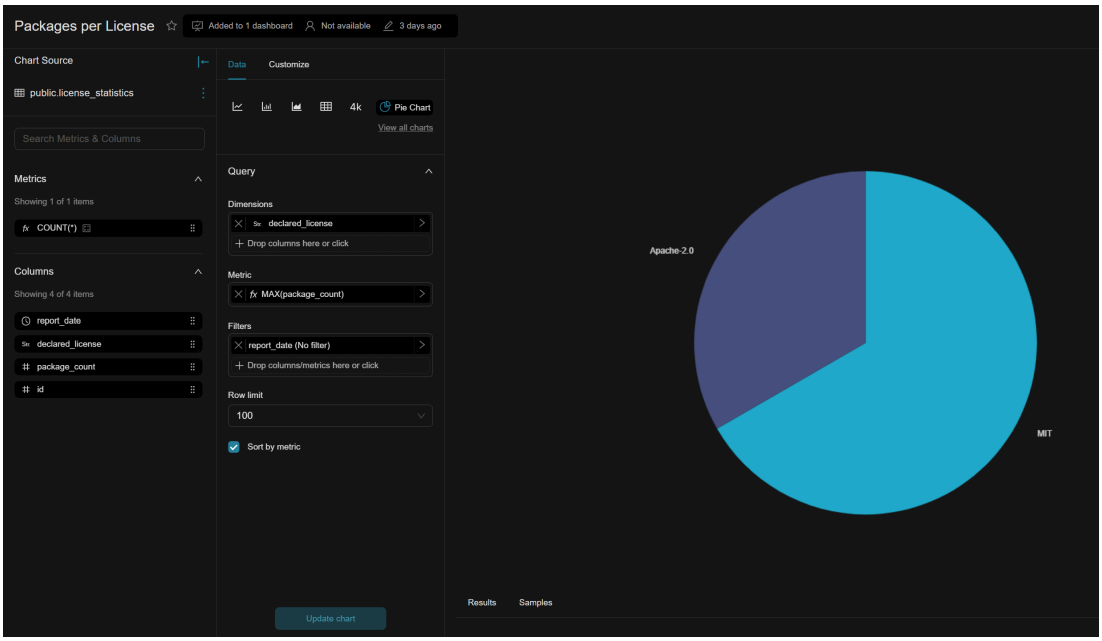

Figure 3: Packages per license

F User Dashboard

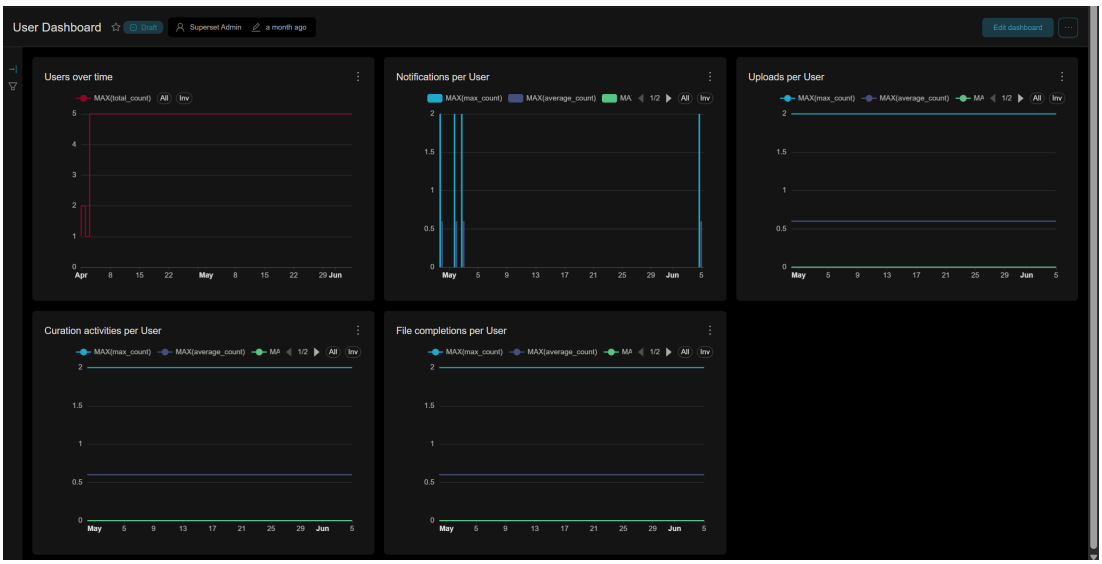

Figure 4: User Dashboard

G Projects Dashboard

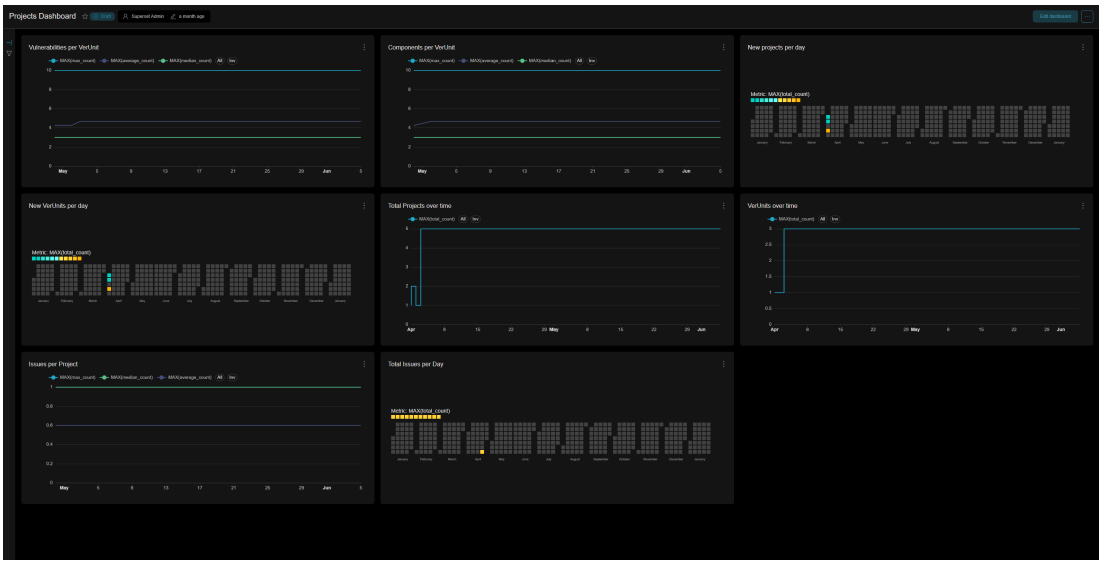

Figure 5: Project Dashboard

H Organization Dashboard

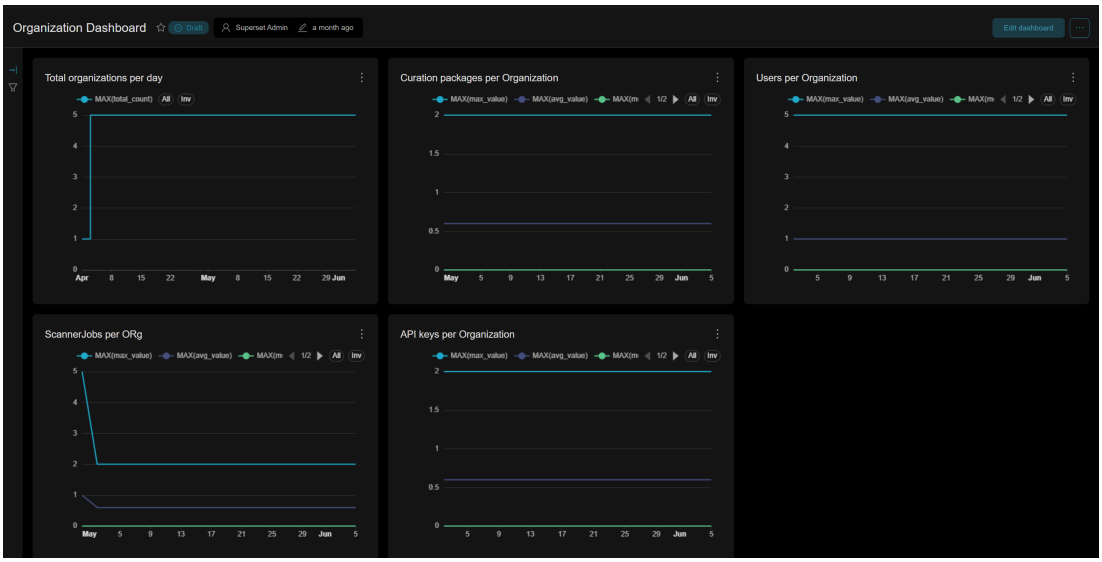

Figure 6: Organization Dashboard

I Validation Script

```
-- SQL script to insert sample data into the scatool-dev database.
-- Each row is a separate INSERT statement for easy modification.
-- This script repopulates the tables with a small dataset for validating Apache Hop
-- pipelines and for providing consistent development and test data.
--
-- EXPECTED-VALUE CONVENTIONS
-- 1. "Current" records are records whose deleted_at value is NULL.
-- 2. Per-parent statistics include seeded parent records with zero matching children.
-- 3. Daily arrow sequences cover every calendar day in the stated inclusive range.
-- 4. Values based on now() are marked as execution-date dependent.
-- 5. User statistics below treat Bob as deleted because deleted_at is populated,
--    even though his state value is still ACTIVE.

-- ============================================
-- Use TRUNCATE to clear existing data and reset sequences before inserting sample data.
-- ============================================
DO $$
  DECLARE
    r RECORD;
  BEGIN
    -- We filter out the flyway history table here
    FOR r IN (
      SELECT tablename
      FROM pg_tables
      WHERE schemaname = 'public'
        AND tablename NOT LIKE '%flyway%' -- Catches flyway_schema_history
    ) LOOP
        EXECUTE 'TRUNCATE TABLE ' || quote_ident(r.tablename) || ' RESTART IDENTITY CASCADE';
      END LOOP;
  END $$;

-- ============================================
-- CODEBASES (3)
```

```
-- Seeded codebases: 3
-- ========================================

INSERT INTO codebases (id, swhid) VALUES
('10000000-0000-4000-8000-000000000001', decode('abcdef1234567890abcdef1234567890abcdef1234567890abcdef', 'hex'));
INSERT INTO codebases (id, swhid) VALUES
('10000000-0000-4000-8000-000000000002', decode('bcdef1234567890abcdef1234567890abcdef1234567890abcdef1', 'hex'));

INSERT INTO codebases (id, swhid) VALUES
('10000000-0000-4000-8000-000000000003', decode('cdef1234567890abcdef1234567890abcdef1234567890abcdef12', 'hex'));

-- ========================================
-- ORGANIZATIONS (5)
-- Organizations over time, 2026-04-01 to 2026-04-04: 1 -> 1 -> 1 -> 5
-- New organizations per day:                          1 -> 0 -> 0 -> 4
-- Final organization count: 5
