Pedro David
Lopes Matos

# Desenvolvimento de um pipeline para extração e mapeamento de termos biomédicos

# Development of a pipeline for the extraction and mapping of biomedical terms

Universidade de Aveiro
2021

Pedro David
Lopes Matos

# Desenvolvimento de um pipeline para extração e mapeamento de termos biomédicos

# Development of a pipeline for the extraction and mapping of biomedical terms

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Informática, realizada sob a orientação científica do Doutor José Luís Guimarães Oliveira, Professor catedrático do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro.

**o júri / the jury**

presidente / president

Prof. Doutor Luís Filipe de Seabra Lopes
professor associado da Universidade de Aveiro

vogais / examiners committee

Prof. Doutora Paula Alexandra Gomes da Silva
professora auxiliar da Faculdade de Ciências e Tecnologias da Universidade de Coimbra

Prof. Doutor José Luís Guimarães Oliveira
professor catedrático da Universidade de Aveiro (orientador)

**agradecimentos / acknowledgements**

Com o terminar de mais uma etapa no meu percurso académico, não o poderia fazer sem deixar algumas palavras de agradecimento às pessoas que dele fizeram parte.

Em primeiro lugar, à minha família, em especial aos meus pais e à minha avó, por estarem presentes e por todo o apoio que me deram durante todos estes anos e por me terem proporcionado as condições financeiras para poder enriquecer a minha formação.

Gostaria também de agradecer aos meus amigos e a todos os que tive a oportunidade de conhecer nos últimos anos por terem sido os pilares de apoio nos bons e maus instantes e por terem partilhado comigo os momentos de descontração.

Por último, mas não menos importante, ao Professor Doutor José Luís Oliveira e ao João Almeida pela excelente orientação e acompanhamento durante o desenvolvimento desta dissertação.

**Palavras Chave**

Registos eletrónicos de saúde, modelo comum de dados, harmonização de dados, desenvolvimento de aplicações web

**Resumo**

Desde a sua origem, há algumas décadas, os registos eletrónicos de saúde têm sido usados para melhorar os processos associados aos cuidados de saúde. Por outro lado, muito investigadores têm vindo a utilizar este tipo de dados para realizar estudos observacionais, de modo a enriquecer o conhecimento na área, por exemplo, prevendo se um conjunto de características fisiológicas e/ou genéticas nos torna mais vulneráveis a determinadas doenças.

Contudo, para que os resultados obtidos nestes estudos sejam considerados relevantes, é necessário recolher e tratar múltiplos dados, recolhidos de diversas fontes, hospitais ou unidades médicas. Apenas desta forma será possível prevenir o enviesamento de dados. No entanto, a falta de harmonização de dados e de utilização de normas entre estes sistemas cria uma barreira durante a condução de estudos observacionais. Uma possível solução é mapear os termos contidos numa base de dados de registos médicos para um modelo comum de dados. Este tem sido o objetivo principal da Observational Health Data Sciences and Informatics (OHDSI), uma rede internacional de investigadores.

A aplicação Rabbit in a Web, apresentada nesta dissertação, foi desenvolvida para facilitar o processo de mapeamento, criando um ambiente colaborativo onde vários utilizadores podem participar na especificação das relações entre os dados. A aplicação segue um modelo cliente-servidor com uma arquitetura dividida em três camadas e foca-se sobretudo na usabilidade, criando uma interface visual que utiliza várias tecnologias web modernas. O servidor foi desenvolvido usando a *framework* Spring Boot ao passo que do lado do cliente, foi usado ReactJS para o *frontend* da aplicação.

**Keywords**

Electronic health records, common data model, data harmonization, full-stack web development

**Abstract**

Since its conception, some decades ago, Electronic Health Records (EHRs) have been used to improve healthcare procedures. On the other hand, many researchers have been using this data to conduct observational studies in order to enhance the knowledge in the medical area, predicting whether a set of physiological and/or genetic characteristics makes us vulnerable to certain diseases.

Nevertheless, it is necessary to collect and process multiple data, collected from different sources, hospitals, or medical units for the obtained results to be considered relevant. However, the lack of data harmonization and usage of standards between these systems creates a barrier when conducting observational studies. One possible solution is to map the concepts stored in an EHR database to a common data model. This has been the main objective of the Observational Health Data Sciences and Informatics (OHDSI), an international network of researchers.

The Rabbit in a Web application, presented in this dissertation, was developed to ease the mapping process, creating a collaborative environment where several users can participate in specifying the relationships between data. The application follows a client-server model with an architecture divided into three layers and focuses mainly on usability creating a user interface that uses modern web technologies. The server was developed using the Spring Boot framework while on the client-side, ReactJS was used for the application's frontend.

# Contents

# List of Figures

# List of Tables

# Glossary

| | |
|---|---|
| **ANSI** | American National Standards Institute |
| **API** | Application Programming Interface |
| **AUI** | Atom Unique Identifier |
| **AUROC** | Area Under the Receiver Operating Characteristic |
| **CDM** | Common Data Model |
| **CI/CD** | Continuous Integration/Continuous Delivery |
| **CRUD** | Create, Read, Update and Delete |
| **CSS** | Cascade Style Sheet |
| **CSV** | Comma Separated Value |
| **CUI** | Concept Unique Identifier |
| **HL7 FHIR** | HL7 Fast Healthcare Interoperability Resources |
| **ETL** | Extract-Transform-Load |
| **EHR** | Electronic Health Record |
| **EMR** | Electronic Medical Record |
| **GUI** | Graphic User Interface |
| **GWT** | Google Web Toolkit |
| **HL7** | Health Level Seven International |
| **HL7 v2** | HL7 Version 2 |
| **HL7 v3** | HL7 Version 3 |
| **HTTP** | Hypertext Transfer Protocol |
| **JVM** | Java Virtual Machine |
| **JWT** | JSON Web Token |
| **HTML** | HyperText Markup Language |
| **ICD** | International Classification of Diseases |
| **ISO** | International Organization for Standardization |
| **LOINC** | Logical Observation Identifiers, Names, and Codes |
| **MPP** | Massively Parallel Processing |
| **NoSQL** | Not only SQL |
| **NPM** | Node Package Manager |
| **OAS** | OpenAPI Specification |
| **OHDSI** | Observational Health Data Sciences and Informatics |
| **OMOP CDM** | Observational Medical Outcomes Partnership CDM |
| **OS** | Operating System |
| **PHR** | Patient Health Record |
| **POM** | Project Object Model |
| **RBAC** | Role-Based Access Control |
| **RDBMS** | Relational Database Management System |
| **RIM** | HL7 Reference Information Model |
| **RNN** | Recurrent Neural Network |
| **SDO** | Standard Development Organization |

**SNOMED CT** Systematized Nomenclature of Medicine Clinical Terms
**SQL** Structured Query Language
**VM** Virtual Machine
**UI** User Interface
**UMLS** United Medical Language System

CHAPTER 1

# Introduction

## 1.1 Motivation

The Electronic Health Records (EHRs) have been used in hospitals and medical units since a few decades ago to improve the quality of healthcare provided to patients [1]. However, each one of these institutions uses a data model that is different from others, and the records, which are usually plain text, don't follow a terminology or classification system to represent data. This lack of standardization and data harmonization reduces interoperability between EHR systems and makes the comparison of the results difficult [2].

Common Data Models (CDMs), namely the Observational Medical Outcomes Partnership CDM (OMOP CDM), allow mapping medical concepts stored in EHR databases to their standard definition. This way is possible to use data from multiple sources in order to improve the quality of obtained results and to mitigate the risk of biased results [3]. However, the mapping process is lingering mainly because of the lack of data harmonization described above.

The desktop application Rabbit in a Hat was designed to help in the process: it allows to dynamically create mappings between tables and fields (or columns) from both databases and provides detailed information to create correct mappings. The lack of a collaborative environment that would allow specialists to work simultaneously from different locations and other usability issues make this application quite unusable.

Possible solutions to implement the collaborative features and to solve the usability issues is to adapt Rabbit in a Hat into a web application so that multiple users could use it in different locations.

This dissertation proposes to create a pipeline to migrate data from an EHR database to the OMOP CDM which includes the integration of the web application with an Extract-Transform-Load (ETL) tool.

## 1.2 Objectives

The dissertation aims to create a web application capable of mapping medical concepts stored in an EHR database to their standard definiton in the OMOP CDM. The application should follow a client-server model with modular components. Furthermore, features must be implemented to achieve a collaborative environment. Finally, the application needs to be integrated with an ETL tool to migrate data from one database to another.

Following a top-down approach, it is necessary to follow the next steps:

- Study the importance of interoperability in EHR systems and how standards can be used to improve it.
- Analyse ETL tools and explore the desktop application Rabbit in a Hat to understand its strengths and limitations. It was possible to conclude that it was necessary to adapt it into a web application.
- Analyze different approaches to understand how the adaptation process could be conducted. The best solution is to create a web application following a client-server model.
- Define the system's requirements as well as its architecture.
- System development using proper languages and frameworks.

## 1.3 Outline

The remaining chapters of this dissertation follow the structure described below:

**Chapter 2** describes the state-of-art of mechanisms to improve interoperability between EHR systems. The dissertation's goal only considers the OMOP CDM, but it was essential to understand the usage of standards in other levels. It also presents a description of the most relevant ETL tools and other approaches that could be followed.

**Chapter 3** aims to define the system's functional requirements describing the existing actors and possible use cases. It also specifies which quality attributes are desired to achieve.

**Chapter 4** presents the system's architecture and its division into three tiers with multiple layers and modules. It provides a theoretical description of each module and its responsibilities following a technology-free approach.

On the other hand, **chapter 5** studies the system's implementation, discussing the used frameworks and why possible alternatives were not chosen.

CHAPTER 2

# Background

This chapter aims to describe the reasons that lead to the lack of interoperability and data harmonization between EHR systems and presents solutions to mitigate this problem on multiple levels, from the record itself until the communication between systems. It also provides an overview of the current best ETL frameworks, discussing the advantages and disadvantages of each one considering the needs of this project.

Finally, it presents the desktop applications White Rabbit and Rabbit in a Hat, detailing different methodologies to migrate the last into a web environment where it could be accessible through a web browser.

## 2.1 Electronic Health Records

An EHR is, according to its International Organization for Standardization (ISO) definition, "an official repository of information that keeps track of all health status and treatments that a patient has been submitted to" [4]. In other words, it is an electronic version of a patient's medical information and includes their medical history, personal information or test results, and administrative clinical data [1], [5].

The EHRs have replaced the traditional paper forms, which usually include redundancy and typically don't use standardized terminology. Besides, they fail to deliver accurate data, and the documentation is generally incomplete [6].

Electronic Medical Records (EMRs), Patient Health Records (PHRs), and EHRs have been used almost as synonyms. However, they differ in the target users and in who maintains them. EMRs are only used by healthcare providers, and PHRs are exclusively used by patients. EHRs are used by both groups but maintained only by the patients [7].

These health records are patient-centered, and the primary goal of their usage is to provide better medical quality to patients and improve the population's health [8]. Patient information retrieval is faster and more secure, increasing the quality of care and productivity. It is also possible to access and manage a broader range of data, including demographic information [1].

The EHRs can also be used secondarily by researchers to conduct observational studies to improve our knowledge in the medical area. These studies analyze the cause-effect relationship of procedures or treatments to a risk factor in subjects that select their treatment environment. The main difference from experimental studies is the absence of control and test groups and, therefore, who and who are not submitted to the procedure or treatment [9].

Yang *et al.* used EHRs to build a Poisson model in order to discover patterns that would predict the number of Angioplasty, a coronary artery procedure. The goal of this research was to provide better estimations on coronary artery disease [10]. Another way of improving medical quality with health data is by providing more adequate medical prescriptions to patients. Electronic prescriptions (E-prescriptions) systems, such as E-Prescribing[1], use EHRs and share the information to provide more appropriate drug prescriptions, helping in the decision making and reducing the risks that could occur in the traditional paper prescription.

The observational studies have greater timeliness and generalizability; however, using data from a small number of sources could lead to biased results. To mitigate this problem is essential to increase the number of data sources from different medical units or hospitals. Only with a more significant data set with more patients' data, it is possible to contrast the obtained results and understand the effect of potential capture bias [11].

However, EHR data and systems have the disadvantage of lack of interoperability, i.e., don't have the "ability (...) to work with other systems or products without special effort on the part of the customer" [12]. Furthermore, some records consist of unstructured text and don't use standards to define codes, classifications, or nomenclatures [13]. This lack of standardization reduces data quality and hinders results comparison [2], [11]. Improving interoperability is essential to increase efficiency by providing information more quickly, which helps in the decision-making process [14].

The following sections will present mechanisms that can be used to increase interoperability in various levels of a healthcare information system.

### 2.1.1 Specification standards

The EHRs collected from different sources might have different structures, which decreases interoperability. Standards that create and manage health data with the same structure can be used to mitigate this problem.

A standard is a document approved by a recognized entity that provides guidelines or rules for repeated use. The advantages that using standards can bring are flexibility and risk and cost reduction. The larger the number of databases or systems used, the greater the benefits that can be achieved [14].

Open-source standards increase interoperability and reduce the impact on the production of valuable knowledge for clinical decisions. The openEHR[2], developed by the openEHR Foundation, is an example of an open-source standard created to grant interoperability between

[1] `https://www.practiceehr.com/features/e-prescribing`
[2] `https://www.openehr.org/`

systems and is used to store, manage, and run queries on EHRs [15]. Due to its free access, professionals without medical experience can contribute to the modeling and structuring of forms [16].

Figure 2.1 represents the composition of the openEHR standard and how data is structured. The model is mainly composed of two of the components presented in the figure: the reference model and the archetypes. This two-level modeling allows the separation of representation of EHR instances from the domain knowledge, enabling semantic interoperability.

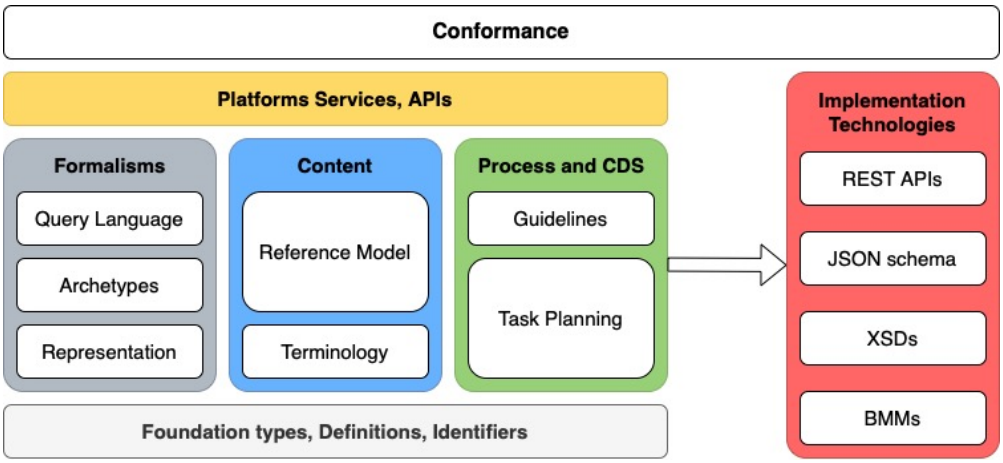



**Figure 2.1:** openEHR standard components and structure [17]

The first level, the reference model, defines the necessary structures and attributes to represent an EHR. On the other hand, the second level focuses on archetypes and templates and holds the medical knowledge. An archetype represents medical concepts that are structured following a meta-standard, the Archetype Definition Language, improving semantic interoperability [18]. Since archetypes use the same structure, it is possible to reuse them, which promotes reusability. Therefore, the openEHR Foundation provides a free access repository, the Clinical Knowledge Management[3], to store and manage the archetypes and templates. A template collects and restrains archetypes and is used to create Application Programming Interfaces (APIs) and user interface forms.

Other advantages of the OpenEHR are its easy-to-use interface and database schema that provides critical information to users [5].

OpenEHR is independent of the clinical terminology standard adopted and allows to conceptualize every clinical concept. However, it is still possible to use other terminology standards, such as the Systematized Nomenclature of Medicine Clinical Terms (SNOMED CT) or the Logical Observation Identifiers, Names, and Codes (LOINC), in the archetypes [19].

### 2.1.2 Semantic standards

Besides using standards to define the EHR structure, it is possible to increase the interoperability of systems by adopting semantic and terminology standards.

[3] `https://ckm.openehr.org/ckm/`

Only a few decades ago clinical terms started being organized which led to ambiguity and lack of maintainability in information systems. A terminology standard allows creating codes for clinical terms and to more easily create relations between them [14].

The SNOMED CT[4] is a multilingual clinical healthcare terminology and is one of the most adopted in the world. It is used to ease clinical documentation and analyze data more efficiently [14]. It was released in 2002 and in 2018 consisted of more than 341000 active concepts [20].

More than a coding scheme that allows identifying medical concepts and terms, it is also a multi-dimension classification, enabling relations between concepts. This coding system is better than others for two main reasons: it supports multiple relationships and is virtually future-proof since concepts, terms, their respective codes, and relations can be changed.

The International Classification of Diseases (ICD)[5] is another semantic standard similar to SNOMED CT and it is used worldwide. It was created in 1900 and is currently maintained by the World Health Organization. A new revision is released with more data almost every decade and is currently on the 11th revision [21].

Another way of increasing interoperability with semantic standards is by using repositories of clinical terminologies. The United Medical Language System (UMLS)[6] is an example of a repository of vocabularies and contains mappings to almost all clinical terminologies [22].

It is composed of 4.26 million concepts and more than 10 million relationships between terms. It uses approximately 15.2 million terms from 211 vocabularies, including SNOMED CT and ICD [23].

The existence of multiple names to express the same concept among terminologies and the lack of a unique format to represent them create a barrier in information retrieval. The UMLS solves this issue by mapping various names from different clinical terminologies into a single concept.

The UMLS is composed of the Semantic Network containing semantic types and relations and the SPECIALIST Lexicon and Lexical Tools. However, its main component is the Metathesaurus which includes terms and codes from other terminology standards.

The entity that structures knowledge is called concept and is identified by a Concept Unique Identifier (CUI), name, one or more semantic types, and other attributes. Terms from clinical terminologies, e.g., atrial fibrillation, are mapped into a single concept and are denominated as atoms. They are composed of an Atom Unique Identifier (AUI) given by the UMLS, a code that identifies the standard where it comes from, and a name [23]. Figure 2.2 is a schematic representation of the information in this standard.

[4] `https://www.snomed.org/snomed-ct/get-snomed`

[5] `https://www.who.int/standards/classifications/classification-of-diseases`

[6] `https://www.nlm.nih.gov/research/umls/index.html`

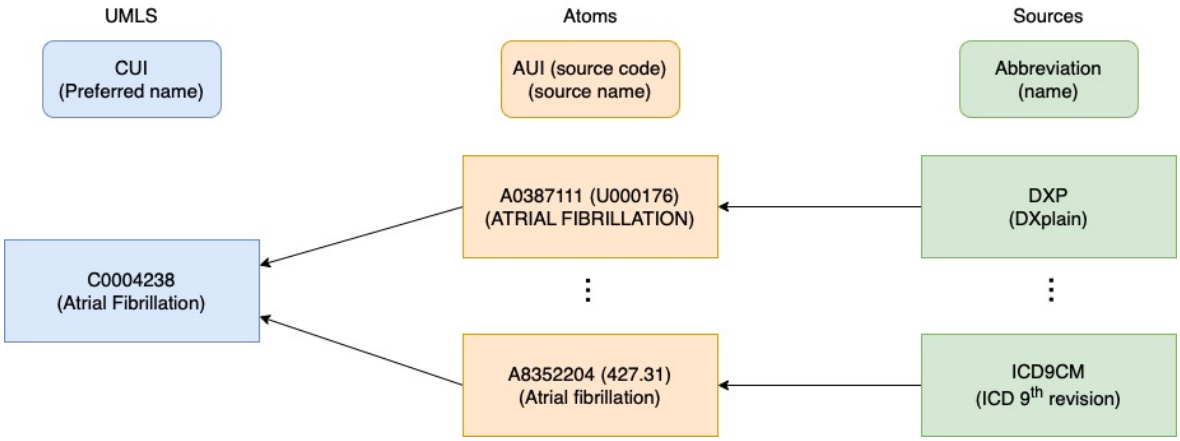



**Figure 2.2:** Knowledge representation in UMLS [23]

Rasmy *et al.* studied the impact of the terminology standard in two prediction tasks [24]. They used data from 5 different standards, including the UMLS and the 9th and 10th revision of the ICD, to create an L2LR and Recurrent Neural Network (RNN) prediction models. The tasks consisted of predicting heart failure caused by diabetes and the occurrence of pancreatic cancer.