-- ========================================

INSERT INTO organizations (id, name, is_dummy_organisation, created_at, updated_at, features, color, slug, version) VALUES
('20000000-0000-4000-8000-000000000001', 'Org A', false, '2026-04-01 14:36:39.571584', now(), '{"scanning": true}', '#264653', 'org-a', 0);

INSERT INTO organizations (id, name, is_dummy_organisation, created_at, updated_at, features, color, slug, version) VALUES
('20000000-0000-4000-8000-000000000002', 'Org B', false, '2026-04-04 14:36:39.571584', now(), '{"scanning": true}', '#2a9d8f', 'org-b', 0);

INSERT INTO organizations (id, name, is_dummy_organisation, created_at, updated_at, features, color, slug, version) VALUES
('20000000-0000-4000-8000-000000000003', 'Org C', false, '2026-04-04 14:36:39.571584', now(), '{"scanning": true}', '
```

```
   #e76f51', 'org-c', 0);

INSERT INTO organizations (id, name, is_dummy_organisation,
   created_at, updated_at, features, color, slug, version)
   VALUES
('20000000-0000-4000-8000-000000000004', 'Org D', false, '
   2026-04-04 14:36:39.571584', now(), '{"scanning": true}', '
   #f4a261', 'org-d', 0);

INSERT INTO organizations (id, name, is_dummy_organisation,
   created_at, updated_at, features, color, slug, version)
   VALUES
('20000000-0000-4000-8000-000000000005', 'Org E', false, '
   2026-04-04 14:36:39.571584', now(), '{"scanning": true}', '
   #457b9d', 'org-e', 0);

-- ========================================
-- USERS (6)
-- Current users over time, 2026-04-01 to 2026-04-04: 1 -> 2
   -> 1 -> 5
-- New users per day:                                  1 -> 1
   -> 0 -> 4
-- Final current-user count: 5
-- Current users per organization: max 5, mean 1.00, median 0
-- ========================================

INSERT INTO users (id, organization_id, identity_id, email,
   name, created_at, updated_at, deleted_at, waitlisted_at,
   is_system_admin, state, state_changed_at) VALUES
('30000000-0000-4000-8000-000000000001', '
   20000000-0000-4000-8000-000000000001', '
   40000000-0000-4000-8000-000000000001', 'alice@org-a.com', '
   Alice', '2026-04-01 14:36:39.571584', now(), NULL, NULL,
   false, 'ACTIVE', now());

INSERT INTO users (id, organization_id, identity_id, email,
   name, created_at, updated_at, deleted_at, waitlisted_at,
   is_system_admin, state, state_changed_at) VALUES
('30000000-0000-4000-8000-000000000002', '
   20000000-0000-4000-8000-000000000001', '
   40000000-0000-4000-8000-000000000002', 'bob@org-a.com', '
   Bob', '2026-04-02 14:36:39.571584', now(), '2026-04-03
   14:36:39.571584', NULL, false, 'ACTIVE', now());