The L2LR model presented the best results when using the UMLS in both tasks considering the Area Under the Receiver Operating Characteristic (AUROC) metric. On the other hand, with the RNN model, the UMLS was better in predicting the occurrence of pancreatic cancer; PheWAS obtained better results for the heart failure task. Table 2.1 presents the best models in each task using the AUROC metric to compare results.

**Table 2.1:** AUROC results obtained for the best terminology standard in each task for both prediction models

| | Heart failure task | Pancreatic cancer task |
|---|---|---|
| L2LR model | UMLS (81.15) | UMLS (80.53) |
| RNN model | PheWAS (85.87) | UMLS (82.24) |

The good results obtained with the UMLS terminology can be explained by its higher number of codes and better semantic consistency and hierarchical relations. Furthermore, it has other advantages, such as changing tables and updating or adding new concepts [25].

### 2.1.3 Messaging standards

Health Level Seven International (HL7)[7] is an international Standard Development Organization (SDO) affiliated with the American National Standards Institute (ANSI) that produces standards to exchange EHR data for clinical and administrative purposes [14]. HL7

[7] `https://www.hl7.org`

also delivers electronic documents structure and content standards, but this section focuses on the messaging standards [26].

The first messaging standard developed by HL7 and the most used globally is HL7 Version 2 (HL7 v2)[8]. Created in 1989 following an *ad hoc* approach, its main feature is integrating hospital systems, including administrative and clinical systems. One of the reasons for its success is because it is based on a few and simple principles. However, the lack of an ontology that could control the communicated concepts and the heavy reliance on local customizations led to the development of a new messaging standard, HL7 Version 3 (HL7 v3)[9].

The third version started being developed in 1995, and unlike the previous version, it followed a top-bottom approach. It introduced the HL7 Reference Information Model (RIM), which defined the structure of the semantic and lexical elements, despite not being considered a model of healthcare data nor a model of any message. However, this standard requires the development of other software systems that can include complex model transformations since it is not directly implementable. Furthermore, HL7 v2 and HL7 v3 are not interoperable, and translation software is necessary to exchange data. Its syntactic complexity and the issues that arose from it were an opportunity to create a new messaging standard, HL7 Version 4 or HL7 Fast Healthcare Interoperability Resources (HL7 FHIR)[10].

This standard, on the other hand, was developed considering the advantages and disadvantages of existing standards and the examples of interoperability that were successfully implemented. The top-bottom approach followed in developing the HL7 v3 was one of its disadvantages; therefore, HL7 FHIR uses an iterative approach. It is a RESTful API that provides Hypertext Transfer Protocol (HTTP) services capable of performing basic Create, Read, Update and Delete (CRUD) operations in EHR data. The healthcare data is captured and shared as a modular data unit called resource. They are distinct and identifiable, have a well-defined boundary, and differ from each other in usage and meaning. Besides, the HL7 FHIR extensions allow extending these resources to meet new requirements [23].

This version of the standard is easy to implement, and its applications are rapidly deployed and can be used with a semantic standard, such as SNOMED CT. The HL7 FHIR defines how EHR data is communicated, and a semantic standard defines the nomenclature that must be followed, increasing the interoperability between systems.

### 2.1.4 Common Data Model

The EHRs can be used in observational studies, but to reduce the risk of obtaining biased results it is important to use multiple sources and databases. However, each one has its data model and uses local terminologies which results in a lack of interoperability. A solution to this problem is to migrate data to a CDM.

This concept is used in a wide range of areas, and it can be defined as a set of schemes that allows the storage of data from different sources with different data models in a standard

[8] `https://www.hl7.org/implement/standards/product_brief.cfm?product_id=185`
[9] `https://www.hl7.org/implement/standards/product_brief.cfm?product_id=186`
[10] `https://www.hl7.org/fhir/`

data model. The utilization of a CDM is especially important when an observational study is being conducted because it allows harmonizing data from various sources of information. Only analyzing data from a large number of patients and disparate sources is possible to contrast the obtained results in order to understand the effect of potential capture bias and to draw conclusions with statistical value [3].

Moreover, when EHR systems are being used, it is important to carefully handle the patient's private information. A CDM alleviates these restrictions because it omits the extraction step, and the data analysis is executed in its native environment.

Data migration into a CDM allows to write and test the analyses only once and then run them on the databases with slight modifications [11]. The most difficult task in the mapping procedure is in the initial phases when the local codes are being mapped to their respective standard concepts in the CDM which require specific knowledge of the local information stored in each EHR database.

However, it is important to note that the data stored in each database will not be migrated to a single CDM database instance. In fact, for each database, it will be used an instance of the CDM to migrate data.

Over the last years, organizations invested to create their own CDM with the same purpose of storing EHR data from various systems. One example is the Sentinel Common Data Model[11] created by Sentinel Operations Center. Similarly, the OMOP CDM, adopted by the Observational Health Data Sciences and Informatics (OHDSI)[12], has been used with the final goal of producing a better analysis of the data collected from different sources using standard vocabularies to store data. Its primary goal is to produce statistical analysis code once and re-use it at the other sources and it was designed considering four observational research purposes:

- To identify groups of patients with healthcare interventions and outcomes.
- To characterize these groups for various parameters such as demographic information or healthcare delivery.
- To predict the occurrence of the outcomes in individual patients.
- To estimate the effect that the interventions have on a population.

The OMOP CDM has changed and evolved and is currently in version 6.0. Figure 2.3 represents an overview of tables and some of the relationships between them.

The data model is divided into various areas:

- **Standardized clinical data** - data about clinical events collected in observation periods and personal information about the patient, such as procedures.
- **Standardized health system data** – information about the healthcare provider.
- **Standardized derived elements** – data extracted from other tables including aggregation in periods, denominated here as "eras," of condition, drugs, and condition occurrence.

[11] `https://www.sentinelinitiative.org/methods-data-tools/sentinel-common-data-model`
[12] `https://www.ohdsi.org/`

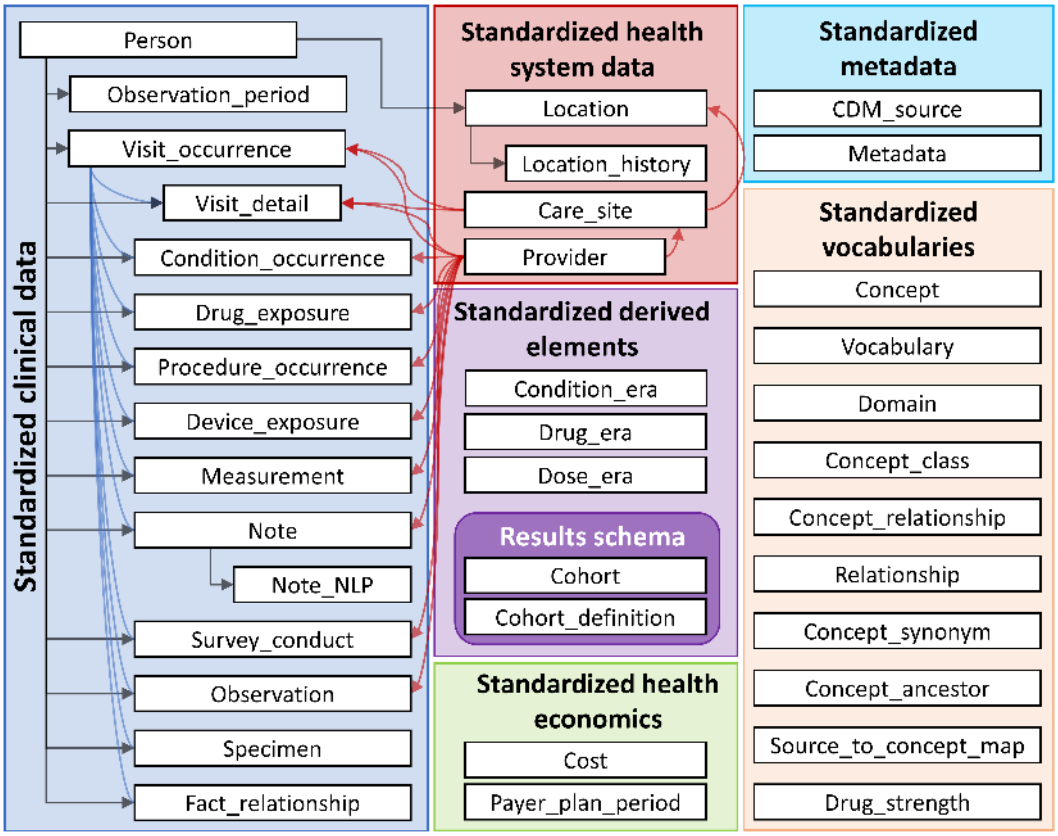



**Figure 2.3:** OMOP CDM structure and relations

- **Standardized health economics** – information about the health plan, costs of patients.
- **Standardized metadata** – general metadata derived from the stored data during the ETL process.
- **Standardized vocabularies** – information about concepts that are used in fact tables, including its domain and class [27].

The focus of this dissertation is to create a pipeline to migrate data from EHR databases to the OMOP CDM. It is important to understand how interoperability in EHR systems can be achieved, but it is also essential to understand how data migration is executed and what are the best frameworks to do that. In the next section, it will be presented the concept of ETL and discussed which tools are better for that end.

## 2.2 Extract-Transform-Load tools

The ETL process consists of storing data that comes from disparate sources into a data warehouse or database. This data must be transformed to be represented in the desired state [28]. This procedure has three steps:

- **Extraction** - where data is analyzed and collected from sources that can be databases, files, or applications.

- **Transform** - can include correction and cleansing of data, removing incorrect or duplicate data, and fixing errors; the main goal is to change collected data so it can fit into the desired format.
- **Loading** - transformed data is stored into a database or a multidimensional structure [29].

The most complete ETL tools and those that satisfy more needs are Oracle Data Integrator[13] and IBM Information Server[14]; however, they are not open-source. From the tools with free access, the ones that are considered the best are Pentaho Data Integration[15] and Talend Open Studio[16] [28].

Most of these tools allow parallelization to obtain gains in perfomance [30]. Other advantages provided by ETL tools are the control of metadata, retrieval of statistics, data profiling and high-quality management, and cleansing [31].

The Pentaho Data Integration possesses a Massively Parallel Processing (MPP) system that uses internal and external memory and databases and, therefore, increases the performance. Nevertheless, one of its major disadvantages is the lack of a mechanism that allows recognizing the data that is changed during the extraction and transformation steps.

OHDSI developed a desktop application, Rabbit in a Hat[17], that is based on the same principles of the ETL process. It allows mapping tables and fields of an EHR database to tables and fields of the OMOP CDM, respectively. ETL tools could be integrated with this application to migrate the data, however, some of its features create usability issues and reduce the potential that it could achieve.

## 2.3 OHDSI tools

OHDSI is an international network of researchers whose main goal is to increase the population's health using health data [32]. They are responsible for the development of some desktop applications, namely Rabbit in a Hat and White Rabbit[18], that typically operate over EHR databases, and the data contained in them.

Rabbit in a Hat, as said before, allows performing the mapping of concepts stored in an EHR database to their standard definition in the OMOP CDM. However, the files containing the information about the database must be generated using the White Rabbit.

The following sections will detail the functioning of these two applications, discussing their main goal and why they are important in the scope of this project.

[13] https://www.oracle.com/middleware/technologies/data-integrator.html
[14] https://www.ibm.com/products/information-server-for-data-integration
[15] https://help.pentaho.com/Documentation/9.1/Products/Pentaho_Data_Integration
[16] https://www.talend.com/products/talend-open-studio/
[17] http://ohdsi.github.io/WhiteRabbit/RabbitInAHat.html
[18] http://ohdsi.github.io/WhiteRabbit/WhiteRabbit.html

### 2.3.1 White Rabbit

White Rabbit is a desktop application developed with Java and using the toolkit Swing to prepare the ETL process [33]. It is responsible for reading the structure of an EHR database and creating a file with it.

The structure can be contained in SAS or delimited text files, but the application can also access the database and directly makes the structure reading and subsequently write it into the file.

It is possible to use multiple Relational Database Management System (RDBMS), including PostgreSQL, MySQL, or Redshift and the obtained file is an XLSX file where each page represents a table and, for each one, the lines represent the columns of the table and some of its attributes, such as the data type, if it is nullable and the table it belongs to. Figure 2.4 represents the application's User Interface (UI) where it is possible to configure the data source.

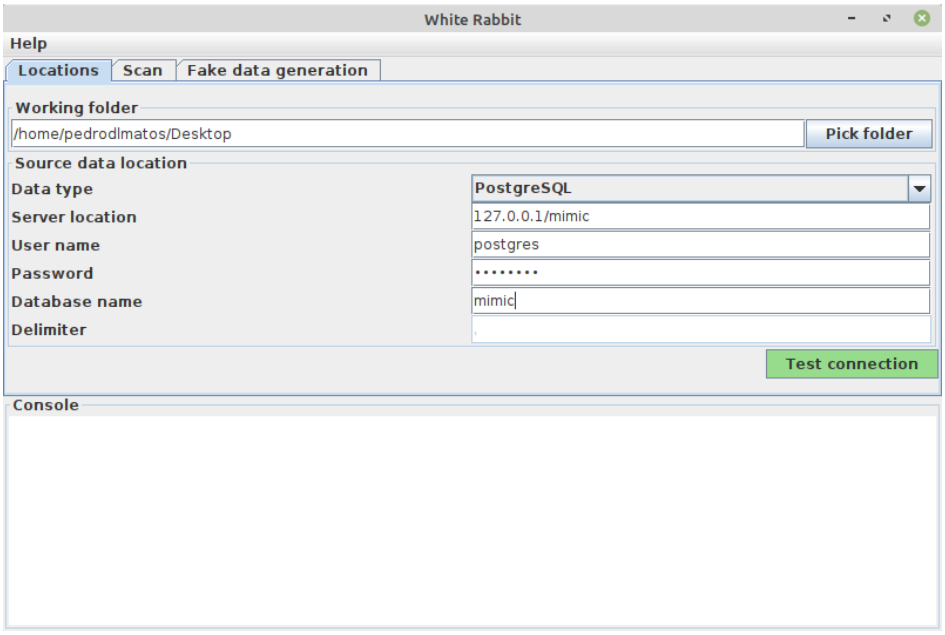



**Figure 2.4:** White Rabbit application's UI

After the configuration, it is possible to select the tables that should be considered in the file and define some parameters, namely the number of rows that must be read for each table. This is important because it gives an idea of the number of empty columns or rows or the maximum length of some columns. Figure 2.5 is a portion of the obtained result. It represents only the data stored on the table *chartevents_7* when the database is created but not populated, i.e., the tables are completely empty.

| Table | Field | Description | Type | Max length | N rows | N rows checked | Fraction empty | N unique values | Fraction unique |
|---|---|---|---|---|---|---|---|---|---|
| chartevents_7 | row_id | | integer | 0 | 0 | 0 | 100,0% | 0 | 0,0% |
| chartevents_7 | subject_id | | integer | 0 | 0 | 0 | 100,0% | 0 | 0,0% |
| chartevents_7 | hadm_id | | integer | 0 | 0 | 0 | 100,0% | 0 | 0,0% |
| chartevents_7 | icustay_id | | integer | 0 | 0 | 0 | 100,0% | 0 | 0,0% |
| chartevents_7 | itemid | | integer | 0 | 0 | 0 | 100,0% | 0 | 0,0% |
| chartevents_7 | charttime | | timestamp without time zone | 0 | 0 | 0 | 100,0% | 0 | 0,0% |
| chartevents_7 | storetime | | timestamp without time zone | 0 | 0 | 0 | 100,0% | 0 | 0,0% |
| chartevents_7 | cgid | | integer | 0 | 0 | 0 | 100,0% | 0 | 0,0% |
| chartevents_7 | value | | character varying | 0 | 0 | 0 | 100,0% | 0 | 0,0% |
| chartevents_7 | valuenum | | double precision | 0 | 0 | 0 | 100,0% | 0 | 0,0% |
| chartevents_7 | valueuom | | character varying | 0 | 0 | 0 | 100,0% | 0 | 0,0% |
| chartevents_7 | warning | | integer | 0 | 0 | 0 | 100,0% | 0 | 0,0% |
| chartevents_7 | error | | integer | 0 | 0 | 0 | 100,0% | 0 | 0,0% |
| chartevents_7 | resultstatus | | character varying | 0 | 0 | 0 | 100,0% | 0 | 0,0% |
| chartevents_7 | stopped | | character varying | 0 | 0 | 0 | 100,0% | 0 | 0,0% |

**Figure 2.5:** File obtained using White Rabbit (partially)

### 2.3.2 Rabbit in a Hat

Various CDMs were designed to receive the migrated data from EHR databases mapping the medical concepts to their standard definition. However, the mapping procedure is a task that can take months to be executed and includes research and the presence of specialists on both databases. Ideally, it should be made in a collaborative environment and using tools that would allow the specialist to work in different locations.

The desktop application Rabbit in a Hat is a Java Swing application developed by OHDSI to ease the mapping procedure between an EHR database and the OMOP CDM. A table from the EHR database is mapped to an OMOP CDM table creating a table mapping. Then, for each one is possible to map fields from the table of the EHR database to the fields of the OMOP CDM table, creating a field mapping.

The application's UI is very intuitive and designed to help users to create the most correct mappings providing details about tables - their columns and respective data type and description - whenever needed. The information about the EHR database is given by a file that is generated using White Rabbit. The version of the OMOP CDM is variable and is possible to choose between version 4.0 and version 6.0. Then, using drag and drop is possible to create table and field mappings. The output obtained is a summary file with the logic behind each mapping that is used by specialists to create scripts to migrate data. Figures 2.6 and 2.7 present Rabbit in a Hat's UI when creating mappings between tables and fields, respectively.