```

```
INSERT INTO users (id, organization_id, identity_id, email, name, created_at, updated_at, deleted_at, waitlisted_at, is_system_admin, state, state_changed_at) VALUES
('30000000-0000-4000-8000-000000000003', '20000000-0000-4000-8000-000000000001', '40000000-0000-4000-8000-000000000003', 'charlie@org-b.com', 'Charlie', '2026-04-04 14:36:39.571584', now(), NULL, NULL, false, 'ACTIVE', now());

INSERT INTO users (id, organization_id, identity_id, email, name, created_at, updated_at, deleted_at, waitlisted_at, is_system_admin, state, state_changed_at) VALUES
('30000000-0000-4000-8000-000000000004', '20000000-0000-4000-8000-000000000001', '40000000-0000-4000-8000-000000000004', 'dave@org-b.com', 'Dave', '2026-04-04 14:36:39.571584', now(), NULL, NULL, false, 'ACTIVE', now());

INSERT INTO users (id, organization_id, identity_id, email, name, created_at, updated_at, deleted_at, waitlisted_at, is_system_admin, state, state_changed_at) VALUES
('30000000-0000-4000-8000-000000000005', '20000000-0000-4000-8000-000000000001', '40000000-0000-4000-8000-000000000005', 'eve@org-c.com', 'Eve', '2026-04-04 14:36:39.571584', now(), NULL, NULL, false, 'ACTIVE', now());

INSERT INTO users (id, organization_id, identity_id, email, name, created_at, updated_at, deleted_at, waitlisted_at, is_system_admin, state, state_changed_at) VALUES
('30000000-0000-4000-8000-000000000006', '20000000-0000-4000-8000-000000000001', '40000000-0000-4000-8000-000000000006', 'frank@org-c.com', 'Frank', '2026-04-04 14:36:39.571584', now(), NULL, NULL, false, 'ACTIVE', now());

-- ========================================
-- FOLDERS (6 for projects)
-- Seeded folders: 6
-- ========================================

INSERT INTO folders (id, organization_id, name, created_at, updated_at, parent_id, version) VALUES
('50000000-0000-4000-8000-000000000001', '20000000-0000-4000-8000-000000000001', 'Folder A1', now(),
```

```
    now(), NULL, 0);
100
101 INSERT INTO folders (id, organization_id, name, created_at,
    updated_at, parent_id, version) VALUES
102 ('50000000-0000-4000-8000-000000000002', '
    20000000-0000-4000-8000-000000000002', 'Folder B1', now(),
    now(), NULL, 0);
103
104 INSERT INTO folders (id, organization_id, name, created_at,
    updated_at, parent_id, version) VALUES
105 ('50000000-0000-4000-8000-000000000003', '
    20000000-0000-4000-8000-000000000003', 'Folder C1', now(),
    now(), NULL, 0);
106
107 INSERT INTO folders (id, organization_id, name, created_at,
    updated_at, parent_id, version) VALUES
108 ('50000000-0000-4000-8000-000000000004', '
    20000000-0000-4000-8000-000000000004', 'Folder D1', now(),
    now(), NULL, 0);
109
110 INSERT INTO folders (id, organization_id, name, created_at,
    updated_at, parent_id, version) VALUES
111 ('50000000-0000-4000-8000-000000000005', '
    20000000-0000-4000-8000-000000000005', 'Folder E1', now(),
    now(), NULL, 0);
112
113 INSERT INTO folders (id, organization_id, name, created_at,
    updated_at, parent_id, version) VALUES
114 ('50000000-0000-4000-8000-000000000006', '
    20000000-0000-4000-8000-000000000005', 'Folder E2', now(),
    now(), NULL, 0);
115
116 -- ========================================
117 -- PROJECTS (6)
118 -- Current projects over time, 2026-04-01 to 2026-04-04: 1 ->
    2 -> 1 -> 5
119 -- New projects per day:                                  1 ->
     1 -> 0 -> 4
120 -- Final current-project count: 5
121 -- ========================================
122
123 INSERT INTO projects (id, folder_id, name, scope, created_at,
    updated_at, deleted_at, analysis_notifications) VALUES
124 ('60000000-0000-4000-8000-000000000001', '
    50000000-0000-4000-8000-000000000001', 'Project A1', '
```

```
    PRIVATE', '2026-04-01 14:36:39.571584', now(), NULL, false)
    ;
125
126 INSERT INTO projects (id, folder_id, name, scope, created_at,
    updated_at, deleted_at, analysis_notifications) VALUES
127 ('60000000-0000-4000-8000-000000000002', '
    50000000-0000-4000-8000-000000000001', 'Project A2', '
    PRIVATE', '2026-04-02 14:36:39.571584', now(), '2026-04-03
    14:36:39.571584', false);
128
129 INSERT INTO projects (id, folder_id, name, scope, created_at,
    updated_at, deleted_at, analysis_notifications) VALUES
130 ('60000000-0000-4000-8000-000000000003', '
    50000000-0000-4000-8000-000000000002', 'Project B1', '
    INTERNAL', '2026-04-04 14:36:39.571584', now(), NULL, false
    );
131
132 INSERT INTO projects (id, folder_id, name, scope, created_at,
    updated_at, deleted_at, analysis_notifications) VALUES
133 ('60000000-0000-4000-8000-000000000004', '
    50000000-0000-4000-8000-000000000002', 'Project B2', '
    INTERNAL', '2026-04-04 14:36:39.571584', now(), NULL, false
    );
134
135 INSERT INTO projects (id, folder_id, name, scope, created_at,
    updated_at, deleted_at, analysis_notifications) VALUES
136 ('60000000-0000-4000-8000-000000000005', '
    50000000-0000-4000-8000-000000000003', 'Project C1', '
    PUBLIC', '2026-04-04 14:36:39.571584', now(), NULL, false);
137
138 INSERT INTO projects (id, folder_id, name, scope, created_at,
    updated_at, deleted_at, analysis_notifications) VALUES
139 ('60000000-0000-4000-8000-000000000006', '
    50000000-0000-4000-8000-000000000003', 'Project C2', '
    PUBLIC', '2026-04-04 14:36:39.571584', now(), NULL, false);
140
141 -- ========================================
142 -- VER_UNITS (4)
143 -- Current version units over time, 2026-04-01 to 2026-04-04:
    1 -> 2 -> 1 -> 3
144 -- New version units per day:
     1 -> 1 -> 0 -> 2
145 -- Final current-version-unit count: 3
146 -- Components per current version unit: max 10, mean 4.67,