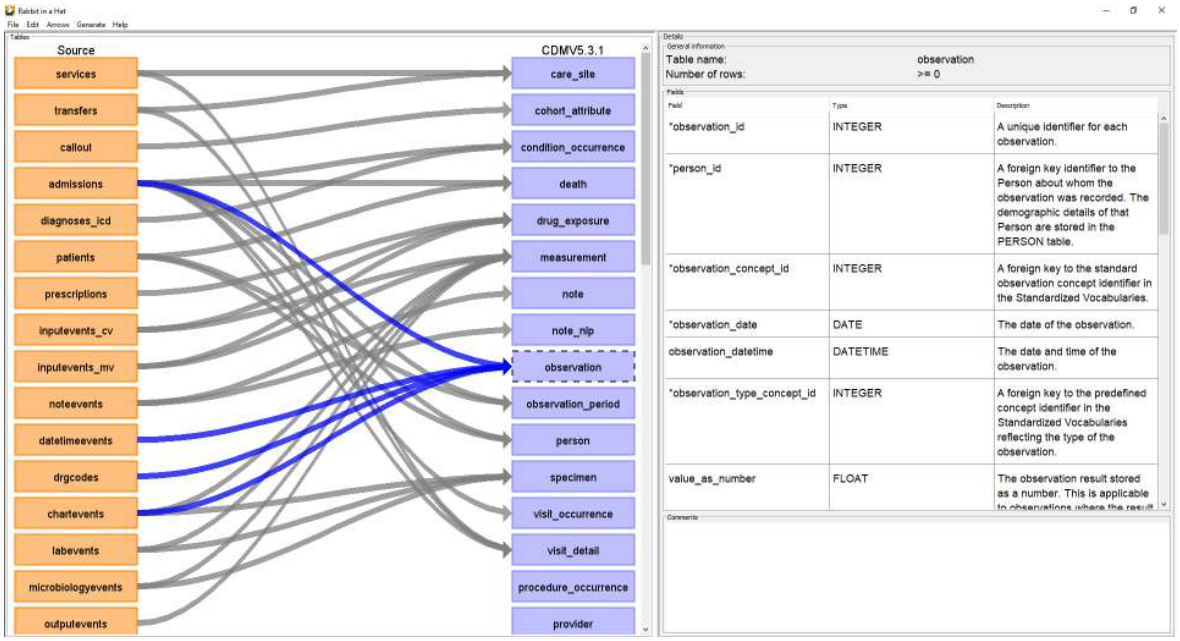



**Figure 2.6:** Mappings between table from MIMIC-III and OMOP CDM version 5.3.1 according to [34]. Selection of the table *observation*

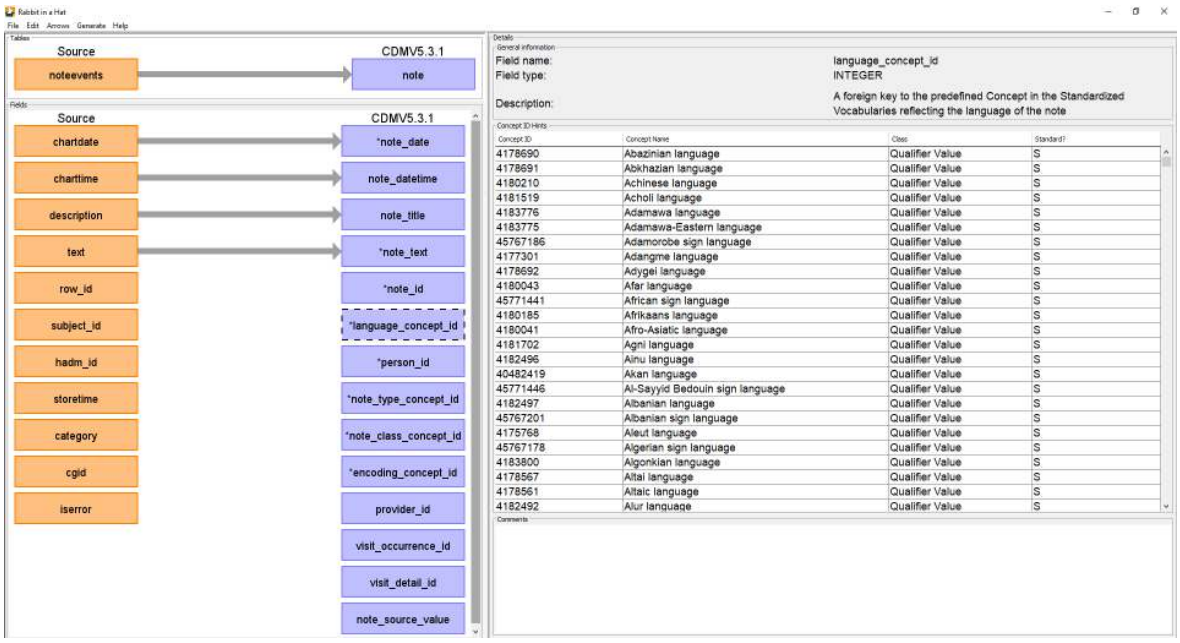



**Figure 2.7:** Mappings between fields from the tables *noteevents* (MIMIC-III) and *note* (OMOP CDM v5.3.1). Selection of the field *language_concept_id* and its concepts.

This tool can be useful, especially in the early stages of the mapping procedure when the knowledge about the databases is still not sufficient to create the most correct mappings. Rabbit in a Hat provides visual information about the databases' tables and fields and its UI is intuitive and easy to use which facilitates the procedure.

However, desktop applications are developed under a set of specific characteristics which makes it impossible or hard to use in a heterogeneous environment where each user has a

machine with different specifications. Another disadvantage inherent to desktop applications is that a collaborative environment is impossible to adopt if the users are working in different locations which is normal during the mapping procedure.

Specifically, Rabbit in a Hat, besides the problems related to desktop applications, is also very restrictive because it only allows using the OMOP CDM, even if it is possible to use its various versions. Another disadvantage is the result obtained: the summary file only contains the documentation for the mappings that were made, the Structured Query Language (SQL) instructions to migrate the data from an EHR database to the CDM database must be written by a programmer.

Moreover, writing the SQL instructions to make the ETL procedure also brings some disadvantages that could be mitigated using proper tools. These instructions usually are executed in a single-threaded environment which results in a worse performance [30].

One of the main disadvantages is the lack of a collaborative environment. A possible solution for this problem is following a web-based approach, migrating the existing desktop application into a web server, or creating a new web application. In the next section, it will be discussed which alternative is better, presenting some tools and the obtained results.

## 2.4 Web applications and tools

Web applications are applications with a Graphic User Interface (GUI) that can be accessed using a web browser. In the specific case of Rabbit in a Hat, using it as a web application helps to adopt a collaborative environment where specialists could work in different locations simultaneously. Another advantage of web applications over desktop applications is that software and hardware limitations are fewer, any user can access the application independently of the operating system or the hardware specifications of the machine that is used.

There are multiple approaches to transform Rabbit in a Hat into a web application, including:

- **Migrate to a web server** - using automatic migration tools that use the original application and deploy it in a web server. The UI is equal to the original but only accessible through a web browser.
- **Adapt the UI** - using frameworks, namely Vaadin[19], to create a GUI in Java similar to the original so that browsers could interpret it. The remaining code could be reused.
- **Create a new web application** - creating a web application following a client-server architecture and using proper frameworks. It is possible to use some of the code.

The next subsections will present these approaches and some of the tools used, discussing the results obtained using some of them.

[19] `https://vaadin.com`

### 2.4.1 Automatic migration tools

A desktop application can be migrated to a web environment using tools that automatically launch a web server and execute the application on that server. This solution does not need a reverse engineering process to adapt the original application to the new environment.

For applications developed in Java Swing, these tools create a Java Virtual Machine (JVM) that is shared between the processes and limits the resources that are being spent. The GUI is presented to the user by transforming Java code into HyperText Markup Language (HTML) elements. However, when more complex libraries are used this process becomes less efficient and some elements are wrongly rendered. Despite reusing totally or partially the code already developed, the results obtained can be lower than expected [35].

AJAX Swing[20] and Webswing[21] are both automatic migration tools that were used in the first instance to understand if this approach was the best one to follow. The obtained results are presented in figures 2.8 and 2.9, respectively.

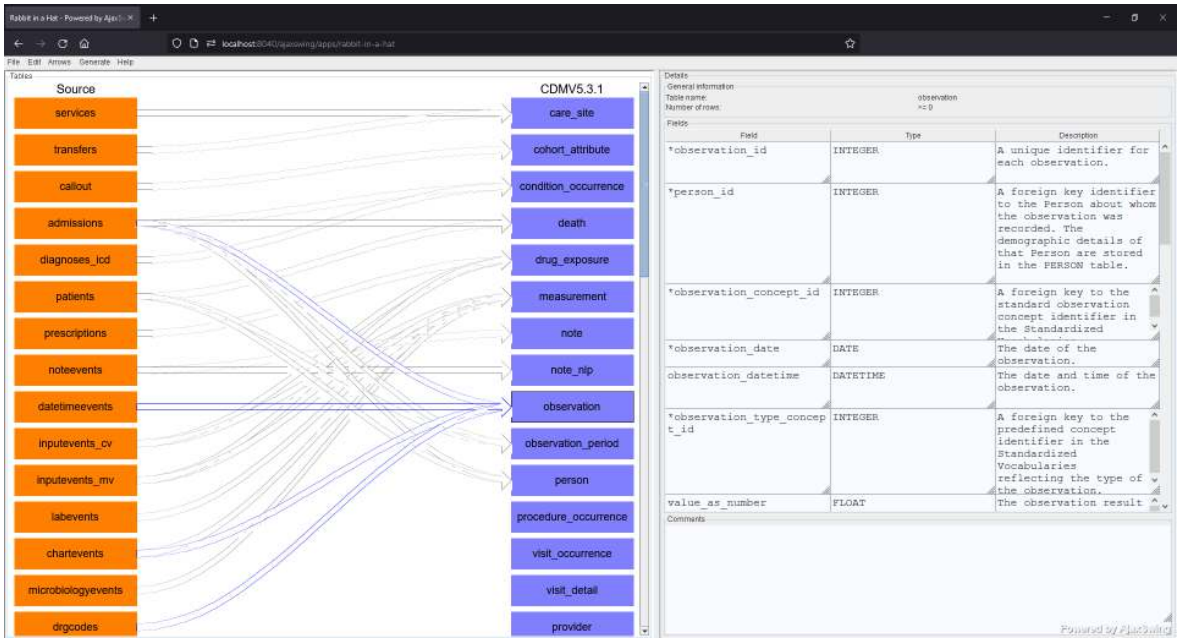



**Figure 2.8:** Rabbit in a Hat running on a web server created by AJAX Swing

[20] http://www.creamtec.com/products/ajaxswing/
[21] https://www.webswing.org/

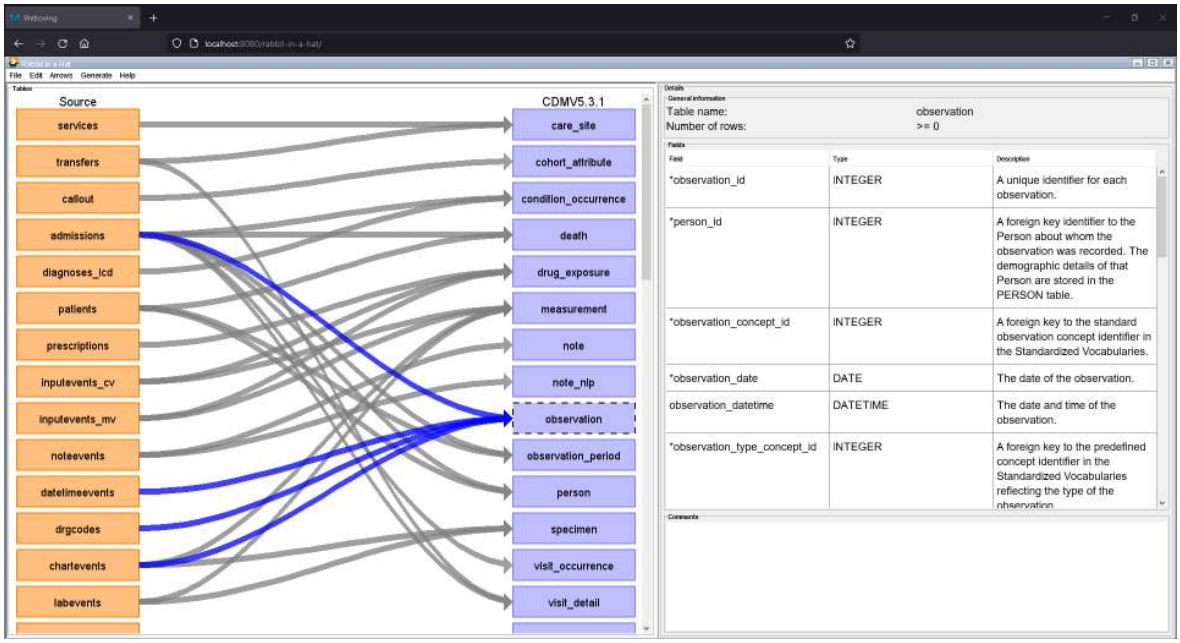



**Figure 2.9:** Rabbit in a Hat running on a web server created by Webswing

As observed, the GUI obtained when using AJAX Swing (Figure 2.8) lacks some elements: arrows that connect tables or fields are not correctly rendered and borders are not present in the final result. On the other hand, Webswing (Figure 2.9), which is a more powerful and complete tool, provides a GUI very identical to the original. However, in both situations, a usability issue comes up: there is a window inside the browser due to the code that is reused.

Another problem that emerges when the code is totally reused is that each time a user accesses the GUI, a new process is launched deleting the previous one and the data stored in it. Therefore, it is only possible to have a single mapping session simultaneously making it impossible to work in a collaborative environment.

The study of these tools was important to consider if it was worthy to try to adapt the original code in order to implement collaborative features. It was possible to conclude that the best approach was to create a new web application using some of the code but creating a new GUI. Since the code is developed in Java, it was firstly used Vaadin which is a framework to develop full-stack applications entirely in Java.

### 2.4.2 Vaadin

Vaadin is a framework that allows the development of a full-stack web application using Java, transforming Java objects into TypeScript objects that would be rendered as HTML elements, making it possible to show them on a web browser.

It is composed of many UI components based in the Google Web Toolkit (GWT) resulting in the same output independently of the platform used. Furthermore, it is possible to receive requests from the browser and render the response in it using a “terminal adapter” [36]. It allows integration with database systems to store data and the usage of Java classes allows the abstraction of the database structure in the "domain layer" [37].

Since Vaadin only uses Java to create both the backend and the frontend, it is possible to reuse the code that makes the management of the mappings between tables and fields. The code of the GUI must be adapted to respect this framework's semantics. Besides reusability, another advantage is the easy integration of Cascade Style Sheet (CSS) in the Java code.

However, events such as drag and drop or mouse-click are not very well supported and JavaScript libraries that would help in some features do not exist for this framework or are difficult to integrate with it. Therefore, some features are almost impossible to develop.

The results obtained using Vaadin allows us to conclude that a new approach must be followed. The proposal is to develop a new web application using proper languages and tools and more decentralized.

### 2.4.3 Full-stack app

The final approach adopted and the most classic is the development of a full-stack web application with the frontend separated from the data layer by a services layer, following a client-server approach. Since the components are separated, it is possible to develop each one using proper tools or languages.

In this case, to develop the frontend, it was used ReactJS[22] which is a JavaScript framework, and its main characteristic is the component-based architecture. This allows the development of reusable modular components. Angular was also considered but it is a more complete framework and, therefore, heavier. Besides, it is less modular than ReactJS which is the main reason why the last was chosen.

As said before, the frontend and the data storage are separated by a processing layer that provides services consumed by the frontend. This intermediary layer was developed using Spring Boot[23], a framework that is used to build Spring applications using Java. The development of the API in Java was a crucial aspect to reuse as much code as possible from the OHDSI applications. This approach is the one that was followed to develop the application, that was denominated Rabbit in a Web, and to solve the dissertation's proposal.

## 2.5 Summary

This chapter started by addressing the problem of the lack of interoperability between EHR systems and databases while presenting standards and other tools, namely CDMs, to mitigate it. Multiple standards were discussed, including openEHR, UMLS, or HL7 FHIR, but the focus was on the OMOP CDM since the dissertation's goal is to migrate data from EHR databases to it.

The concept of ETL was studied and some frameworks were presented but OHDSI's Rabbit in a Hat was discussed in more detail since it is the only application that, despite not being an ETL framework, allows to map data from an EHR database into the OMOP CDM.

[22] https://reactjs.org/

[23] https://spring.io/projects/spring-boot

Its usability issues and the lack of a collaborative environment were presented as the main reasons why it was important to migrate the application into a web environment. Different approaches were studied and followed, and it was possible to conclude that the best one was to create a new web application, more decentralized, based on a client-server model where adequate frameworks and languages must be used on each side of the model.

The next chapters detail the system's specifications, providing a list of well-defined functional and non-functional requirements, and presenting the proposed architecture.

CHAPTER 3

# Requirements analysis

The previous chapter analyzed the desktop application Rabbit in a Hat and its disadvantages, namely its usability issues and lack of collaborative features. Three approaches were followed to understand how to overcome these problems. It was possible to conclude that automatic migration tools and the Java frameworks to create GUIs for web applications were not viable. They would reuse all or most of the existing code, but the major issues would remain.

The development of a new web application using proper frameworks and languages seems to be the best approach to follow. This chapter details the developed solution's functional and non-functional requirements.

This chapters aims to define the system's scope, providing a list of functional requirements. It also discusses the quality attributes and non-functional requirments that must be achieved.

## 3.1 Functional requirements

While exploring the Rabbit in a Hat desktop application and with the opinions of some of its users, it was possible to conclude that some implemented features needed alterations. Moreover, in order to achieve a collaborative environment, it was mandatory to develop new ones. Therefore, some functional requirements have been defined to meet users' needs.

The functional requirements define the system's behaviour, i.e., what it should do to respond to the inputs [38]. However, some of the actions are not available to all end users. The system will be used by two different groups of users, actors. Each one will have a well-defined role and a set of available features. These actors are defined as:

- **Collaborator** – Has permissions to create and edit ETL procedures. Can access other users' profiles and invite them to or remove them from procedures. Doesn't have privileges to manage users in the system.

- **Administrator** - Entity that moderates the application. Has total control over ETL procedures and user management. It is a special kind of collaborator, so has access to the same set of operations as they do.

Figure 3.1 is a simple use case diagram representing the most critical use cases and their relations with the actors. The diagram is divided into two areas:

- **Users management** – Use cases that involve the creation and management of user accounts in the system.
- **ETL procedures management** – Considers the use cases that involve ETL procedures and the operations that users can perform within them.


**Figure 3.1:** Use cases diagram

The use cases represented above can be described as:

- **Sign up and log in** – Users must create an account or log in into the system to access the remaining features. At the moment of registration, the user is given the role of collaborator.
- **Give privileges to collaborators** – Administrators can change the users' roles in the system, giving privileges to collaborators, transforming them into administrators. On the other hand, they can not remove privileges from any actor.
- **Edit user account** - Both administrators and collaborators can edit their own profile. The possible operations include changing username and e-mail.
- **Delete collaborator account** – Administrators can only delete collaborator's accounts. It is not possible to delete accounts from other administrators and collaborators cannot remove any kind of account. Deleting its own account is also not possible.
- **Create ETL procedures** – Collaborators and administrators are allowed to create ETL procedures by using two methods: using the database scan file provided by White Rabbit or using a procedure's summary file created and supplied by the system. The database scan file method creates a procedure that only contains information about tables and fields; the summary file option also contains the information about both databases but might include mappings between tables and fields.
- **Manage collaborators in ETL procedures** – Administrators and collaborators with access to a procedure can invite other users or remove them from the procedure's list of collaborators. However, it is not possible to remove themselves to avoid situations where a procedure does not have collaborators with access to it.
- **Edit ETL procedure** – users with access to a procedure or administrators can edit them. The operations available are:
  - Change the ETL procedure's name.
  - Change the EHR database name.
  - Change the OMOP CDM version.
  - Change comment of tables and fields from both databases.
  - Create and remove mappings that connect a table from the EHR database and another from the OMOP CDM database, change their completion status and logic.
  - Inside a mapping that connects tables, create and remove mappings between the fields from both tables following the same logic: a field from the EHR side connects to another from the OMOP CDM side and change the field mapping's logic.
- **Download summary files** – Summary files are generated as the output of the system to users. They contain information about a procedure from different perspectives and it is possible to create different types of files, including:
  - Files for each database that include a list of their fields and additional information for each one, namely the comment, the table it belongs to, and the field mappings connected to it.

  - Summary file for table mappings that shows for each table from the EHR database, which tables from the OMOP CDM it is connected to and respective logic and field mappings.
- **Save ETL procedure into file** – Collaborators and administrators can save a procedure into a file. This file includes all alterations that have been made since its creation and can be used to create a new procedure.
- **Mark an ETL procedure as deleted** – Collaborators can delete a procedure. However, it is not removed from the system; it remains persisted but not visible to the collaborators that had access to it, only to administrators.
- **Mark an ETL procedure as not deleted** – Administrators have access to all procedures, even those that have been marked as deleted by their collaborators. They can also change their deletion state, making them accessible again to the collaborators that previously had access to it.
- **Delete an ETL procedure** – Administrators can remove a procedure from the system, deleting it entirely. This operation can be applied to all procedures, independently of their deletion state. Moreover, it is an irreversible action, hence only administrators can perform it.

## 3.2 Non-functional requirements

Functional requirements specify what the system should do. On the other hand, non-functional requirements, or quality attributes, define how it should perform the features; they are measurable properties that define how well the system satisfies the users' needs [38].

The application was developed to achieve some quality attributes, namely maintainability, usability, performance, and modularity. Below, some requirements that were considered to achieve them are described:

- **Usability** - The application's GUI must be intuitive and easy to use. Tips must be provided to help the users in case of not knowing how to proceed. For some of the features, there must be multiple ways to perform them.
- **Modularity** - The system must be designed to have various small components with well-defined responsibilities. The modularity and responsibility assignment helps in maintainability and the discovery of sources of error.
- **Role-Based Access Control (RBAC)** – There are multiple roles, and each has a set of permissions, i.e., a well-defined list of possible features. Users are assigned roles at the moment of registration, and administrators can manage the roles, giving permissions to users [39].
- **Easy to deploy** – The system's deployment should be a procedure that hides all the complexity from the end-users in all stages. With a couple of steps, users must be able to have the application ready to use. Proper technologies must be adopted to ease the installation process [40].