    median 3
```

```
147 -- Vulnerabilities per current version unit: max 10, mean
    4.67, median 3
148 -- ========================================
149
150 INSERT INTO ver_units (id, project_id, type, file_id, status,
    task_id, num_components, num_vulnerabilities, commit_hash,
    commit_message, commit_date, created_at, updated_at,
    deleted_at, archived_at, version, revision, upload_id, demo
    ) VALUES
151 ('70000000-0000-4000-8000-000000000001', '
    60000000-0000-4000-8000-000000000001', 0, NULL, 'COMPLETED'
    , NULL, 1, 1, 'abc123', 'Initial commit', now(), '
    2026-04-01 14:36:39.571584', now(), NULL, NULL, 1, 'main',
    NULL, false);
152
153 INSERT INTO ver_units (id, project_id, type, file_id, status,
    task_id, num_components, num_vulnerabilities, commit_hash,
    commit_message, commit_date, created_at, updated_at,
    deleted_at, archived_at, version, revision, upload_id, demo
    ) VALUES
154 ('70000000-0000-4000-8000-000000000002', '
    60000000-0000-4000-8000-000000000002', 0, NULL, 'COMPLETED'
    , NULL, 3, 3, 'def456', 'Second commit', now(), '2026-04-02
     14:36:39.571584', now(), '2026-04-03 14:36:39.571584',
    NULL, 2, 'develop', NULL, false);
155
156 INSERT INTO ver_units (id, project_id, type, file_id, status,
    task_id, num_components, num_vulnerabilities, commit_hash,
    commit_message, commit_date, created_at, updated_at,
    deleted_at, archived_at, version, revision, upload_id, demo
    ) VALUES
157 ('70000000-0000-4000-8000-000000000003', '
    60000000-0000-4000-8000-000000000003', 0, NULL, 'COMPLETED'
    , NULL, 10, 10, 'ghi789', 'Third commit', now(), '
    2026-04-04 14:36:39.571584', now(), NULL, NULL, 3, 'main',
    NULL, false);
158
159 INSERT INTO ver_units (id, project_id, type, file_id, status,
    task_id, num_components, num_vulnerabilities, commit_hash,
    commit_message, commit_date, created_at, updated_at,
    deleted_at, archived_at, version, revision, upload_id, demo
    ) VALUES
160   ('70000000-0000-4010-8000-000000000002', '
    60000000-0000-4000-8000-000000000002', 0, NULL, 'COMPLETED'
    , NULL, 3, 3, 'def456', 'Second commit', now(), '2026-04-04
```

```
     14:36:39.571584', now(), NULL, NULL, 2, 'develop', NULL,
    false);
161
162
163 -- ========================================
164 -- LICENSES (6)
165 -- Seeded licenses: 6
166 -- ========================================
167
168 INSERT INTO licenses (id, license_text_id, name, short_name,
    owner, homepage) VALUES
169 ('license-001', NULL, 'MIT License', 'MIT', 'Open Source
    Initiative', 'https://opensource.org/licenses/MIT');
170
171 INSERT INTO licenses (id, license_text_id, name, short_name,
    owner, homepage) VALUES
172 ('license-002', NULL, 'Apache License 2.0', 'Apache-2.0', '
    Apache Software Foundation', 'https://www.apache.org/
    licenses/LICENSE-2.0');
173
174 INSERT INTO licenses (id, license_text_id, name, short_name,
    owner, homepage) VALUES
175 ('license-003', NULL, 'BSD 3-Clause License', 'BSD-3-Clause',
    'Regents of the University of California', 'https://
    opensource.org/licenses/BSD-3-Clause');
176
177 INSERT INTO licenses (id, license_text_id, name, short_name,
    owner, homepage) VALUES
178 ('license-004', NULL, 'GNU GPL v3.0', 'GPL-3.0', 'Free
    Software Foundation', 'https://www.gnu.org/licenses/gpl
    -3.0.en.html');
179
180 INSERT INTO licenses (id, license_text_id, name, short_name,
    owner, homepage) VALUES
181 ('license-005', NULL, 'Mozilla Public License 2.0', 'MPL-2.0',
     'Mozilla Foundation', 'https://www.mozilla.org/en-US/MPL
    /2.0/');
182
183 INSERT INTO licenses (id, license_text_id, name, short_name,
    owner, homepage) VALUES
184 ('license-006', NULL, 'ISC License', 'ISC', 'Internet Systems
    Consortium', 'https://opensource.org/licenses/ISC');
185
186 -- ========================================
187 -- PACKAGES (3 for references)
```

```
-- Seeded package records: 3
-- Unique package names: 2
-- Packages per license: MIT 2, Apache-2.0 1
-- package scan frequency: 1, 1, 1 for the current day
-- ========================================

INSERT INTO packages (id, declared_license, checksums, source_artifact, vcs_info, name, cpe, description, homepage, authors, metadata_resolver, metadata_source, metadata_updated_at, declared_license_spdx, created_at, updated_at) VALUES
('pkg:npm/sample-pkg-1@1.0.0', 'MIT', '[]', NULL, NULL, 'sample-pkg-1', NULL, 'Sample package 1', 'https://example.com/pkg1', '{}', 'NPM', 'manual', now(), 'MIT', now(), now());

INSERT INTO packages (id, declared_license, checksums, source_artifact, vcs_info, name, cpe, description, homepage, authors, metadata_resolver, metadata_source, metadata_updated_at, declared_license_spdx, created_at, updated_at) VALUES
('pkg:npm/sample-pkg-1@3.0.0', 'Apache-2.0', '[]', NULL, NULL, 'sample-pkg-2', NULL, 'Sample package 2', 'https://example.com/pkg2', '{}', 'NPM', 'manual', now(), 'Apache-2.0', now(), now());

INSERT INTO packages (id, declared_license, checksums, source_artifact, vcs_info, name, cpe, description, homepage, authors, metadata_resolver, metadata_source, metadata_updated_at, declared_license_spdx, created_at, updated_at) VALUES
('pkg:npm/sample-pkg-1@2.0.0', 'MIT', '[]', NULL, NULL, 'sample-pkg-1', NULL, 'Sample package 1 v2', 'https://example.com/pkg1', '{}', 'NPM', 'manual', now(), 'MIT', now(), now());

-- ========================================
-- SCAN_JOBS (6)
-- Seeded scan jobs: 6
-- Status distribution: SUCCEEDED 3, FAILED 1, SKIPPED 1, CACHED 1
-- Discovered-license distribution: MIT 2, Apache-2.0 2, GPL-3.0
-- ========================================

```

```
210 INSERT INTO scan_jobs (id, status, package_id, scan_version,
       scancode_version, scancode_format_version,
       spdx_license_list_version, files_count, cached_count,
       scan_result_path, created_at, updated_at,
       discovered_license, quality, version, scanner_job_id,
       cache_key, codebase_id, scan_input) VALUES
211 ('80000000-0000-4000-8000-000000000001', 'SUCCEEDED', 'pkg:npm
       /sample-pkg-1@1.0.0', 1, '32.0.0', '2.0', '3.24', 100, 0, '
       /path/to/result1.json', '2026-04-01 14:36:39.571584', now()
       , 'MIT', 80, 0, NULL, 'cache-key-001', NULL, NULL);
212
213 INSERT INTO scan_jobs (id, status, package_id, scan_version,
       scancode_version, scancode_format_version,
       spdx_license_list_version, files_count, cached_count,
       scan_result_path, created_at, updated_at,
       discovered_license, quality, version, scanner_job_id,
       cache_key, codebase_id, scan_input) VALUES
214 ('80000000-0000-4000-8000-000000000002', 'FAILED', 'pkg:npm/
       sample-pkg-1@1.0.0', 1, '32.0.0', '2.0', '3.24', 100, 0, '/
       path/to/result2.json', '2026-04-01 14:36:39.571584', now(),
        'Apache-2.0', 80, 0, NULL, 'cache-key-002', NULL, NULL);
215
216 INSERT INTO scan_jobs (id, status, package_id, scan_version,
       scancode_version, scancode_format_version,
       spdx_license_list_version, files_count, cached_count,
       scan_result_path, created_at, updated_at,
       discovered_license, quality, version, scanner_job_id,
       cache_key, codebase_id, scan_input) VALUES
217 ('80000000-0000-4000-8000-000000000003', 'SUCCEEDED', 'pkg:npm
       /sample-pkg-1@1.0.0', 1, '32.0.0', '2.0', '3.24', 100, 0, '
       /path/to/result3.json', '2026-04-01 14:36:39.571584', now()
       , 'GPL-3.0', 80, 0, NULL, 'cache-key-003', NULL, NULL);
218
219 INSERT INTO scan_jobs (id, status, package_id, scan_version,
       scancode_version, scancode_format_version,
       spdx_license_list_version, files_count, cached_count,
       scan_result_path, created_at, updated_at,
       discovered_license, quality, version, scanner_job_id,
       cache_key, codebase_id, scan_input) VALUES
220 ('80000000-0000-4000-8000-000000000004', 'SKIPPED', 'pkg:npm/
       sample-pkg-1@1.0.0', 1, '32.0.0', '2.0', '3.24', 100, 0, '/
       path/to/result4.json', '2026-04-02 14:36:39.571584', now(),
        'MIT', 80, 0, NULL, 'cache-key-004', NULL, NULL);
221
```

```
INSERT INTO scan_jobs (id, status, package_id, scan_version,
    scancode_version, scancode_format_version,
    spdx_license_list_version, files_count, cached_count,
    scan_result_path, created_at, updated_at,
    discovered_license, quality, version, scanner_job_id,
    cache_key, codebase_id, scan_input) VALUES
('80000000-0000-4000-8000-000000000005', 'CACHED', 'pkg:npm/
    sample-pkg-1@1.0.0', 1, '32.0.0', '2.0', '3.24', 100, 100,
    '/path/to/result5.json', now(), '2026-04-02 14:36:39.571584
    ', 'Apache-2.0', 80, 0, NULL, 'cache-key-005', NULL, NULL);

INSERT INTO scan_jobs (id, status, package_id, scan_version,
    scancode_version, scancode_format_version,
    spdx_license_list_version, files_count, cached_count,
    scan_result_path, created_at, updated_at,
    discovered_license, quality, version, scanner_job_id,
    cache_key, codebase_id, scan_input) VALUES
('80000000-0000-4000-8000-000000000006', 'SUCCEEDED', 'pkg:npm
    /sample-pkg-1@1.0.0', 1, '32.0.0', '2.0', '3.24', 100, 0, '
    /path/to/result6.json', now(), '2026-04-02 14:36:39.571584'
    , 'GPL-3.0', 80, 0, NULL, 'cache-key-006', NULL, NULL);

-- ========================================
-- SCANNER_JOBS (3)
-- scanner jobs per organization: max 2, mean 0.60, median 0
-- ========================================

INSERT INTO scanner_jobs (id, status, priority, created_at,
    updated_at, organization_id, ver_unit_id, version) VALUES
('90000000-0000-4000-8000-000000000001', 'COMPLETED', 1, now()
    , now(), '20000000-0000-4000-8000-000000000001', '
    70000000-0000-4000-8000-000000000001', 0);

INSERT INTO scanner_jobs (id, status, priority, created_at,
    updated_at, organization_id, ver_unit_id, version) VALUES
('90000000-0000-4000-8000-000000000002', 'COMPLETED', 2, now()
    , now(), '20000000-0000-4000-8000-000000000001', '
    70000000-0000-4000-8000-000000000002', 0);

INSERT INTO scanner_jobs (id, status, priority, created_at,
    updated_at, organization_id, ver_unit_id, version) VALUES
('90000000-0000-4000-8000-000000000003', 'RUNNING', 3, now(),
    now(), '20000000-0000-4000-8000-000000000002', '
    70000000-0000-4000-8000-000000000003', 0);