- **Virtualization** - Virtualization must be considered, adopting the usage of containers for each system's component that runs over the host's Operating System (OS). Virtualization also allows achieving availability by launching new container instances whenever needed (horizontal scaling). In the production environment, the system must be deployed in a Virtual Machine (VM) instead of physical machines [41].
- **Web technologies** – The system must run in a web environment and the application's GUI must be accessible using a web browser, namely Mozilla Firefox and Google Chrome. Technologies, such as HTML5, CSS3, and JavaScript must be used to create a responsive interface, regardless of the user's environment [42].

## 3.3 Summary

This chapter defined the system's functional requirements, describing the actors and their roles and the set of features available for each. Moreover, it was also analyzed the quality attributes that the system should achieve and that must be considered during its development.

The next chapter presents the proposed solution's architecture, explaining the responsibilities of each component and module.

CHAPTER 4

# Architecture Proposal

The previous chapter detailed the system's requirements and to fulfill them it is essential to specify the application's architecture. The study and definition of the architecture is a central aspect of the development because it improves the system adaptability. This chapter describes the solution's architecture focusing on the theoretical specifications and not on the used technologies.

## 4.1 Client-server model

The software architecture defines the system's organization: the scope and responsibilities of each component and how they interact with the others [43].

The solution's architecture is based on a client-server model where the clients send requests that the server responds to. This model allows separating the processing through various machines or containers since the server is separated from the client-side [44]. Furthermore, it follows a 3-tier architecture which includes the client-side, the application server, or middleware, and the database server, as in Figure 4.1.

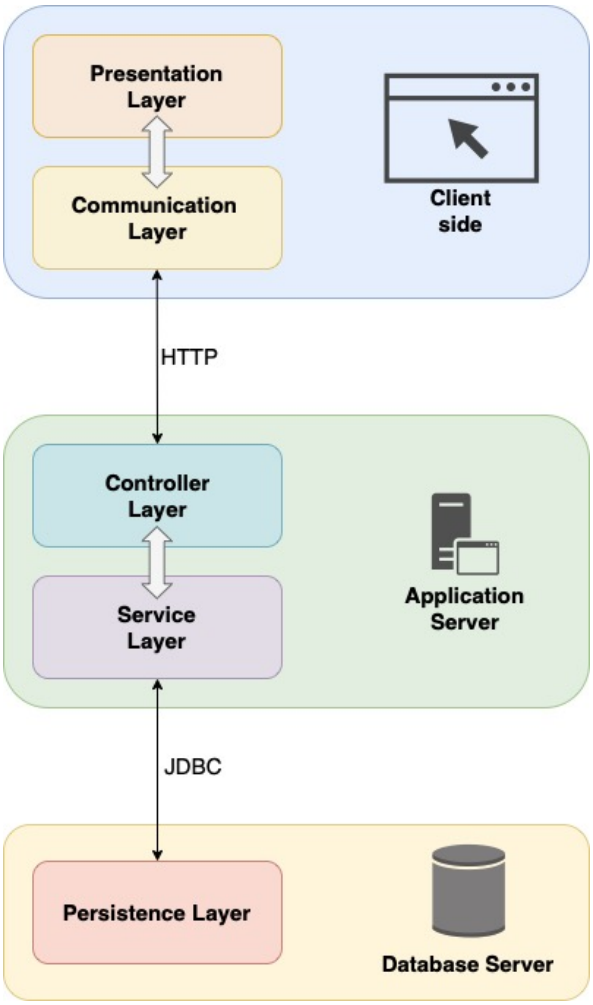



**Figure 4.1:** System's architecture following a 3-tier client-server model

The client-side tier is responsible for sending requests to the application server and presenting the application's interface to users in a web browser with the data obtained from the response. It is divided into two layers:

- **Presentation layer** – renders the GUI and presents the correct information according to the request the user sent.
- **Communication layer** – sends requests to the application server's RESTful API and sends the response to the presentation layer. The communication between the client-side and the application server adopts the HTTP protocol.

The application server receives requests from the client-side and sends the response with all necessary information. Besides, all the system's logic is processed and executed in this tier, adding, removing, or updating entries in the database. Therefore, it also communicates with the database server to manage the stored data. As with the client-side tier, it is also divided into two layers:

- **Controller layer** – consists of a RESTful API that provides a set of services to be consumed by the clients. These services are accessible using endpoints.

- **Service layer** – is the layer that contains all the system's business logic. When the controller layer receives a request, it calls the methods in this layer in order to execute the necessary operations and to obtain the data to respond properly.

The last tier, the database server, only contains one layer, the persistence layer. This tier is responsible for the data management in the database. The service layer communicates with the database server to get data from the database, add new entries and update the existing ones.

This architecture only specifies the responsibilities of each layer and why they communicate with the others. However, some layers or tiers might have a large set of responsibilities; for example, the application service tier is responsible for creating ETL procedures and all the edition features of them, managing users and roles, and for the authentication methods.

Separating the system into multiple components helps increasing cohesion since each one is responsible for a defined and smaller set of responsibilities. Furthermore, it also improves the system's modularity and adaptability because new features are easier to implement. Figure 4.2 presents the system's component diagram.

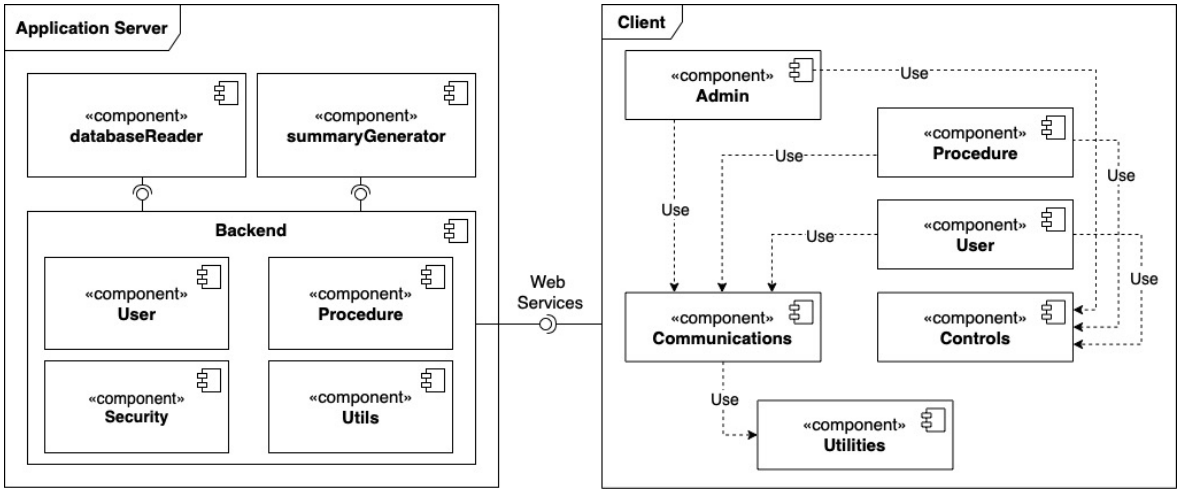



**Figure 4.2:** Component diagram of system's architecture

The tiers and layers presented previously are not represented in this diagram. The client-server model is a more abstract representation of the system's architecture, focusing on its physical details. On the other hand, the component diagram allows creating a logical separation of the component's scope while keeping the adoption of good practices and principles. The following sections will discuss the application server and client components, detailing each module and its responsibilities.

## 4.2 Server-side components

The application server is mainly responsible for the data management, namely the management and edition of ETL procedures and users. It contains a RESTful API that provides

the services consumed by the client-side and all the business logic and rules. Moreover, this component also contains the authentication methods and settings that define the public and private endpoints.

### 4.2.1 Database reader

The database reader module is responsible for reading the files that contain information about databases. For each version of the OMOP CDM since its version 4.0, there is a Comma Separated Value (CSV) file that contains the information of tables and columns (or fields) of that version. These files are organized by fields, i.e., each row describes a column of the table and includes attributes such as the name, the data type, and the table where it belongs to.

On the other hand, the files containing the EHR database's structure must be created using the White Rabbit application. The created file is similar to the CSV files described above but includes more information about tables, namely their number of rows, the number of checked rows or the number of rows that are empty.

In both situations, this module contains the classes and methods to read the files, to create the fields, tables, and databases objects, and to persist them on the database. Some methods from this component were based on the existing ones from the White Rabbit application. The main alterations were made to adapt the objects to the new data model.

### 4.2.2 Summary generator

Summary files are a central aspect of the system's functional requirements, as defined previously in section 3.1. There are three types of summary files: two CSV files for the columns from the EHR database and the OMOP CDM, and a text file with the mappings between tables and fields, as presented in the web browser.

The CSV files are created iterating over the tables of the desired database and writing one line at a time. However, to create the summary file of the mappings, it is not possible to create an image from the GUI since the application server is separated from the client.

This component contains the methods to write the files, including methods to create the images. In the application Rabbit in a Hat, it was possible to create an image of what was being displayed in the UI. Therefore, it is necessary to create an adapter that receives an ETL procedure object respecting the system's data model as input and changes the attributes to respect the model from the desktop application. Then, using Java Swing is possible to create a panel and create an image from it. Figure 4.3 represents the adaption process and into which classes are attributes adapted.

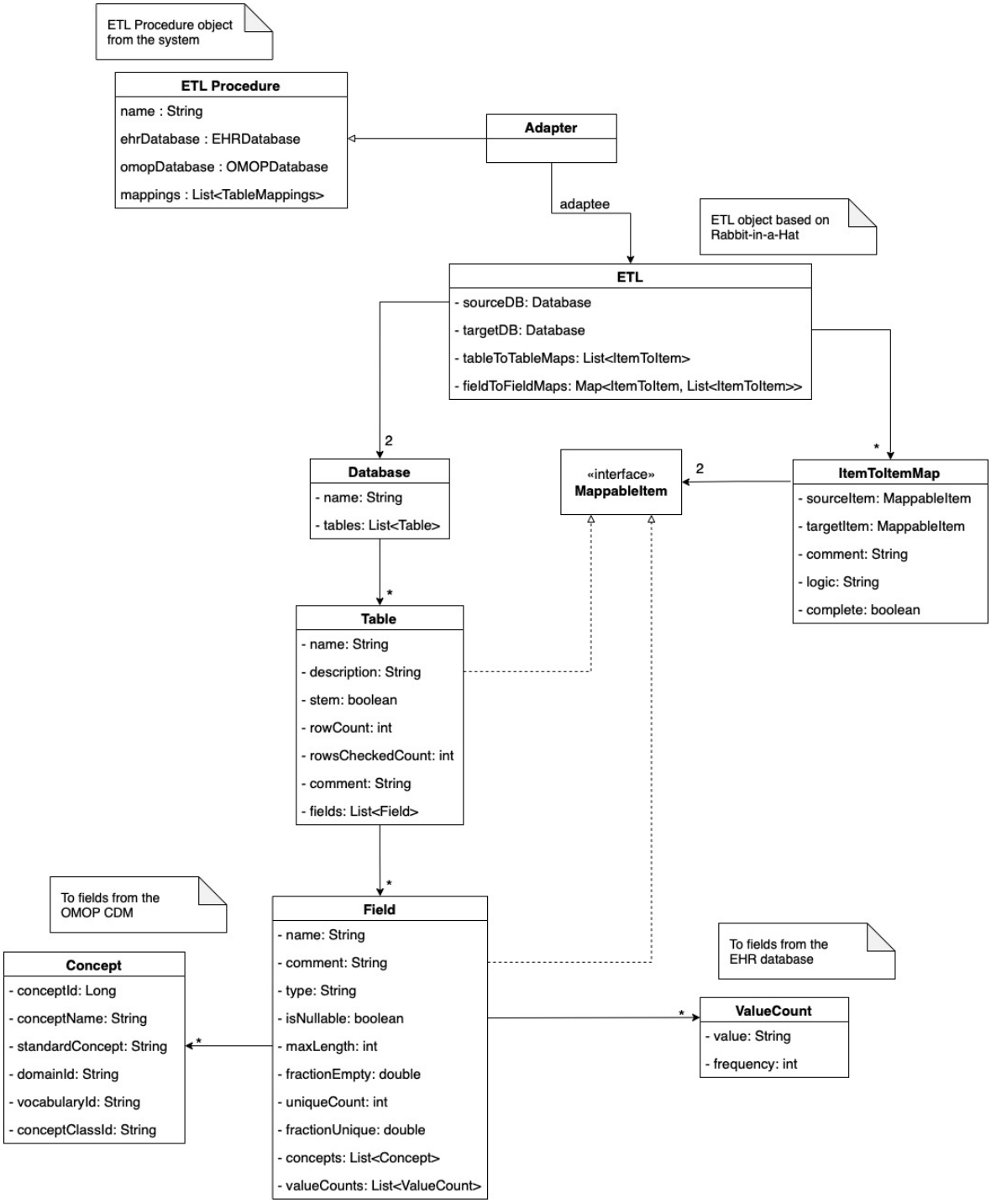



**Figure 4.3:** Adapter pattern to transform ETL objects

The classes to create the panel with the tables and mappings, and the image from it were adapted from the original code from Rabbit in a Hat. This component and the database reader component only use the necessary code from the original applications and modules. The main reason to only use a portion of code is that it's easier to change classes and attributes and obtain the objects in the desired state.

Fully exporting them as dependencies of the system is more straightforward but has the

disadvantage of adding unnecessary complexity. Besides, it could be necessary to create more adapters to use the classes of those modules, which increases the coupling between classes.

These two components essentially handle the system's inputs and outputs. However, despite the adaption of some of the White Rabbit's code, it is still necessary to use the application to generate the EHR database's structure file. Figure 4.4 is a diagram that represents the input and output files and the interaction between applications.

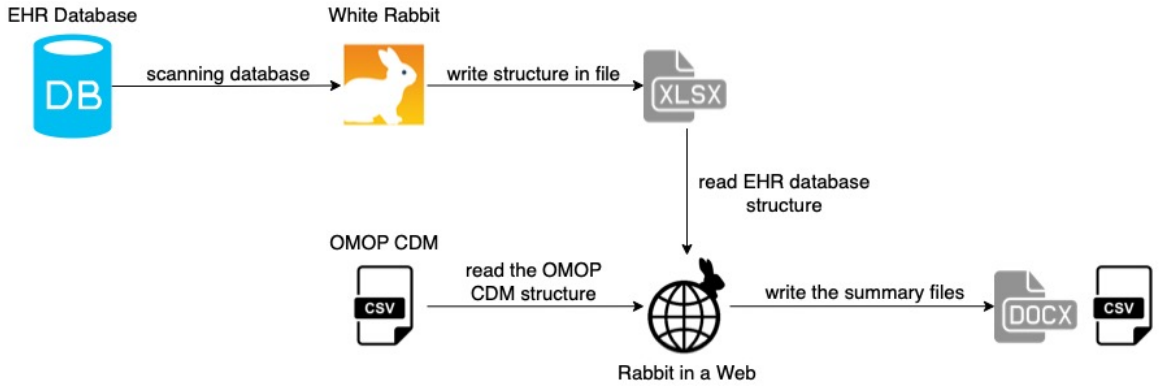



**Figure 4.4:** System's data pipeline and interaction between applications

### 4.2.3 Backend

The modules described previously are responsible for the file reading and writing operations and are used in some of the system's features. However, the main component of the application server is the backend.

It is the largest component of the application server since it contains most of its modules. Furthermore, the RESTful API that provides the web services to be consumed by the clients is present in this component as well as all the business logic and rules. The backend is also responsible to add security to the system, applying the necessary authentication and authorization methods. Figure 4.2 showed in detail how the backend module is structured, specifying the modules that provide services to the users.

*Procedure*

The ETL procedures are involved in the majority of the functional requirements since it is possible to create, manage, and delete them. Moreover, there are multiple ways of editing them, including adding or removing mappings between tables or fields, changing the comment or the logic.

This module would contain most of the business logic, so the best way to keep the modularity is to divide it into multiple submodules, each one dealing with a specific element from the procedure object. Figure 4.5 represents the module's composition.

It is possible two observe that a procedure component contains modules for the two databases and the table mappings. Each database module is responsible for the creation of

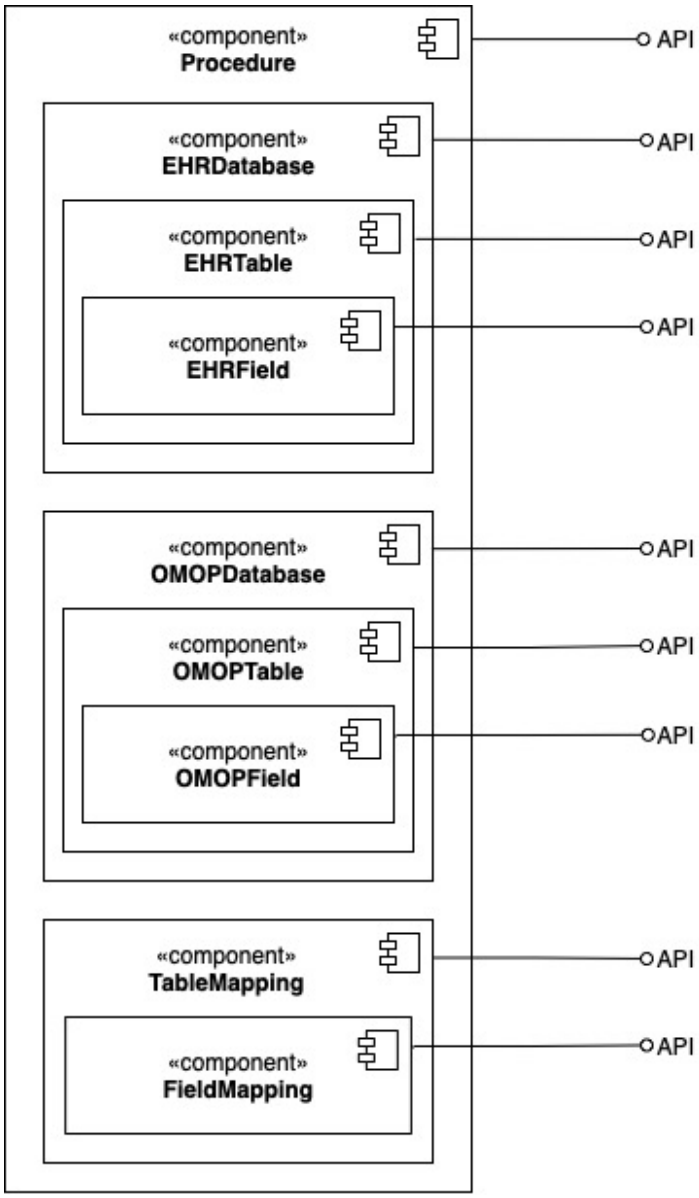



**Figure 4.5:** Procedure component modules

the respective object and its persistence on the database. Furthermore, the modules that deal with their tables and fields are also contained inside the database module because they are a part of it.

The table mapping module contains the operations related to the management of mappings between tables and fields, allowing to change the logic of mappings, or mark a mapping between tables as complete.

All the modules provide services to the users through the RESTful API. Some of them only include one or two services, and for that reason, they could be aggregated with the other services from another module. However, this would increase the coupling between the components, reducing the system's modularity.

*Users*

The users can only use the application if they are logged in with the credentials provided at the registration moment. A user needs to specify its username, email, and password, which is encrypted before storing it on the database.

The existence of roles allows users to execute different sets of operations and features. Therefore, to specify the privileges, it is also necessary to define a set of roles as a user's attribute, that can be later altered by administrators.

The management of users and roles is a responsibility of the user component, which also allows the edition of the users' accounts details, such as changing the email. It also provides the services to create an account, verifying if some of the provided attributes have already been taken, and log in to the system.

*Security*

Security was not one of the quality attributes defined in section 3.2; however, it is important to keep some of the user's information private and the services are only accessible if the user is authenticated and has the authorization to access them.

One of the responsibilities of this component is to retrieve data from the request to sign up and log in to the system. When a user sends a request to register in the system, the password must be encrypted. On the other hand, if the user tries to sign in to the system, this component must verify if the credentials are correct. The security and user components seem to have the same functionalities. The main difference is that the user component must provide the services to receive the authentication requests and the security component must validate the data in them.

When the user successfully logs in to the system, a JSON Web Token (JWT) is created and sent back as a response. This token is a security token that identifies the user who made the request and verifies if he has the authorization to access a resource or service [45]. Then, the following request made by the user contains the JWT in its header and if it is still valid, the system sends a proper response according to the request. Figure 4.6 is an interaction diagram that exemplifies the procedure described above.

Finally, this module is also responsible for defining which endpoints are public: there is no need to be authenticated to access them.

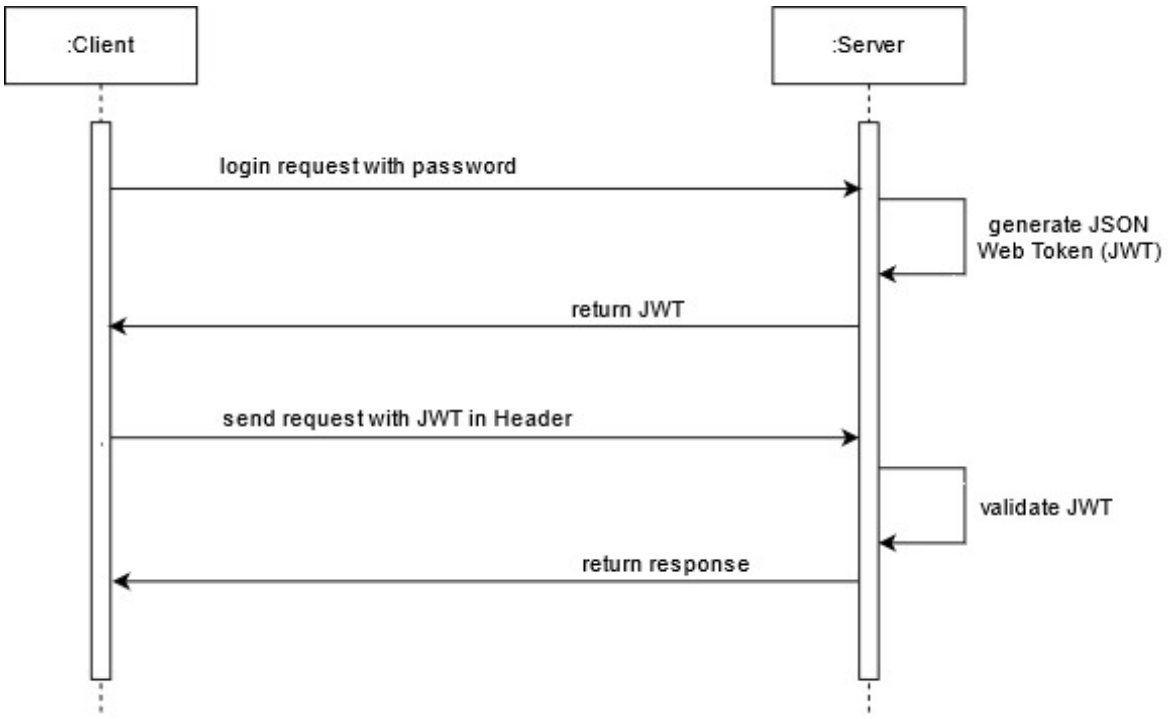



**Figure 4.6:** Authentication using JWT behaviour

*Utils*

This component has the responsibility of sending proper responses to clients. When operations related to a request are executed with no errors, the response should only contain the necessary data. For example, if the client changes a table comment, the only object that must be sent in the response is the altered table object. Static classes are defined in this component to specify which attributes should be included in the response, according to the request that has been received.

On the other hand, if the user is not authorized to access a service, a response with the error code and a description should be sent. Also, if there is a failure during the execution of the operations, the response should include a description of the error.

## 4.3 Client-side components

The client-side communicates with the application server's RESTful API to consume its services. The data contained in the received responses should be interpreted and displayed in the GUI to the users.

As in the server-side, the focus was to divide the client-side into small modules with well-defined responsibilities. Not only does modularity increase, but reusability also increases because some classes that define HTML elements, such as buttons, can be used in multiple places.

Figure 4.7 is a component diagram that represents how the client-side is structured. Almost all components communicate with the server's API but are mediated by a specific module with methods designed to handle communications. In the following subsections, these modules and their responsibilities will be detailed.

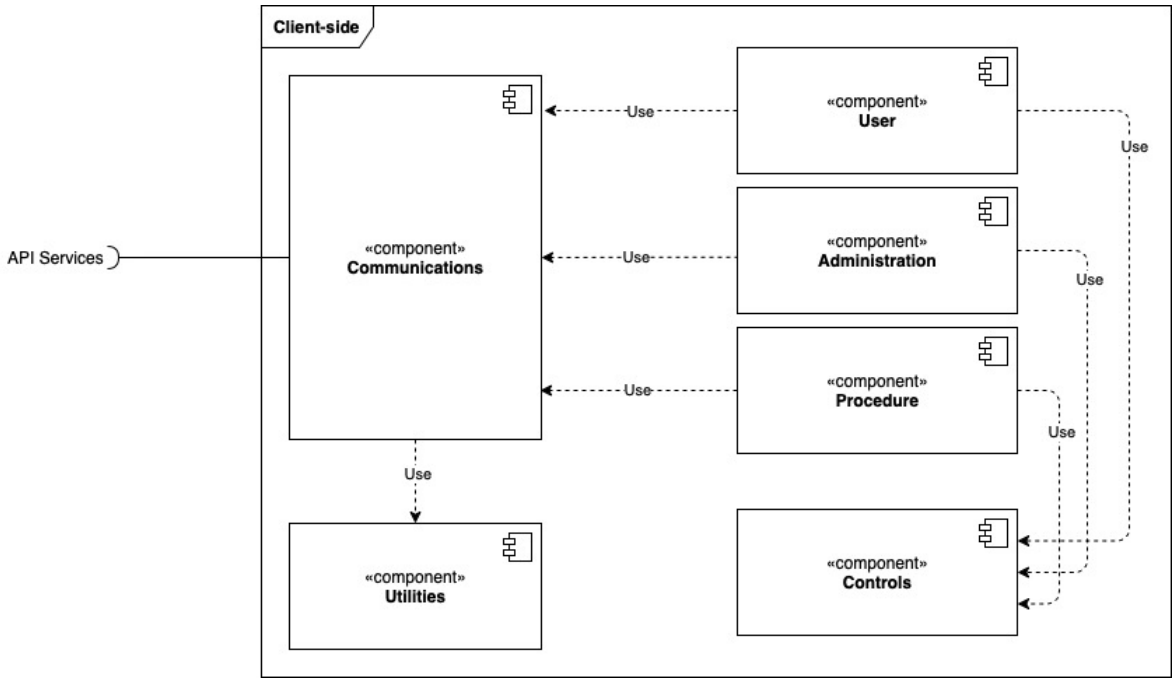



**Figure 4.7:** Client-side component diagram

### 4.3.1 Procedure

As in the application server, the procedure is also the most important component from the client side. It provides the list of procedures that the logged user has access to, and when selecting one, it provides the interface that shows the tables and fields from both databases as rectangles and the mappings as arrows, similarly to the Rabbit in a Hat's UI.

Besides, it is also in this component that all the procedure edition features must take place. This includes altering tables or fields comments, adding or removing mappings, changing their logic, but also retrieving the summary files, managing users with access to the procedure, and even marking it as deleted.

### 4.3.2 User

The user component must be responsible for the profile management, allowing to change the logged user's username and e-mail. It is also in this component that it is possible to visit other users' profiles, and if the user who is visiting it is an administrator, it is possible to give privileges to the visited user.

The interfaces to create an account on the system and to log in must also be provided by the user component.

### 4.3.3 Administration

The administration module provides the interface for all the administrative features. The management of procedures is executed displaying a list with all the procedures, even those which have been marked as deleted. Here, administrators can actually remove them, mark them as visible again and access any of them.

The user management is also performed in this component, allowing the administrators to give privileges to collaborators or delete their accounts.

### 4.3.4 Controls

The controls component contains classes that define generic HTML elements with predefined parameters that are used and configured in multiple other modules. Among the elements are simple buttons, text fields, file inputs, or rectangles to define tables and fields with and without tooltips.

The elements could be defined directly in the classes where they would be necessary. However, keeping them separated increases modularity and maintainability. If some parameter is wrongly defined, it is only required to alter it in the generic class instead of in every occurrence. Furthermore, it allows keeping the action or purpose of the element, in a given context, separated from its layout.

The components that have been detailed until now focus on providing the interface to clients. Combined, they could represent the presentation layer described in the client-server model (Figure 4.1).

### 4.3.5 Utilities

The utilities module is a simple component whose main objective is to provide the correct headers to be sent in the requests to the RESTful API.

There are two types of headers: a simple header for requests that only contain data as parameters and another that is used when it is necessary to also send a file, namely when creating ETL procedures. In both cases, the JWT received after the log in operation is used in the header so that the application server can authenticate the user.

### 4.3.6 Communications

The RESTful API is responsible for providing a list of services to be consumed by the client-side. This component handles the communication with the application server.

The headers described before are used in the requests that are sent. The obtained responses are then sent back to the component which made the request in its original state. The data manipulation is the responsibility of the component that made the request, this one acts as a bridge between the modules that render the layout and the application server.

The utilities and communications modules are equivalent to the communication layer since they have the same purpose and objectives.

## 4.4 Summary

This chapter detailed the system's architecture, defining it as a client-server model with 3-tier architecture. It also described the composition and goals of each tier, focusing on the responsibilities and not on the technologies used in the implementation. It was also presented the system's component diagram where it was possible to have a better comprehension of the system's structure.

The next chapter will present the system's implementation and discuss the adopted technologies to develop each component.

CHAPTER 5

# System's implementation

Chapter 3 defined both the functional and non-functional requirements, and considering it, the system's architecture was defined in chapter 4, focusing on the responsibilities of each component from the client and server-side.

This chapter will focus on the system's implementation, detailing the technological aspect and presenting the tools that were used.

## 5.1 Technological stack

Figure 4.1 represents the system's architecture that follows a client-server model with 3 tiers and multiple layers. Each layer uses a set of programming languages and frameworks to implement the respective responsibilities.

Based on that model, figure 5.1 presents the technologies that were used in each tier and layer.

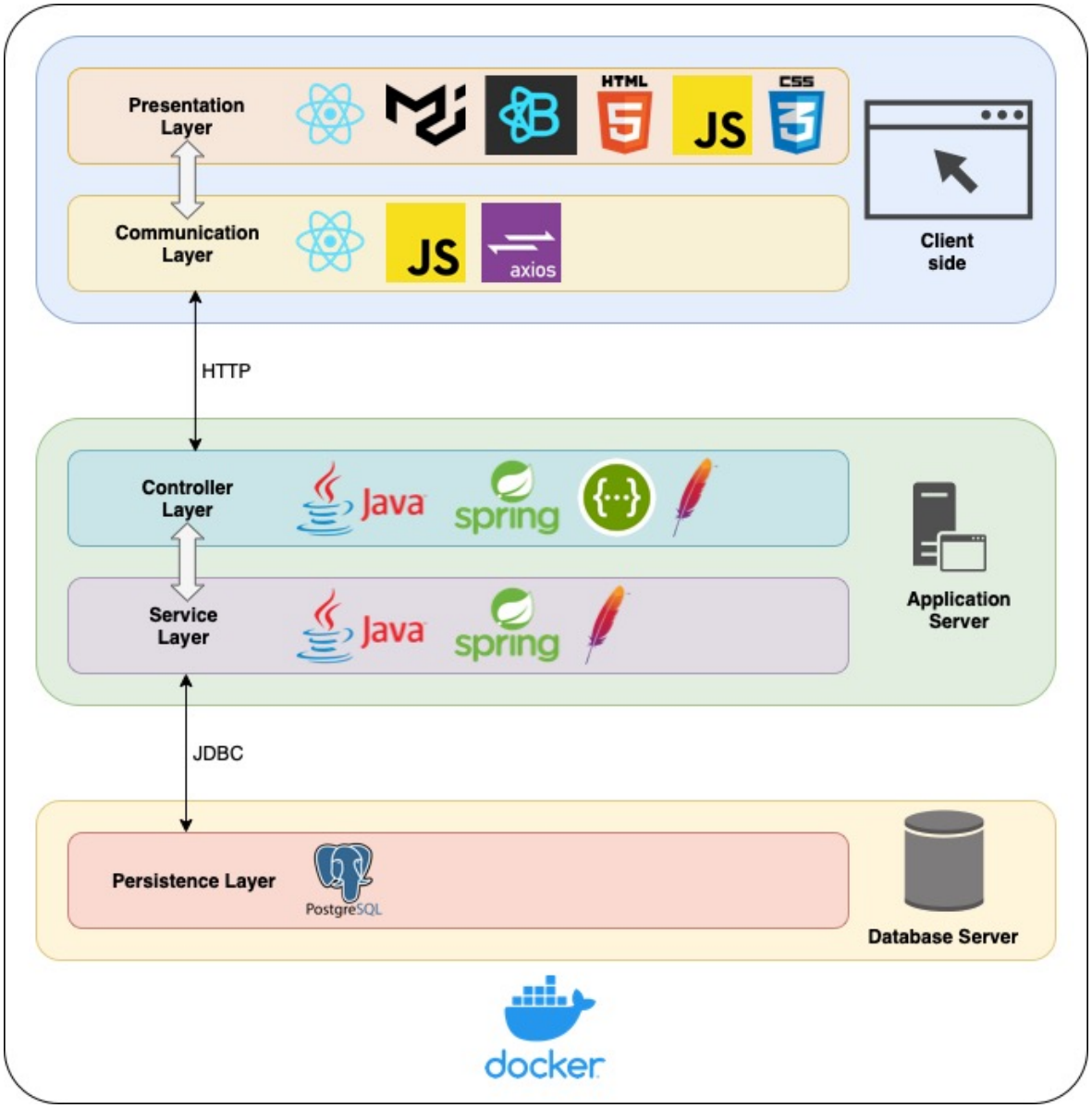



**Figure 5.1:** Languages and frameworks used in system's implementation

These are the technologies that were used to implement the system but were not the only ones. Other tools were used, for example, to apply Continuous Integration/Continuous Delivery (CI/CD) principles. These tools include:

- **Github**[1] - it is a hosting service to store software's source code [46]. The branch workflow was adopted, where for each feature, a new branch is created. When the feature is implemented, a pull request is created, and the code is merged into the master branch.
- **Jenkins**[2] - Jenkins is a continuous integration tool that is connected with the repository [47]. Every push to the master branch triggers the Jenkins pipeline that will be

[1] `https://github.com`
[2] `https://www.jenkins.io`

explained later in this chapter.

- **Google Cloud Platform**[3] - it is a set of cloud computing services provided by Google. It was used to launch two virtual machines: one for the CI/CD server, where Jenkins would be running, and another to deploy the system in the production environment.
- **Nginx**[4] - Nginx is, besides a web server, a software load balancer that handles the request traffic that is sent by the clients to the application server [48].

The next sections will detail the technologies that were used in each tier, discussing the advantages and disadvantages of other possibilities and the reasons that led to choosing the adopted framework or language. Since the database server only uses one technology, it will be aggregated with the explanation of the application server's tools.

## 5.2 Implementation of the server-side

The system server-side includes the application server and the database server. The architecture and the theoretical description of the first one and its modules were detailed in section 4.2. On the other hand, this section will focus mainly on the technologies that were used in those two tiers and the reasons that led to their adoption.

### 5.2.1 Used technologies

The desktop application Rabbit in a Hat was developed using Java. Most of the tools and frameworks used in the server development are based on Java in order to reuse or adapt some of the code already created.

From all the tools that will be presented, the only one that is not based on Java is PostgreSQL which is used for database management.

*Spring Boot*

Spring Boot is one of the frameworks from the Spring environment used to create web applications using Java [49]. This framework allows the usage of all Spring's modules but much more easily and efficiently [50]. It was used to develop the application server structure, including the RESTful API that provides services to the clients and all the business logic and rules.

The advantages of Spring Boot include its auto-configuration that allows having an embedded web server which makes the applications portable [51]. The configuration files, management of dependencies, and the easy connection to various database systems were also critical aspects that led to the choice of this framework. The documentation and bibliography referring to Spring Boot are rich, providing multiple ways of solving the same problem, according to the needs. Another advantage of this framework is the possibility to create

[3] `https://cloud.google.com`
[4] `https://www.nginx.com`

various profiles with different configurations for the various environments (development, production).

Django[5] could also be used to develop the application server since it has a lot in common with Spring Boot. The connection to database management systems is very easy and direct, and since most of the dependencies are already installed in the framework, their management is simpler [52]. The main difference is that Django uses Python as the programming language instead of Java, which can be a disadvantage in some cases, depending on the application and its purpose [53].

The main reason for choosing Spring Boot is because the desktop application Rabbit in a Hat is developed using Java. Most of the logic of reading the input files, writing the output files, managing tables, fields, or mappings was reused or partially adapted to respect the new data model or implement new features.

*Apache Maven*

Spring Boot, described previously, was selected to develop the application server's modules. The next step is to define which automation tool to compile the project will be used.

There are three well-known tools for this purpose: Apache Ant[6], Apache Maven[7], and Gradle[8]. The last is the most recently developed from the three and it is the one with better documentation and online community. However, the Apache tools have a great advantage over Gradle: the number of plugins publicly available is higher [54]. Therefore, the chosen tool was Apache Maven.

It is a building tool for projects developed in Java and what distinguishes it from the other tools is its extensible architecture. It uses the Project Object Model (POM) to describe the project and it is defined in the *pom.xml* file. This file includes the project name, group and artifact ID, and version. Moreover, it is possible to define instructions for the different lifecycles of the project which simplifies the command execution [55].

*PostgreSQL*

PostgreSQL[9] is an open-source RDBMS that uses the SQL language to manage and run queries over the data stored in the database.

Spring Boot can be easily integrated with relational databases, including PostgreSQL and Not only SQL (NoSQL) databases, such as MongoDB[10] or Neo4j[11]. NoSQL databases were not considered to store data; although, they can be more efficient than PostgreSQL when executing queries over large amounts of EHR data [56].

[5] `https://www.djangoproject.com`
[6] `https://ant.apache.org`
[7] `https://maven.apache.org`
[8] `https://gradle.org`
[9] `https://www.postgresql.org`
[10] `https://www.mongodb.com`
[11] `https://neo4j.com`

Other relational databases, such as MySQL[12] and MariaDB[13], were also studied. All of them are open-source RDBMS, and the performance is similar between them [57]. PostgreSQL was chosen because it is a viable framework for the system's implementation.

*OpenAPI Specification*

The OpenAPI Specification (OAS)[14] is a standard interface that allows understanding the behavior of a service provided by a RESTful API [58]. The purpose is to describe what a service does and the impact that its actions have on the system. It specifies the parameters that should be present on the request and the possible responses and respective HTTP codes.

Documenting an API is a good practice when developing software because it makes maintenance easier in the future and, when it is a public API, makes it more easily adoptable.

### 5.2.2 Data model

Figure 4.3, previously presented, is a class diagram that represents an ETL object from the Rabbit in a Hat application. There, it is possible to understand that there will be two database objects, each with a list of tables and these with a list of fields. If the field is from the EHR database, then it is possible to exist a list of value counts; on the other, if it is from the OMOP CDM, then it might contain a list of concepts. Furthermore, the ETL object will very likely have a list of connections between tables and fields.

The system's data model needs to respect these constraints and also consider the existence of users and roles. Figure 5.2 is an entity-relationship diagram that describes the system data model.