```

```
242 -- ========================================
243 -- CURATION_ACTIVITIES (3)
244 -- Seeded curation activities: 3
245 -- Activities per current user: max 2, mean 0.60, median 0
246 -- ========================================
247
248 INSERT INTO curation_activities (id, activity_type,
       organization_id, codebase_id, user_id, created_at) VALUES
249 ('a0000000-0000-4000-8000-000000000001', 'FILE_COMPLETION', '
       20000000-0000-4000-8000-000000000001', '
       10000000-0000-4000-8000-000000000001', '
       30000000-0000-4000-8000-000000000001', now());
250
251 INSERT INTO curation_activities (id, activity_type,
       organization_id, codebase_id, user_id, created_at) VALUES
252 ('a0000000-0000-4000-8000-000000000002', 'FINDING_CURATION', '
       20000000-0000-4000-8000-000000000001', '
       10000000-0000-4000-8000-000000000002', '
       30000000-0000-4000-8000-000000000001', now());
253
254 INSERT INTO curation_activities (id, activity_type,
       organization_id, codebase_id, user_id, created_at) VALUES
255 ('a0000000-0000-4000-8000-000000000003', 'BULK_ACTION', '
       20000000-0000-4000-8000-000000000003', '
       10000000-0000-4000-8000-000000000003', '
       30000000-0000-4000-8000-000000000003', now());
256
257 -- ========================================
258 -- FILE_COMPLETIONS (3) - with valid codebase_id values
259 -- File completions per current user: max 2, mean 0.60, median
        00
260 -- ========================================
261
262 INSERT INTO file_completions (id, organization_id, codebase_id
       , package_id, file_path, file_sha256, scope, status, dirty,
        license, user_id, created_at, updated_at) VALUES
263 ('b0000000-0000-4000-8000-000000000001', '
       20000000-0000-4000-8000-000000000001', '
       10000000-0000-4000-8000-000000000001', 'pkg:npm/sample-pkg
       -1@1.0.0', '/src/index.js', 'sha256-file1', 'SAME_PACKAGE',
        'REVIEWED', false, 'MIT', '
       30000000-0000-4000-8000-000000000001', now(), now());
264
265 INSERT INTO file_completions (id, organization_id, codebase_id
       , package_id, file_path, file_sha256, scope, status, dirty,
```

```
    license, user_id, created_at, updated_at) VALUES
('b0000000-0000-4000-8000-000000000002', '
   20000000-0000-4000-8000-000000000002', '
   10000000-0000-4000-8000-000000000002', 'pkg:npm/sample-pkg
   -2@1.0.0', '/src/app.js', 'sha256-file2', 'SAME_PACKAGE', '
   EXCLUDED', false, 'Apache-2.0', '
   30000000-0000-4000-8000-000000000001', now(), now());

INSERT INTO file_completions (id, organization_id, codebase_id
   , package_id, file_path, file_sha256, scope, status, dirty,
    license, user_id, created_at, updated_at) VALUES
('b0000000-0000-4000-8000-000000000003', '
   20000000-0000-4000-8000-000000000003', '
   10000000-0000-4000-8000-000000000003', 'pkg:maven/com.
   example/sample-pkg-3@1.0.0', '/src/Main.java', 'sha256-
   file3', 'SAME_ORGANIZATION', 'BLIND_ACCEPTED', false, 'GPL
   -3.0', '30000000-0000-4000-8000-000000000003', now(), now()
   );

-- ========================================
-- ISSUES (3) - using valid UUIDs starting with 'c'
-- Issues per project: max 1, mean 0.60, median 0
-- Issues per organization: max 1, mean 0.60, median 0
-- ========================================

INSERT INTO issues (id, code, labels, organization_id,
   project_id, version_id, scope, severity, status, created_at
   , updated_at) VALUES
('c0000000-0000-4000-8000-000000000001', 'ISS-001', '{"
   security"}', '20000000-0000-4000-8000-000000000001', '
   60000000-0000-4000-8000-000000000001', '
   70000000-0000-4000-8000-000000000001', 'SECURITY', 'HIGH',
   'UNRESOLVED', now(), now());

INSERT INTO issues (id, code, labels, organization_id,
   project_id, version_id, scope, severity, status, created_at
   , updated_at) VALUES
('c0000000-0000-4000-8000-000000000002', 'ISS-002', '{"license
   "}', '20000000-0000-4000-8000-000000000002', '
   60000000-0000-4000-8000-000000000003', '
   70000000-0000-4000-8000-000000000001', 'LICENSE', 'MEDIUM',
    'RESOLVED', now(), now());

INSERT INTO issues (id, code, labels, organization_id,
   project_id, version_id, scope, severity, status, created_at
```

```
    , updated_at) VALUES
('c0000000-0000-4000-8000-000000000003', 'ISS-003', '{"quality
    "}', '20000000-0000-4000-8000-000000000003', '
    60000000-0000-4000-8000-000000000005', '
    70000000-0000-4000-8000-000000000002', 'QUALITY', 'LOW', '
    UNRESOLVED', now(), now());

-- ========================================
-- NOTIFICATIONS (3) - using valid UUIDs starting with 'd'
-- Notifications per user: max 2, mean 0.60, median 0
-- ========================================

INSERT INTO notifications (id, user_id, is_read, status, title
    , message, link, archived_at, created_at, updated_at)
    VALUES
('d0000000-0000-4000-8000-000000000001', '
    30000000-0000-4000-8000-000000000001', false, 'INFO', 'Scan
     Completed', 'Your scan has completed successfully.', '/
    projects/1/versions/1', NULL, now(), now());

INSERT INTO notifications (id, user_id, is_read, status, title
    , message, link, archived_at, created_at, updated_at)
    VALUES
('d0000000-0000-4000-8000-000000000002', '
    30000000-0000-4000-8000-000000000001', false, 'WARNING', '
    License Issue', 'A license issue was detected in your
    project.', '/projects/3/versions/2', NULL, now(), now());

INSERT INTO notifications (id, user_id, is_read, status, title
    , message, link, archived_at, created_at, updated_at)
    VALUES
('d0000000-0000-4000-8000-000000000003', '
    30000000-0000-4000-8000-000000000003', true, 'SUCCESS', '
    Curation Complete', 'File curation has completed.', '/
    projects/5/versions/3', now(), now(), now());

-- ========================================
-- PACKAGE_SCAN_STATS (3)
-- Overall totals: succeeded 23, failed 6, skipped 3, cached 6
-- Most successful package: sample-pkg-1@1.0.0 (10 succeeded
    scans)
-- Most failed package: sample-pkg-1@3.0.0 (3 failed scans)
-- Most scanned without version: sample-pkg-1 (16 total scans)
-- ========================================