[12] `https://www.mysql.com`
[13] `https://mariadb.org`
[14] `https://swagger.io/specification/`

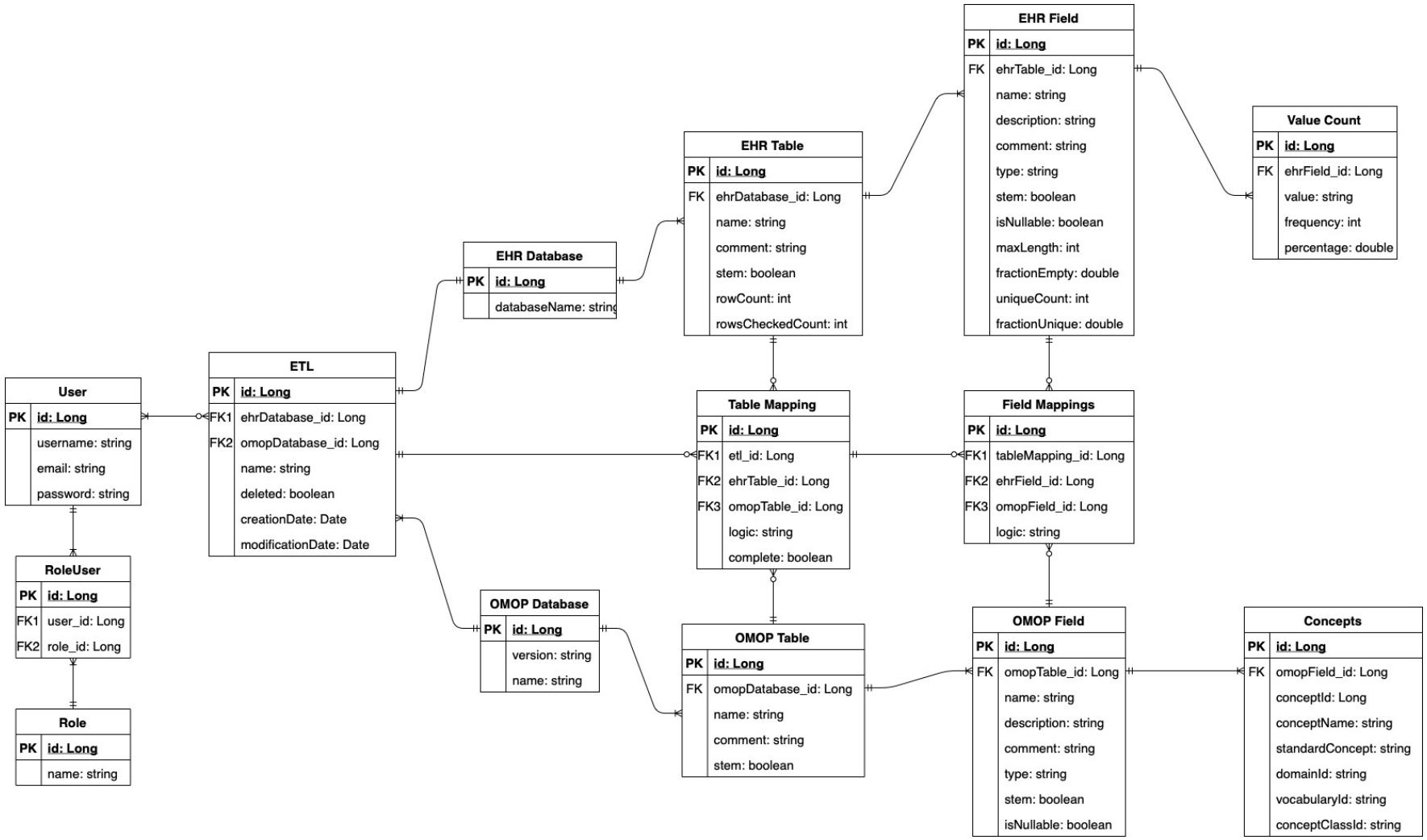



**Figure 5.2:** System's data model

The tables considering the EHR side (*EHR Database*, *EHR Table*, *EHR Field* and *Value Count*) are separated from the tables from the OMOP CDM side (*OMOP Database*, *OMOP Table*, *OMOP Field* and *Concepts*) despite having various fields in common. It is necessary to differentiate value counts from concepts and, therefore, it is not possible to have a generic *Field* class that could be related to both. With two different classes to define fields, it is better to have two classes to define tables and databases because it increases modularity and maintainability.

Besides, if generic classes were used, there would be tables in the database with more entries and with more empty fields, which worsens performance. For example, the table *EHR Table* has two columns that the table *OMOP Table* hasn't, and using a generic table would lead to multiple rows with those two columns empty.

In the next section, the languages and frameworks used in the client-side will be detailed, discussing their advantages and disadvantages while comparing them with other alternatives.

## 5.3 Implementation of the client-side

The client-side, as well as the server-side, are composed of multiple components with well-defined responsibilities. Its main function is to retrieve data from the application server using the services provided by the RESTful API and present it to the users.

The main concern when developing the client-side was to create a visual interface with modern web technologies and frameworks. The focus is on usability: users should easily understand how the application works.

Figure 5.1 presented the technologies that were used in the multiple layers. ReactJS was used to develop the GUI alongside well-known libraries, namely Material-UI. These are the primary tools used in the presentation layer. On the other hand, Axios is the library responsible for handling the communications with the server's API and providing correct data to the presentation layer.

### *ReactJS*

ReactJS is an open-source JavaScript framework well known for its component-based architecture, which allows modularity and reusability. This architecture allows updating data in a child component without reloading the parent component.

ReactJS is not considered a full framework since the number of installed dependencies is lower, making it lightweight compared with other frameworks, namely AngularJS[15] [59]. This characteristic is an advantage since it can increase ReactJS performance over the other frameworks [60].

Furthermore, the online community is broad which increases the number of publicly available libraries and components.

[15] `https://angular.io`

*Material-UI and React Bootstrap*

Material-UI[16] is a components library for ReactJS that provides highly customizable components, including buttons, text fields, or sliders. The components are intuitive, easy-to-use, and highly responsive in order to improve user experience.

React Bootstrap[17] is also an external library that provides components but based on Bootstrap. Despite creating familiar and consistent components, Material-UI is used whenever possible because Bootstrap-based applications are usually heavier and can slow them down [61].

In the application's GUI, React Bootstrap was only used to create the navigation bar since it has a standard layout, even if it is less customizable. The remaining components, such as buttons, or modals, were developed using Material-UI.

*Axios*

The frameworks described before are used in the presentation layer to create the application's user interface. However, to present data to the user, it is necessary to consume the services provided by the application server's RESTful API and handle the response.

Axios[18] is a JavaScript library that works as an HTTP client for browsers. It is responsible for sending requests to the API and providing the responses to the presentation layer that will handle it [62].

*Node.js and NPM*

Node.js[19] is a web server for JavaScript applications that supports asynchronous event modeling [63]. One of the main features of Node.js is its Node Package Manager (NPM) which is an essential tool to manage packages and JavaScript external libraries. NPM is also responsible for the server actions and defining its environment.

ReactJS developers usually use NPM to share their components which make this combination of framework advantageous.

The previous sections have been detailing the technologies that were used in the system's development. However, to be able to use the application, it is necessary to deploy it. This includes orchestrating the configuration, compilation, and building of the projects and database. The following section will discuss the deployment strategy that was implemented and the technologies that were used in that process.

## 5.4 Deployment

Easy deployment is one of the non-functional requirements defined and discussed in section 3.2. The goal is that with a few commands, the system is up and running with no further

[16] `https://material-ui.com/pt/`
[17] `https://react-bootstrap.github.io`
[18] `https://axios-http.com`
[19] `https://nodejs.org/en/`

configurations and, if possible, hiding the technical issues of installation from the users.

The system needs to be deployed in a machine with physical resources (hardware). However, deploying it directly in the machine requires adapting the deployment steps to the OS. Virtualization is a technology that allows running services using resources but with abstraction on the physical layer, i. e., on the hardware.

It can be achieved using a VM, where the machine's hardware is emulated to the guest OS as if it exists [64]. However, as represented in Figure 5.3, the architecture of a VM can reduce speed and performance because of the multiple layers below the application layer and the couplings between them [65].

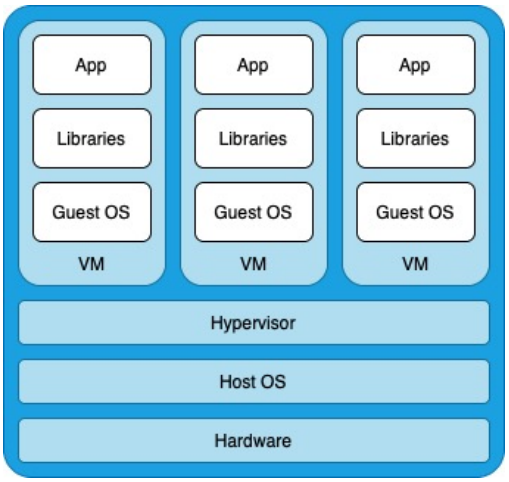



**Figure 5.3:** VM architecture

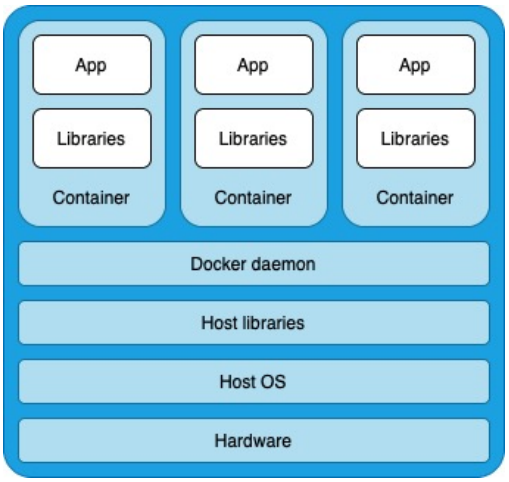



**Figure 5.4:** Container architecture

An alternative to VMs is containers that share the kernel with the machine that is hosting them. This tight integration with the host's OS reduces the overhead observed in VMs [66]. Figure 5.4 represents the architecture of a container running on a host.

As observed above, containers can share hardware and software resources, namely host libraries, with the host. This, allied with the usage of the host kernel, reduces the container size, especially when compared with VM's size, and allows a faster set up and initialization.

Docker[20] is an open-source technology that allows containers virtualization [65]. Docker containers hold applications with all the necessary dependencies to run correctly [67]. It is a suitable solution for systems that follow a micro-service approach where each service runs individually in a container [66]. Since this system's architecture is based on this approach, Docker seems a proper tool for the application deployment.

The system uses multiple Docker images in the deployment, as explained below:

- **PostgreSQL** - image pulled directly from Docker Hub[21], the Docker's official image repository. The connection configurations are previously defined, making the database management and the installation process easier for the user.

[20] `https://www.docker.com`
[21] `https://hub.docker.com`

- **Backend** - image with the application server built using a Dockerfile that contains a set of steps for pulling the Apache Maven and Java images, installing necessary dependencies, defining the profile, and setting up the application.
- **Frontend** – Similar to the application server, a Dockerfile is used to build the client application. In the first place, it pulls and installs Node.js, then installs all dependencies and libraries and starts the application.
- **Nginx** – React provides a web server for development and a better one for production. However, Nginx is used because, besides being a web server, it is also a load balancer and allows the launch of new instances of the client application if one is overloaded.

All these containers need to start in a specific order and communicate between them for system's good functioning. Docker Compose[22] is used for the containers orchestration and to define the environment variables for each container, namely the database authentication variables. This way is possible to set up all or some of the containers in the proper order with just one command. Figure 5.5 is a deployment diagram that represents schematically the containers described above and how they are connected and communicate with each other.

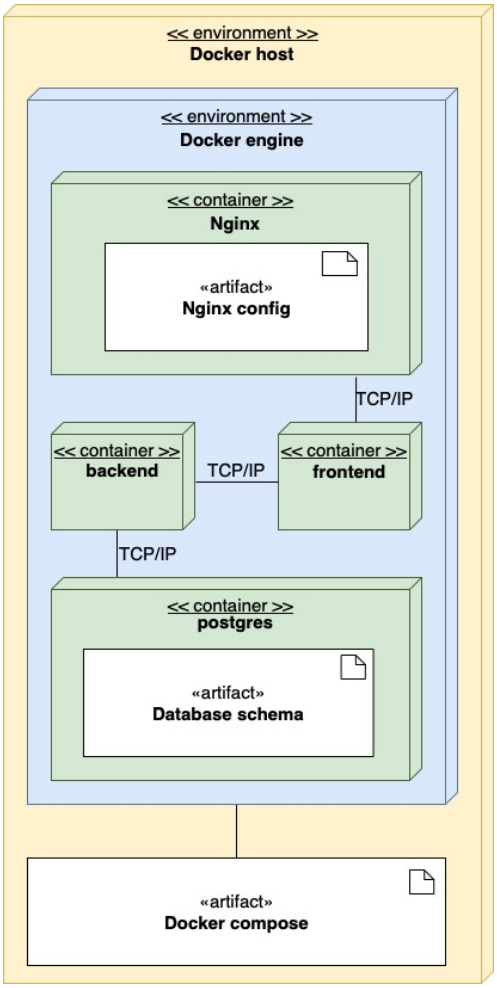



**Figure 5.5:** System's deployment diagram

[22] `https://docs.docker.com/compose/`

Jenkins, an automation server, was used to automate the deployment process. It is responsible to execute tasks related to software building, testing, and deploying and to manage the operations until the application deployment in the production environment.

The server is connected to the application's Github repository so that every time there is a push to the master branch, Jenkins automatically checkouts the code and executes the instructions contained in the Jenkinfile.

First, it builds the server project and pushes the artifact to the artifact repository. After that, Docker images are created for both server and client projects and pushes them to the Docker registry. Finally, in the VM where the application will be running in the production environment, the Docker images are pulled and deployed. Figure 5.6 represents the pipeline described above.

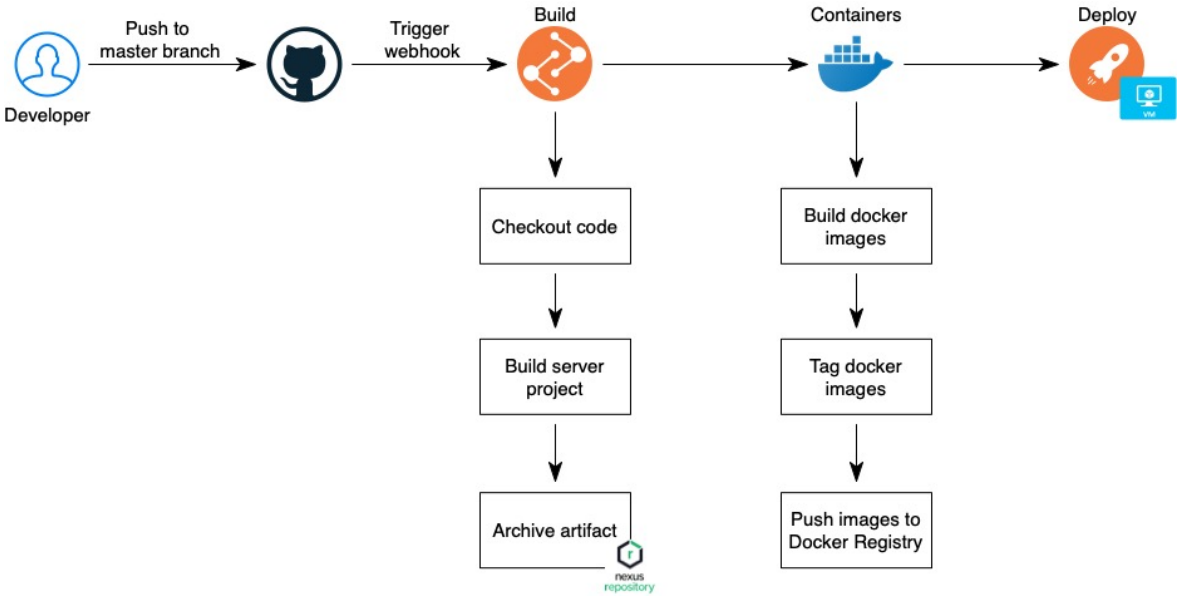



**Figure 5.6:** Jenkins pipeline for CI/CD

The next section will present the system's GUI and discuss the operations possible to do in each component. It will also explain the visual differences according to the roles of the logged user.

## 5.5 User interface overview

The GUI provides a set of separators and workspaces to meet user's needs and to respect the different roles. Since the administrators can do all the tasks that a collaborator can, some of the pages are similar for both roles, differing only on the content that is displayed. One example of that is a page for the list of ETL procedures that a user has access to. However, administrators have a separate but identical page with a list of all procedures with more detail. This section aims to show and describe the possible actions present in each component of the user interface.

The first component presented to users when accessing the application is the home page. It is a simple page with little information that only allows the user to log in and sign up because it is required authentication to access the remaining interface components. It gives some context about the system's goal and insights about the features.

After logging into the application, it is displayed a list of ETL procedures which the logged user has access to. Moreover, the content in the navigation bar changes, providing a set of separators to where is possible to be redirected. This includes redirections to the user's profile page, and a button to log out from the application, besides to the page with the list of procedures (Figure 5.7). On all the remaining interface components is possible to use the navigation bar to be redirected to other pages.

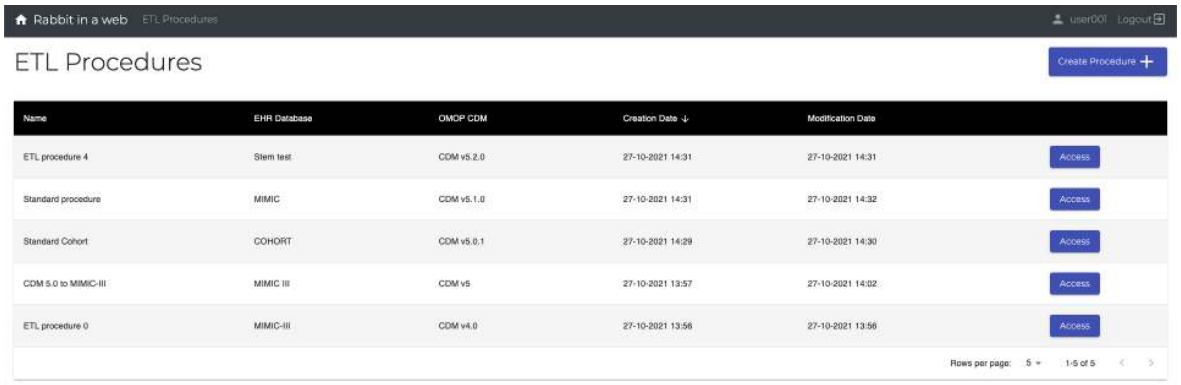
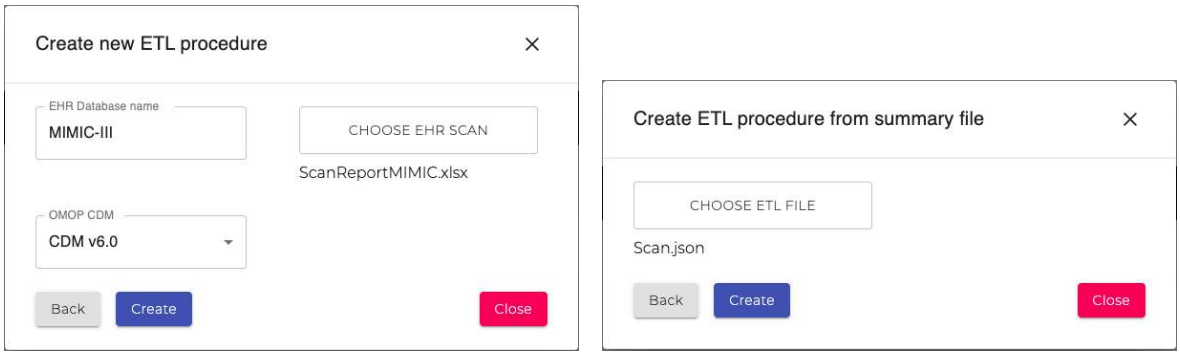



**Figure 5.7:** Collaborator's list of ETL procedures

Only when an administrator is logged in the system, as in Figure 5.9, an additional section appears in the navigation bar that gives access to the components that deal with administration features.

The table layout is also used in other interface components when the goal is identical: provide a list of elements with multiple attributes that can be ordered by some of them. Initially, cards were used to give information about ETL procedures, but tables provide a better organization, and it is easier to sort by any attribute.

Furthermore, it is also on this page where is possible to create new procedures. As said before, there are two ways of creating a procedure, so when a user tries to create one, the two possibilities are shown: using a database scan from the EHR database created with the White Rabbit tool (Figure 5.8a) or using a summary file from other procedure generated in the application (Figure 5.8b).




**(a)** Create procedure using the database scan **(b)** Create procedure using the summary file

**Figure 5.8:** Possibilities to create ETL procedures

One of the operations only accessible to administrators is the retrieval of all ETL procedures. The layout presented in Figure 5.9 is very similar to the one observed in Figure 5.7, but with more displayed attributes, namely, if the procedure has been deleted by users (but not removed from the database), the list of users that have access to the procedure and a button to delete it from the database.

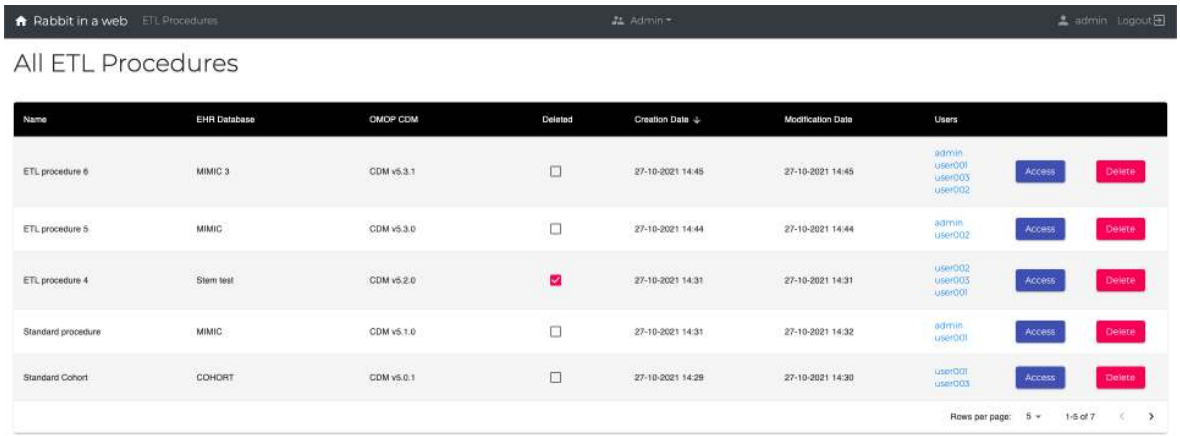



**Figure 5.9:** List of all ETL procedures

The goal in this component is that, despite having more displayed attributes, the administrator could have a clean page with only the important information to properly manage the procedures. Moreover, it is important that in a few steps, the administrator can do the administrative tasks, namely change the procedure's deletion status, and make it visible again to the collaborators or completely remove it from the system.

All the details inside an ETL procedure are accessible to administrators and it can be edited by them. It is better to provide a faster way to access a procedure if necessary, such as a button, than to provide information that could be avoidable.

The other operation only allowed to administrators is the users' management. Also in this component, a table is used to provide information about users including the username, the e-mail, and the privileges. However, in this case, is only possible to sort to show administrators in first or last place. The main difference to the previous tables described before is the

searching bar that allows to only display a list of users whose username or e-mail match or contains the searching text (Figure 5.10).

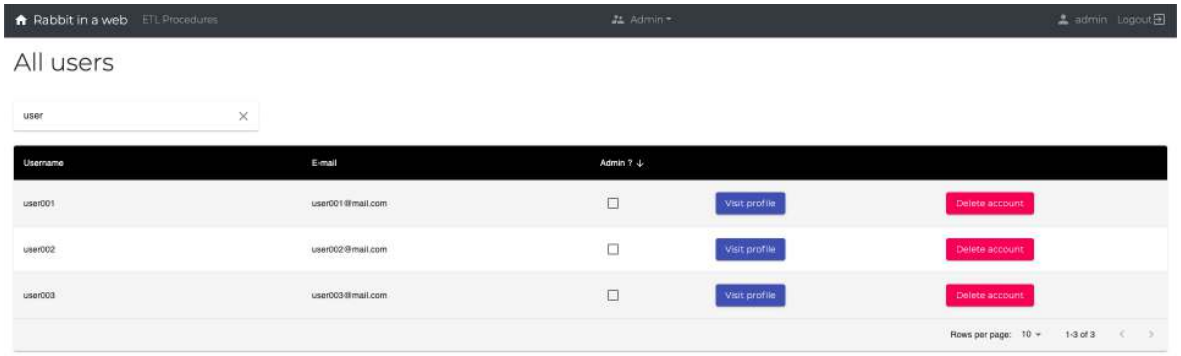



**Figure 5.10:** List of users

One of the remaining interface components is the profile page. This is a simple component whose main goal is to provide information about a user. However, the content displayed and the possible operations are different for each role and if the user is visiting its profile or another user's profile.

When a collaborator is logged in the system and visiting its profile page, as in Figure 5.11, it is possible to edit the username and e-mail. It is also provided a list of the 10 most recently modified ETL procedures in a table with a simpler layout since the goal is to give a quick summary of recent activity.

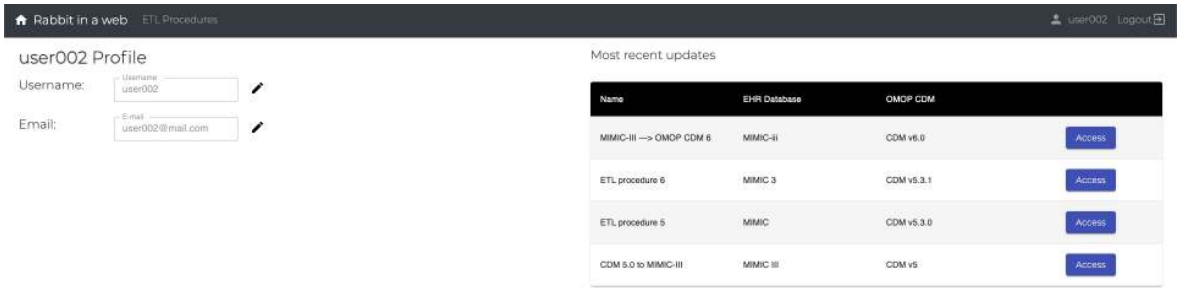
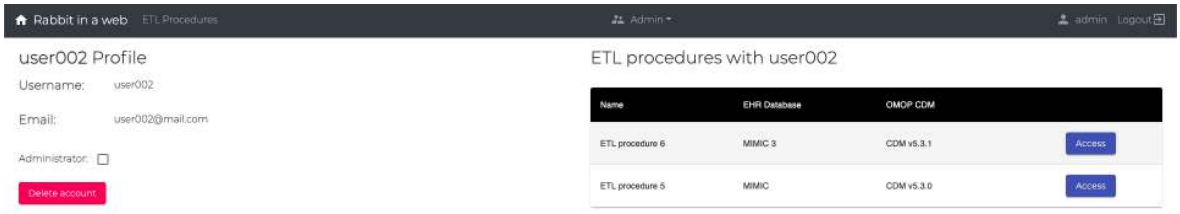


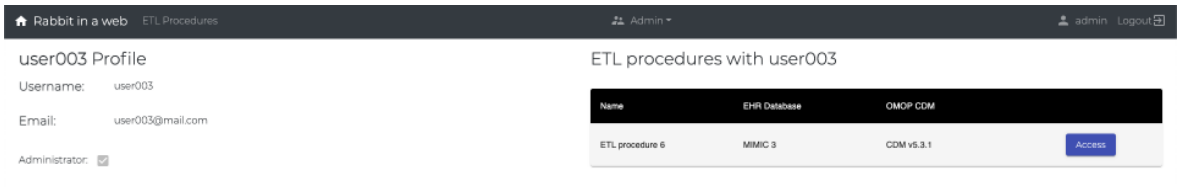

**Figure 5.11:** Logged user's profile page

On the other hand, if another user's profile is being visited, it is given the username and e-mail, with no edition features, and a similar table but with the 10 most recently modified ETL procedures that both the logged and visited users have access to.

Instead, if an administrator is logged into the application, additional information and operations are shown. Figure 5.12a represents the situation where an administrator visits a collaborator's profile. It is provided a checkbox to transform the visited user into an administrator and a button to remove the user from the system. However, the checkbox becomes disabled and checked if the administrator visits another's profile or if he grants privileges to the collaborator. The delete account button also disappears which means that is not possible to remove privileges from other administrators (Figure 5.12b).




**(a)** Administrator visiting collaborator's profile




**(b)** Administrator visiting administrator's profile

**Figure 5.12:** Administrators visiting other users profiles with different roles

The interface's main component is the ETL procedure page. It contains all the information about tables, fields, and mappings between tables and fields. When designing this component, one of the main concerns was to keep it similar to the original application, Rabbit in a Hat. Users that are used to the desktop application should be able to perform multiple operations without difficulty in this new environment.

The main similarities are that boxes represent tables and fields, and arrows create connections that start on boxes located on the left side (EHR database) and end on the right side (OMOP CDM).

However, to implement some features and to improve usability, some aspects are different than the original application. At first sight, the layout is identical to the Rabbit in a Hat's GUI, but some elements were added to implement features, including:

- Text fields to edit the name of the ETL procedure and the EHR database's name.
- Options menu that contains management features, namely, manage users, obtain summary files, add or remove stem tables, save procedure in a file or mark it as deleted.
- Dropdown to select the OMOP CDM version to use.

The main differences are visible when starting to change the procedure. When a box that represents a table is selected, as Figure 5.13 shows, the color of the remaining boxes from that database becomes lighter. This usability feature enhances the selected box, visually separating it from the others. Moreover, it helps to identify which database has a table selected. Besides, a table containing information on the fields, such as their name, data type, and a brief description, is displayed, as well as a text input to edit the table's comment.

The arrows' color also changes considering the selected box. If a table from the EHR database is selected, all the arrows that start on that table become orange. Likewise, if a box located on the OMOP CDM side is selected, the arrows that end on it change the color to blue. All the other arrows become lighter.

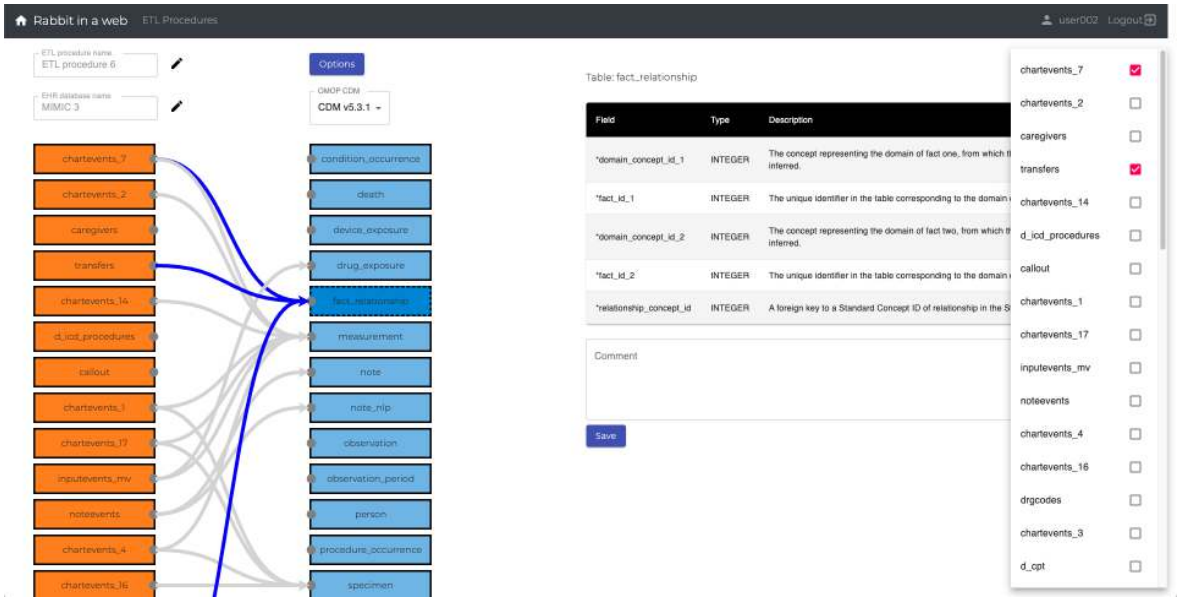



**Figure 5.13:** Content displayed when selecting table *fact_relationship* from the OMOP CDM v5.3.1

One of the goals in this component is to provide multiple ways to perform some of the operations, and, therefore, increase usability. The arrows that connect boxes can be created using one of the three following possibilities:

- Select a box from the left side and then another from the right side.
- Drag the connection points (grey circles) from a left side box to one on the right side.
- Using the dropdown that contains the table names from the other database. This dropdown also allows removing mappings.

These connections between tables need to be explored and edited to map the fields to their standard definition. The idea behind the usability to edit table mappings is that for simple operations, namely change its logic, mark it as complete or remove it, it should be simple and direct. Figure 5.14 represents a table mapping selection when the user clicks only once over an arrow. Besides the elements to perform the previous operations, the selected row changes its color to red to enhance it from the remaining.

It is also possible to observe that some arrows have black color instead of the normal grey, which means that the respective fields are mapped to their standard definition in the OMOP CDM. Therefore, it is considered that the mapping between those tables is complete.

However, to manage the connections between fields inside a table mapping, it is necessary to use another interface component. This should be very similar, including boxes to represent fields and arrows to connect them, but visually, the user should be able to understand the changes and distinguish both situations without difficulty.

When double-clicking over an arrow that connects tables, the panel to connect fields becomes visible. The tables connected in that mapping are displayed immediately below the text field to change the EHR database name and the dropdown to select the version of the OMOP CDM. The fields of each table are also represented as rectangular boxes but with a lighter color to distinguish them from the respective table.

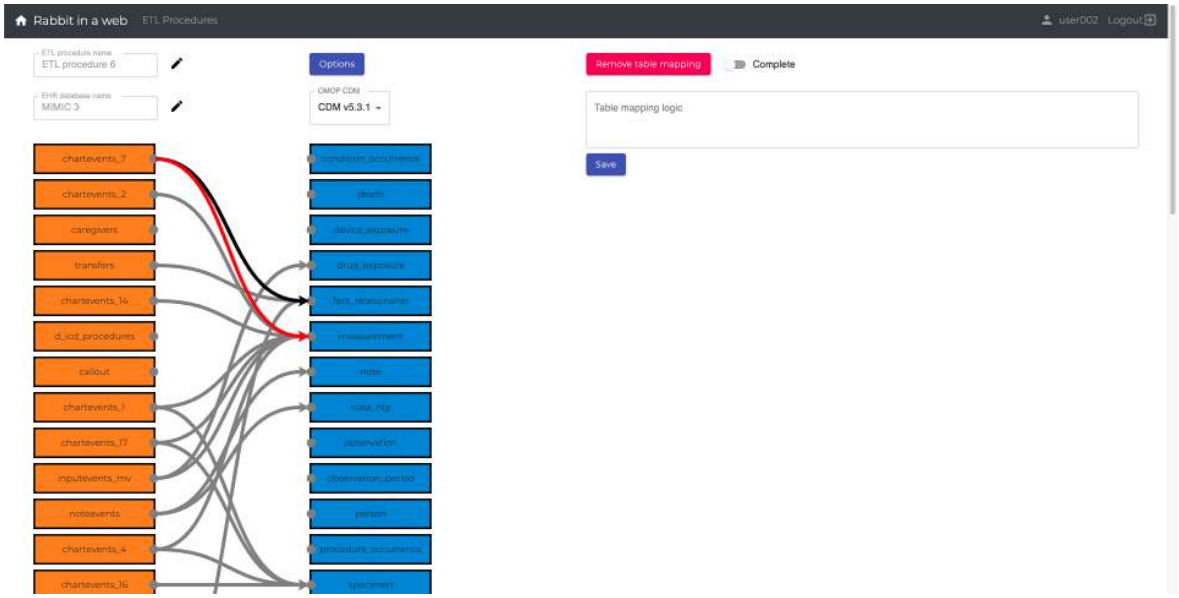



**Figure 5.14:** Mapping between tables *chartevents_7* and *measurement*

On the right side, the same elements to perform the operations related to the table mapping as the ones described before are also visible. These elements are always visible so that the collaborators also have a way to manage the table mapping in this component.

The content on the right side of the component only changes when a box or a mapping is selected. In the first case, all the arrows that start or end on that box, depending on if the field is from the EHR database or the OMOP CDM, become orange or blue, respectively, and the other arrows take a lighter tonality. Some of the fields on the OMOP CDM represent standard concepts and it is important to show them to the collaborator. Figure 5.15 represents the interface component when the table *note_type_concept_id* is selected.

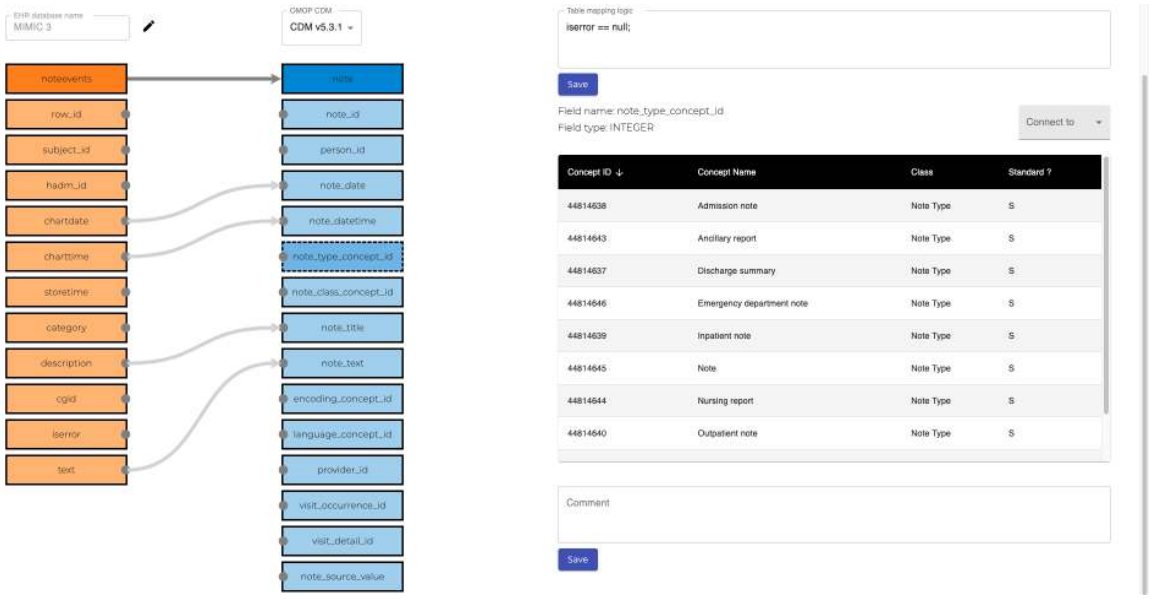



**Figure 5.15:** Concepts of field *note_type_concept_id*

On the other hand, when an arrow is selected, it changes its color to red, and a button to delete the field mapping as well as a text input to edit the field mapping logic become visible (Figure 5.16).