```

```
308 INSERT INTO package_scan_stats (id, name, succeeded_scans,
       failed_scans, skipped_scans, cached_scans) VALUES
309 ('pkg:npm/sample-pkg-1@1.0.0', 'sample-pkg-1', 10, 2, 1, 3);
310
311 INSERT INTO package_scan_stats (id, name, succeeded_scans,
       failed_scans, skipped_scans, cached_scans) VALUES
312 ('pkg:npm/sample-pkg-1@2.0.0', 'sample-pkg-2', 8, 1, 2, 2);
313
314 INSERT INTO package_scan_stats (id, name, succeeded_scans,
       failed_scans, skipped_scans, cached_scans) VALUES
315 ('pkg:npm/sample-pkg-1@3.0.0', 'sample-pkg-3', 5, 3, 0, 1);
316
317 -- ========================================
318 -- UPLOADS (3) - using valid UUIDs starting with 'e'
319 -- Uploads per user: max 2, mean 0.60, median 0
320 -- ========================================
321
322 INSERT INTO uploads (id, user_id, file_id, created_at,
       expires_at) VALUES
323 ('e0000000-0000-4000-8000-000000000001', '
       30000000-0000-4000-8000-000000000001', 'uploads/user1/file1
       .zip', now(), now() + interval '1 day');
324
325 INSERT INTO uploads (id, user_id, file_id, created_at,
       expires_at) VALUES
326 ('e0000000-0000-4000-8000-000000000002', '
       30000000-0000-4000-8000-000000000001', 'uploads/user3/file2
       .zip', now(), now() + interval '1 day');
327
328 INSERT INTO uploads (id, user_id, file_id, created_at,
       expires_at) VALUES
329 ('e0000000-0000-4000-8000-000000000003', '
       30000000-0000-4000-8000-000000000003', 'uploads/user5/file3
       .zip', now(), now() + interval '1 day');
330
331 -- ========================================
332 -- CURATED_PACKAGES (3) - using valid UUIDs starting with 'f'
333 -- Curated packages per organization: max 2, mean 0.60, median
        0
334 -- ========================================
335
336 INSERT INTO curated_packages (id, level, package_id,
       organization_id, ver_unit_id, name, declared_license, cpe,
       description, homepage, authors, download_url, vcs_info)
       VALUES
```

```
('f0000000-0000-4000-8000-000000000001', 'HIGH', 'pkg:npm/
    sample-pkg-1@1.0.0', '20000000-0000-4000-8000-000000000001'
    , NULL, 'sample-pkg-1', 'MIT', NULL, 'Sample package 1', '
    https://example.com/pkg1', '{}', NULL, NULL);

INSERT INTO curated_packages (id, level, package_id,
    organization_id, ver_unit_id, name, declared_license, cpe,
    description, homepage, authors, download_url, vcs_info)
    VALUES
('f0000000-0000-4000-8000-000000000002', 'MEDIUM', 'pkg:npm/
    sample-pkg-1@2.0.0', '20000000-0000-4000-8000-000000000001'
    , NULL, 'sample-pkg-2', 'Apache-2.0', NULL, 'Sample package
     2', 'https://example.com/pkg2', '{}', NULL, NULL);

INSERT INTO curated_packages (id, level, package_id,
    organization_id, ver_unit_id, name, declared_license, cpe,
    description, homepage, authors, download_url, vcs_info)
    VALUES
('f0000000-0000-4000-8000-000000000003', 'LOW', 'pkg:npm/
    sample-pkg-1@1.0.0', '20000000-0000-4000-8000-000000000003'
    , NULL, 'sample-pkg-3', 'GPL-3.0', NULL, 'Sample package 3'
    , 'https://example.com/pkg3', '{}', NULL, NULL);

-- ========================================
-- API_KEYS (3) - using valid UUIDs starting with 'a'
-- Active API keys per organization: max 2, mean 0.60, median
    0
-- ========================================

INSERT INTO api_keys (api_key, id, organization_id, name,
    is_active, created_at, last_used_at, expiration_date)
    VALUES
('key-001', 'a0000000-0000-4000-8000-000000000011', '
    20000000-0000-4000-8000-000000000001', 'API Key 1', true,
    now(), NULL, '2030-04-26');

INSERT INTO api_keys (api_key, id, organization_id, name,
    is_active, created_at, last_used_at, expiration_date)
    VALUES
('key-002', 'a0000000-0000-4000-8000-000000000012', '
    20000000-0000-4000-8000-000000000001', 'API Key 2', true,
    now(), NULL, '2030-04-26');

INSERT INTO api_keys (api_key, id, organization_id, name,
    is_active, created_at, last_used_at, expiration_date)
```

```
    VALUES
('key-003', 'a0000000-0000-4000-8000-000000000013', '
    20000000-0000-4000-8000-000000000003', 'API Key 3', false,
    now(), now(), '2030-12-31');
```