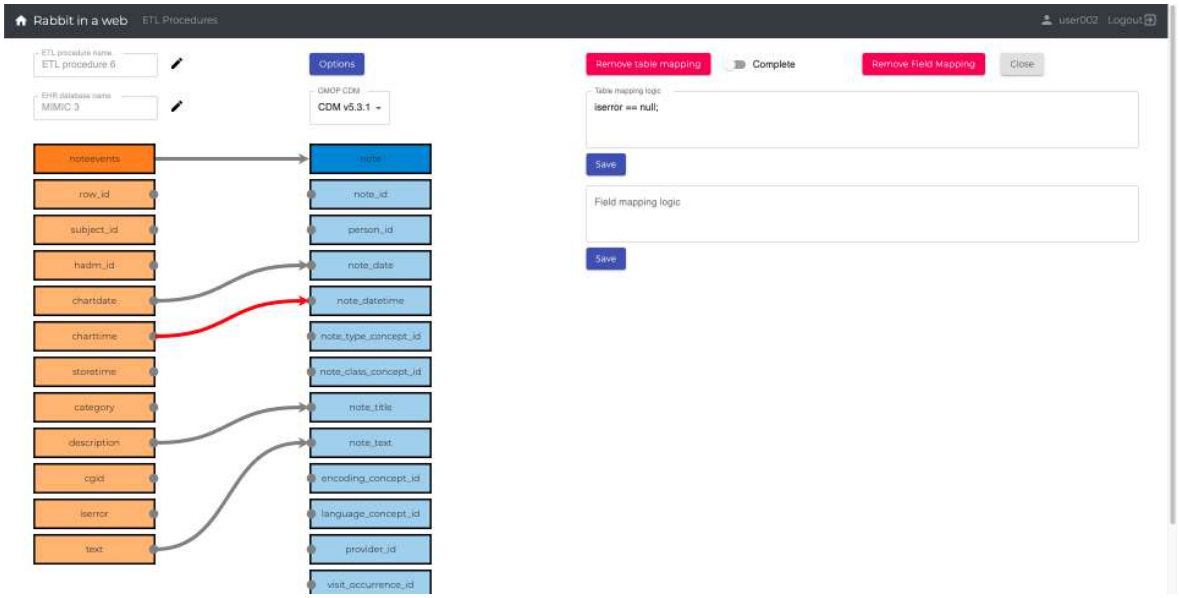



**Figure 5.16:** Field mapping selection between fields *charttime* and *note_datetime*

## 5.6 Summary

This chapter aimed to describe the system's implementation, explaining the reasons that led to the usage of adopted tools and frameworks. It also explored the deployment strategy defining which containers were used and discussed the implementation of a CI/CD pipeline, using Jenkins to automate the building and deployment steps. Finally, it was presented the application's interface, showing images of the main components and the usability features that were implemented.

The system's code is hosted in a public Github repository[23] that counts with more than 200 commits. It was adopted a feature branch workflow where for each feature, a new branch is created and when its development is complete, a pull request is created to merge the code to the master branch.

The repository contains the Spring Boot project for the application server and the Node.js project for the application's interface. Moreover, all the files to deploy the system, as well as the Jenkinsfile, are also available in the repository.

[23] `https://github.com/pedrodlmatos/Rabbit-in-a-web`

CHAPTER 6

# Conclusion

The study of the state-of-the-art tools to map data stored in EHR databases to its standard concepts allows us to conclude there is a gap in this subject. Besides, the lack of interoperability between EHR systems makes the usage of multiple sources a difficult task.

The best approach to map data to a CDM, specifically the OMOP CDM, is to create a new application based on the existing one, Rabbit in a Hat.

## 6.1 Final considerations

The focus of this dissertation is the development of a pipeline to migrate data from EHR databases to their standard definition in the CDM, specifically the OMOP CDM.

The first objective was the development of a web application that would implement collaborative features. This would allow having multiple users, possibly in different locations, working in the same procedure simultaneously, therefore fixing one of the main issues in the existing applications. Furthermore, the other goal was the integration of the developed system with an ETL framework to directly migrate data from one database to another, without writing SQL instructions to do that.

The previous chapters detailed the system's conception from its requirements analysis until its implementation. In all chapters, the followed approaches are examined and other alternatives are also presented, discussing the reasons that led to the adoption of one technology instead of the others.

The first goal related to the system's conception and development was achieved. The lack of collaborative features was a major issue in other applications, namely in Rabbit in a Hat. The system allows to have multiple registered users and involved in an ETL procedure and there are multiple roles to allow user's management. Moreover, the application's GUI doesn't have the issues present on the Rabbit in a Hat's interface but maintains a similar layout. Not only the similarity between the GUIs but also between the summary files were considered so that users that would adopt this solution could better comprehend the application's functioning.

However, it was not possible to integrate an ETL framework with the system. In an initial phase, multiple frameworks were studied, and Pentaho Data Integration was considered due to its Java API.

Since the first step of the development process, one of the main concerns was to use complete frameworks with proofs of good results from the community. Spring Boot is the most used Java framework to create RESTful APIs and ReactJS, alongside Angular, are the most used tools to create user interfaces for web applications.

Furthermore, it was important to adopt good practices during the development, namely following the CI/CD approach, where the code changes were validated before merging them with the previous version of the application. However, one of the most important steps on the CI/CD pipeline, the testing stage, was not implemented. The tests would validate changes and verify if methods were correctly developed. This stage would include unit and integration tests on the server side and usability tests on the client side.

## 6.2 Future work

Some of the dissertation's goals weren't achieved therefore, in the future, the main feature to implement is the system's integration with an ETL framework. This would include creating all the rules to cast data to the desired data type, define into which standard concepts the data is mapped.

Another feature that would be interesting to implement is directly read the database structure when creating a procedure. White Rabbit, another OHDSI tool, already accesses the database and creates the file with its information about tables, fields, and relations between them. Integrate this feature in the system would allow having another and more direct way to create an ETL procedure and reducing the number of tools necessary to use the application properly.

The application currently only considers the OMOP CDM as the destination of the migrated data. The adoption of other CDMs, such as the Sentinel Common Data Model, could make the application more generalizable, and more users could use it. The files that contain all the information about the different versions of the OMOP CDM are simple CSV files where each row describes a field. The other CDMs follow a similar relational structure with tables and fields, so creating this type of file would not be a difficult task and would make the application more comprehensive.

The Jenkins pipeline deploys the system in the production environment with all the correct configurations for all the services. However, the container's scalability was not tested to prevent possible bottlenecks in situations of high demand. The existence of load balancers and replicas for each service would make the system more robust and available even in peak moments.

# References


[1] C. S. Kruse, A. Stein, H. Thomas, and H. Kaur, "The use of Electronic Health Records to Support Population Health: A Systematic Review of the Literature," *Journal of Medical Systems*, vol. 42, no. 11, pp. 203–214, Sep. 2018.

[2] K. Thiru, A. Hassey, and F. Sullivan, "Systematic review of scope and quality of electronic patient record data in primary care," *BMJ*, vol. 326, no. 7398, Jun. 2003.

[3] OHDSI, *The Book of OHDSI*, 1st ed. 2021.

[4] *ISO/TR 20514:2005 Health informatics — Electronic health record — Definition, scope and context*, Oct. 2005. [Online]. Available: `https://www.iso.org/standard/39525.html`.

[5] I. Maglogiannis, "Towards the Adoption of Open Source and Open Access Electronic Health Record Systems," *Journal of Healthcare Engineering*, vol. 3, no. 1, pp. 141–161, Mar. 2012.

[6] H. Kim, P. C. Dykes, D. Thomas, L. A. Winfield, and R. A. Rocha, "A closer look at nursing documentation on paper forms: Preparation for computerizing a nursing documentation system," *Computers in Biology and Medicine*, vol. 41, no. 4, pp. 182–189, Apr. 2011.

[7] C. A. Caligtan and P. C. Dykes, "Electronic health records and personal health records," *Seminars in Oncology Nursing*, vol. 27, no. 3, pp. 218–228, Aug. 2011.

[8] A. Hoerbst and E. Ammenwerth, "Electronic health records: A systematic review on quality requirements," *Methods of Information in Medicine*, vol. 49, no. 4, pp. 320–336, Jul. 2010.

[9] P. R. Rosenbaum, "Observation Study," in *Encyclopedia of Statistics in Behavioral Science*, B. S. Everitt and D. C. Howell, Eds. Chichester, UK: John Wiley & Sons, 2005, pp. 1451–1462.

[10] S. Yang, F. Hadiji, K. Kersting, S. Grannis, and S. Natarajan, "Modeling Heart Procedures from EHRs: An Application of Exponential Families," in *IEEE International Conference on Bioinformatics and Biomedicine*, Kansas City, MO, USA, 2017, pp. 491–497.

[11] J. M. Overhage, P. Ryan, C. Reich, A. Hartzema, and P. Stang, "Validation of a common data model for active safety surveillance research," *Journal of the American Medical Informatics Association*, vol. 19, no. 1, pp. 54–60, Dec. 2011.

[12] IEEE, *Standards glossary*, Sep. 2016. [Online]. Available: `https://www.standardsuniversity.org/article/standards-glossary/#I`, Accessed: 2021-08-09.

[13] K. Häyrinen, K. Saranto, and P. Nykänen, "Definition, structure, content, use and impacts of electronic health records: A review of the research literature," *International Journal of Medical Informatics*, vol. 77, no. 5, pp. 291–304, May 2008.

[14] T. Benson and G. Grieve, *Principles of Health Interoperability: SNOMED CT, HL7 and FHIR*, 3rd ed. New York City, NY, USA: Springer International Publishing, 2016.

[15] L. Min, Q. Tian, X. Lu, and H. Duan, "Modeling EHR with the openEHR approach: An exploratory study in China," *BMC Medical Informatics and Decision Making*, vol. 18, no. 1, pp. 1–15, Aug. 2018.

[16] F. Hak, D. Oliveira, N. Abreu, P. Leuschner, A. Abelha, and M. Santos, "An OpenEHR Adoption in a Portuguese Healthcare Facility," *Procedia Computer Science*, vol. 170, pp. 1047–1052, Jan. 2020.

[17] openEHR Foundation, *openEHR - Specifications Start Page*, 2021. [Online]. Available: `https://specifications.openehr.org`, Accessed: 2021-09-15.

[18] D. Tarenskeen, R. van de Wetering, R. Bakker, and S. Brinkkemper, "The Contribution of Conceptual Independence to IT Infrastructure Flexibility: The Case of openEHR," *Health Policy and Technology*, vol. 9, no. 2, pp. 235–246, Apr. 2020.

[19] G.-H. Ulriksen, R. Pedersen, and G. Ellingsen, "Infrastructuring in Healthcare through the OpenEHR Architecture," *Computer Supported Cooperative Work*, vol. 26, no. 1-2, pp. 33–69, Apr. 2017.

[20] O. Bodenreider, R. Corner, and D. J. Vreeman, "Recent Developments in Clinical Terminologies - SNOMED CT, LOINC, and RxNorm," *Yearbook of Medical Informatics*, vol. 27, no. 1, pp. 129–139, Aug. 2018.

[21] G. C. Bowker, "The History of Information Infrastructures: The Case of the International Classification of Diseases," *Information Processing & Management*, vol. 32, no. 1, pp. 49–61, Jan. 1996.

[22] O. Bodenreider, "The Unified Medical Language System (UMLS): integrating biomedical terminology," *Nucleic Acids Research*, vol. 32, no. 1, pp. D267–D270, Jan. 2004.

[23] R. Saripalle, M. Sookhak, and M. Haghparast, "An interoperable UMLS terminology service using FHIR," *Future Internet*, vol. 12, no. 11, Nov. 2020.

[24] L. Rasmy, F. Tiryaki, Y. Zhou, Y. Xiang, C. Tao, H. Xu, and D. Zhi, "Representation of EHR data for predictive modeling: A comparison between UMLS and other terminologies," *Journal of the American Medical Informatics Association*, vol. 27, no. 10, pp. 1593–1599, Oct. 2020.

[25] A. P. Reimer and A. Milinovich, "Using UMLS for electronic health data standardization and database design," *Journal of the American Medical Informatics Association*, vol. 27, no. 10, pp. 1520–1528, Oct. 2020.

[26] D. Bender and K. Sartipi, "HL7 FHIR: An Agile and RESTful Approach to Healthcare Information Exchange," in *26th IEEE International Symposium on Computer-Based Medical*, Porto, Portugal, 2013, pp. 326–331.

[27] T. M. V. Novo, "Arquitetura para Integração e Exploração de Registos Eletrónicos de Saúde," M.S. thesis, Universidade de Aveiro, Aveiro, Portugal, 2016.

[28] T. A. Majchrzak, T. Jansen, and H. Kuchen, "Efficiency evaluation of open source ETL tools," in *Proceedings of the ACM Symposium on Applied Computing*, Taichung, Taiwan, 2011, pp. 287–294.

[29] S. Vyas and P. Vaishnav, "A comparative study of various ETL process and their testing techniques in data warehouse," *Journal of Statistics and Management Systems*, vol. 20, no. 4, pp. 753–763, Nov. 2017.

[30] A. S. Pall and J. S. Khaira, "A comparative review of Extraction, Transformation and Loading tools," *Database Systems Journal*, vol. 4, no. 2, pp. 42–51, Jul. 2013.

[31] M. B. Biplob, G. A. Sheraji, and S. I. Khan, "Comparison of Different Extraction Transformation and Loading Tools for Data Warehousing," in *International Conference on Innovations in Science, Engineering and Technology*, Chittagong, Bangladesh, 2018, pp. 262–267.

[32] Observational Health Data Sciences and Informatics, *OHDSI - Observational Health Data Sciences and Informatics*, 2021. [Online]. Available: `https://www.ohdsi.org`, Accessed: 2021-10-24.

[33] OHDSI, *White Rabbit*, 2021. [Online]. Available: `http://ohdsi.github.io/WhiteRabbit/WhiteRabbit.html`, Accessed: 2021-10-24.

[34] N. Paris and A. Parrot, "MIMIC in the OMOP Common data model," medRxiv, Tech. Rep., 2020.

[35] B. Chen, H.-P. Hsu, and Y.-L. Huang, "Bringing Desktop Applications to the Web," *IT Professional*, vol. 18, no. 1, pp. 34–40, Jan. 2016.

[36] Y. G. Gutierrez and L. A. M. Chile, "Kelluntekun: Rich Internet Application for Collaborative Snippet Management, Using Oows2.0 and Vaadin," in *International Conference of the Chilean Computer Science Society*, Talca, Maule, Chile, 2014, pp. 107–115.

[37] M. Goeminne and T. Mens, "Towards a survival analysis of database framework usage in Java projects," in *IEEE International Conference on Software Maintenance and Evolution*, Bremen, Germany, 2015, pp. 551–555.

[38] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, 3rd ed. Upper Saddle River, NJ, USA: Addison-Wesley, 2017.

[39] R. S. Sandhu, "Role-based Access Control," *Advances in Computers*, vol. 46, pp. 237–286, 1998.

[40] D. Merkel, "Docker: lightweight Linux containers for consistent development and deployment," *Linux journal*, vol. 2014, no. 239, Mar. 2014.

[41] X. Wan, X. Guan, T. Wang, G. Ba, and B.-Y. Choi, "Application deployment using Microservice and Docker containers: Framework and optimization," *Journal of Network and Computer Applications*, vol. 119, pp. 97–109, Oct. 2018.

[42] B. Frain, *Responsive Web Design with HTML5 and CSS3*, 1st ed. Birmingham, UK: Packt Publishing, 2012.

[43] D. Garlan, "Software Architecture," in *Encyclopedia of Software Engineering*, J. J. Marciniak, Ed. Chichester, UK: John Wiley & Sons, 2002.

[44] H. S. Oluwatosin, "Client-Server Model," *IOSR Journal of Computer Engineering*, vol. 16, no. 1, pp. 67–71, Feb. 2014.

[45] M. Jones, B. Campbell, and C. Mortimore, "JSON Web Token (JWT) Profile for OAuth 2.0 Client Authentication and Authorization Grants," Internet Engineering Task Force, Standard, 2015.

[46] Github, Inc., *Github*, 2021. [Online]. Available: `https://github.com`, Accessed: 2021-10-15.

[47] J. F. Smart, *Jenkins: The Definitive Guide*, 1st ed. Sebastopol, CA, USA: O'Reilly Media, Inc., 2011.

[48] R. Sone, "Chapter 8: Load Balancing with Nginx," in *Nginx*, 1st ed. New York City, NY, USA: Apress, 2005, pp. 153–171.

[49] M. Gajewski and W. Zabierowski, "Analysis and Comparison of the Spring Framework and Play Framework Performance, Used to Create Web Applications in Java," in *International Conference on Perspective Technologies and Methods in MEMS Design*, Polyana, Ukraine, 2019, pp. 170–173.

[50] K. S. P. Reddy, *Beginning Spring Boot 2, Applications and Microservices with the Spring Framework*, 1st ed. New York City, NY, USA: Apress, 2017.

[51] F. Gutierrez, *Pro Spring Boot*, 1st ed. New York City, NY, USA: Apress, 2016.

[52] D. Rubio, "Chapter 1: Introduction to the Django Framework," in *Beginning Django*, 1st ed. New York City, NY, USA: Apress, 2017, pp. 1–29.

[53] L. R. Abbade, M. A. A. da Cruz, J. P. C. Rodrigues, P. Lorenz, R. A. L. Rabelo, and J. Al-Muhtadi, "Performance comparison of programming languages forInternet of Things middleware," *Transactions on Emerging Telecommunications Technologies*, vol. 31, no. 12, Dec. 2020.

[54] A. K. Mayilyan and L. M. Hovsepyan, "Research and Comparative Analysis of Build Technology Programming in Java," *Mathematical Problems of Computer Science*, vol. 50, pp. 107–110, Dec. 2018.

[55] P. Siriwardena, *Maven Essentials*, 1st ed. New York City, NY, USA: Apress, 2015.

[56] J. A. M. Stothers and A. Nguyen, "Can Neo4j Replace PostgreSQL in Healthcare," *AMIA Joint Summits on Translational Science*, vol. 2020, no. 1, pp. 646–653, May 2020.

[57] R. Poljak, P. Poščić, and D. Jakšić, "Comparative Analysis of the Selected Relational Database Management Systems," in *Proceedings of the International Convention MIPRO*, Opatija, Croatia, 2017, pp. 1496–1500.

[58] Swagger, *OpenAPI Specification - Version 3.0.3 | Swagger*, 2021. [Online]. Available: `https://swagger.io/specification/`, Accessed: 2021-10-15.

[59] C. Gackenheimer, *Introduction to React*, 1st ed. New York City, NY, USA: Apress, 2015.

[60] A. Kumar and R. K. Singh, "Comparative analysis of AngularJS and ReactJS," *International Journal of Latest Trends in Engineering and Technology*, vol. 7, no. 4, pp. 225–227, Nov. 2016.

[61] Vitaliy Ilyukha, *Bootstrap: A detailed comparison*, 2019. [Online]. Available: `https://jelvix.com/blog/bootstrap-vs-material`, Accessed: 2021-10-15.

[62] John Jakob "Jake" Sarjeant, *Axios*, 2020. [Online]. Available: `https://axios-http.com`, Accessed: 2021-10-15.

[63] S. Tilkov and S. Vinoski, "Node.js: Using JavaScript to Build High-Performance Network Programs," *IEEE Internet Computing*, vol. 14, no. 6, pp. 80–83, Nov. 2010.

[64] C. Wolf and E. M. Halter, *Virtualization, From the Desktop to the Enterprise*, 1st ed. New York City, NY, USA: Apress, 2005.

[65] C. Anderson, Presenter, *Episode 217: James Turnbull on Docker*, Software Engineering Radio: the podcast for professional software developers, Jan. 2015. [Online]. Available: `https://podcasts.apple.com/us/podcast/episode-217-james-turnbull-on-docker/id120906714?i=1000330227102`.

[66] T. Combe, A. Martin, and R. D. Pietro, "To Docker or Not to Docker: A Security Perspective," *IEEE Cloud Computing*, vol. 3, no. 5, pp. 54–62, Nov. 2016.

[67] B. B. Rad, H. J. Bhatti, and M. Admadi, "An Introduction to Docker and Analysis of its Performance," *International Journal of Computer Science and Network Security*, vol. 17, no. 3, pp. 228–235, Mar. 2017.

# Deployment instructions

The system uses containers to achieve virtualization and adopted Docker for the container management. The following table presents the containers used in the system's deployment, detailing the Docker images and the ports used.

| **Container name** | **Internal port** | **External Port** | **Depends on** |
|---|---|---|---|
| *postgres* | 5432 | 5432 | |
| *backend* | 8000 | 8100 | postgres |
| *frontend* | 3000 | 3000 | backend |
| *nginx* | 3000 | 80 | nginx |

The right order to deploy the containers is *postgres*, *backend*, *frontend* and finally *nginx*.

A Makefile was created to ease the deployment process on the local machine. It allows to set up all the containers at once or to specify the container to build. Therefore, to deploy all system's containers or just one it is necessary to execute the commands:

- *make all* - deploys all the containers of the system.
- *make container_name* - deploys only the specified container and those that it depends on, replacing *container_name* by the name of the container, e.g, *make postgres* to deploy the PostgreSQL container.

The stoppage and removal of containers is also performed using the Makefile with the commands

- *make stop_all* – to stop and remove all containers.
- *make stop_container_name* – to remove the desired container and those that depend on it.