Listing 3: Validation Script

References

Anggrainy, T. D., & Sari, A. R. (2022). Implementation of extract, transform, load on data warehouse and business intelligence using pentaho and tableau to analyse sales performance of offlist store. *International Research Journal of Advanced Engineering and Science*, *7*(2), 368–374.

Azeroual, O., & Theel, H. (2018). The Effects of Using Business Intelligence Systems on an Excellence Management and Decision-Making Process by Start-Up Companies: A Case Study. *INTERNATIONAL JOURNAL OF MANAGEMENT SCIENCE AND BUSINESS ADMINISTRATION*, *4*(3), 30–40. https://doi.org/10.18775/ijmsba.1849-5664-5419.2014.43.1004

Bansal, S. (2014). Towards a Semantic Extract-Transform-Load (ETL) Framework for Big Data Integration. *2014 IEEE International Congress on Big Data*, 522–529. https://doi.org/10.1109/BigData.Congress.2014.82

Becker, C., Kraxner, M., Plangg, M., & Rauber, A. (2013). Improving Decision Support for Software Component Selection through Systematic Cross-Referencing and Analysis of Multiple Decision Criteria [ISSN: 1530-1605]. *2013 46th Hawaii International Conference on System Sciences*, 1193–1202. https://doi.org/10.1109/HICSS.2013.263

Chumpitaz-Huarcaya, J. G., & Auccahuasi, W. (2026). Master Data Architecture for Clinical Process Improvement. *2026 International Conference on AI-Driven Smart Systems and Ubiquitous Computing (ICAUC)*, 2102–2107. https://doi.org/10.1109/ICAUC68182.2026.11441250

Conn, S. (2005). OLTP and OLAP Data Integration: A Review of Feasible Implementation Methods and Architectures for Real Time Data Analysis. *Proceedings. IEEE SoutheastCon, 2005.*, 515–520. https://doi.org/10.1109/SECON.2005.1423297

Conrad, A. (2021). Database of the Year: Postgres. *IEEE Software*, *38*(5), 130–132. https://doi.org/10.1109/MS.2021.3089730

Eberhard, K. (2023). The effects of visualization on judgment and decision-making: A systematic literature review. *Management Review Quarterly*, *73*(1), 167–214. https://doi.org/10.1007/s11301-021-00235-8

Gücük, G.-L., Simic, D., Leible, S., Lewandowski, T., & Kučević, E. (2023). Enhancing educational insights: A real-time data analytics stack for project-based learning. https://doi.org/10.18420/inf2023_196

Han, J., Kamber, M., & Pei, J. (2012). *Data mining: Concepts and techniques* (3rd). Morgan Kaufmann.

Huang, Z.-x., Savita, K. S., & Zhong-jie, J. (2022). The Business Intelligence impact on the financial performance of start-ups. *Information Processing & Management*, *59*(1), 102761. https://doi.org/10.1016/j.ipm.2021.102761

Luhn, H. P. (1958). A Business Intelligence System. *IBM Journal of Research and Development*, *2*(4), 314–319. https://doi.org/10.1147/rd.24.0314

Machado, G. V., Cunha, Í., Pereira, A. C., & Oliveira, L. B. (2019). Dod-etl: Distributed on-demand etl for near real-time business intelligence. *Journal of Internet Services and Applications*, *10*(1), 21.

Mahmud, D., & Ikbal, M. Z. (2022). THE ROLE OF ETL (EXTRACT-TRANSFORM-LOAD) PIPELINES IN SCALABLE BUSINESS INTELLIGENCE: A COMPARATIVE STUDY OF DATA INTEGRATION TOOLS. *ASRC Procedia: Global Perspectives in Science and Scholarship*, *2*(1), 89–121. https://doi.org/10.63125/1spa6877

Mohamed, A., Ruhe, G., & Eberlein, A. (2007). Cots selection: Past, present, and future. *14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)*, 103–114.

Molensky, L., Ketter, W., Collins, J., Bloemhof, J., & van de Koppel, H. (2010). Business intelligence gap analysis: A user, supplier and academic perspective. *Proceedings of the 12th International Conference on Electronic Commerce: Roadmap for the Future of Electronic Business*, 119–128. https://doi.org/10.1145/2389376.2389393

Papp, H., & Hanussek, M. (2020, December). A Methodology for Deriving Evaluation Criteria for Software Solutions [arXiv:2012.01023 [cs]]. https://doi.org/10.48550/arXiv.2012.01023

Pereira, J. L., & Costa, M. (2016). Decision support in big data contexts: A business intelligence solution. In *New advances in information systems and technologies* (pp. 983–992). Springer.

Ramadhani, P. P., Hadi, S., & Rosadi, R. (2021). Implementation of Data Warehouse in Making Business Intelligence Dashboard Development Using PostgreSQL Database and Kimball Lifecycle Method. *2021 International Conference on Artificial Intelligence and Big Data Analytics*, 88–92. https://doi.org/10.1109/ICAIBDA53487.2021.9689697

Sharma, H., Arya, P. K., & Agarwal, A. (2024). Business Intelligence Systems' Effect on Start-Up Companies' Decision-Making and Excellence Management Processes. *2024 IEEE 1st Karachi Section Humanitarian Technology Conference (KHI-HTC)*, 1–6. https://doi.org/10.1109/KHI-HTC60760.2024.10481916

Soares, G. H., & Brito, M. A. (2023). Business Intelligence Over and Above Apache Superset [ISSN: 2166-0727]. *2023 18th Iberian Conference on Information Systems and Technologies (CISTI)*, 1–6. https://doi.org/10.23919/CISTI58278.2023.10211907

Villafanes Lozano, M. (2026). Buenas prácticas para la implantación de un sistema bi académico open source.

Zaidan, A. A., Zaidan, B. B., Hussain, M., Haiqi, A., Mat Kiah, M. L., & Abdulnabi, M. (2015). Multi-criteria analysis for OS-EMR software selection problem: A comparative study. *Decision Support Systems*, *78*, 15–27. https://doi.org/10.1016/j.dss.2015.07.002