INSTITUTO
UNIVERSITÁRIO
DE LISBOA

# OSCAR: A Software-Based System to Support Information Incident Response

Ana Rita Paulo Raposo

Master's degree in Telecommunications and Computer Engineering,

Supervisor:
PhD Carlos José Corredoura Serrão, Associate Professor,
Iscte - Instituto Universitário de Lisboa

Co-Supervisor:
PhD João Pedro Oliveira da Silva, Assistant Professor,
Iscte - Instituto Universitário de Lisboa

October, 2024

Department of Information Science and Technology

# OSCAR: A Software-Based System to Support Information Incident Response

Ana Rita Paulo Raposo

Master's degree in Telecommunications and Computer Engineering

Supervisor:
PhD Carlos José Corredoura Serrão, Associate Professor,
Iscte - Instituto Universitário de Lisboa

Co-Supervisor:
PhD João Pedro Oliveira da Silva, Assistant Professor,
Iscte - Instituto Universitário de Lisboa

October, 2024

## Acknowledgements

I would like to express my heartfelt gratitude to my family for their unwavering support and encouragement throughout my research journey.

I am deeply thankful to my supervisors for their guidance and mentorship, which have been instrumental in shaping this dissertation.

I also extend my appreciation to FCCN for their collaboration and support, particularly to Carlos Friaças, whose assistance has been pivotal to the success of this dissertation.

## Resumo

A capacidade de detetar e realçar incongruências em valores de atributos equivalentes através de fontes de dados heterogéneas é crucial para uma triagem de incidentes eficaz nos processos de resposta a incidentes de informação. Nesta dissertação investigamos se é viável alcançar este objetivo através do desenvolvimento de um sistema software criado para detetar de discrepâncias entre atributos de dados equivalentes.

A pesquisa identifica e aborda requisitos necessários para uma implementação bem-sucedida, incluindo a capacidade de extração de dados programada, deteção eficiente de incongruências e uma interface de utilizador intuitiva que facilita a análise por parte do utilizador. O sistema foi testado no contexto operacional real pelos utilizadores finais, demonstrando a sua eficácia na identificação e realce de inconsistências entre valores de atributos equivalentes em várias fontes de dados.

Os resultados indicam que o sistema cumpre as suas funções pretendidas ao auxiliar a FCCN na identificação e correção de discrepâncias nos dados, melhorando a fiabilidade e a integridade dos mesmos dados utilizados no processo de resposta a incidentes.

**Palavras-chave:** integração de dados, fontes de dados heterogêneas, inconsistência em dados, resolução de conflitos, retificação de conflitos, reconciliação de conflitos

## Abstract

The ability to detect and highlight incongruences in equivalent attribute values across heterogeneous data sources is crucial for effective incident triage in information incident response processes. In this dissertation, we investigate whether it is feasible to achieve this goal through the development of a software-based system designed to detect discrepancies among equivalent data attributes.

The research identifies and addresses requirements necessary for successful deployment, including the capability for scheduled data extraction, efficient incongruence detection, and an intuitive user interface that facilitates user analysis. The system has been tested in the real operational environment by the end-users, demonstrating its effectiveness in identifying and highlighting inconsistencies between equivalent attribute values across multiple data sources.

The outcomes indicate that the system meets its intended functions by aiding FCCN in ensuring that discrepancies in data are identified and corrected, enhancing the reliability and integrity of the same data used in the incident response process.

**Keywords:** data integration, heterogeneous data sources, data inconsistencies, conflict resolution, conflict rectification, conflict reconciliation

## Contents

## List of Figures

## List of Tables

## Glossary of Abbreviations

| | |
|---|---|
| API | Application Programming Interface |
| CERT | Computer Emergency Response Team |
| CLS | Cumulative Layout Shift |
| CSIRT | Computer Security Incident Response Team |
| CSRF | Cross Site Request Forgeries |
| CSS | Cascading Style Sheets |
| DNS | Domain Name System |
| DSRP | Design Science Research Process |
| DTL | Django Template Language |
| FCCN | Fundação para a Computação Cientifica Nacional |
| FCT | Fundação para a Ciência e a Tecnologia |
| HTML | Hypertext Markup Language |
| HTTP | Hypertext Transfer Protocol |
| HTTPS | Hypertext Transfer Protocol Secure |
| INP | Interaction to Next Paint |
| IP | Internet Protocol |
| JSON | JavaScript Object Notation |
| LCP | Largest Contentful Paint |
| NFD | Normalisation Form Decomposition |
| ORM | Object-Relational Mapper |
| PRISMA | Preferred Reporting Items for Systematic Reviews and Meta-Analyses |
| RCTS | Rede Ciência, Tecnologia e Sociedade |
| RLI | Representation Level Inconsistency |
| SLR | Systematic Literature Review |
| SQL | Structured Query Language |
| SSL | Secure Sockets Layer |
| TLS | Transport Layer Security |
| UI | User Interface |
| URL | Uniform Resource Locator |
| XSS | Cross-Site Scripting |
| WSGI | Web Server Gateway Interface |

# Chapter 1 – Introduction

In the first chapter of this dissertation, the motivation and relevance of the topic are elucidated to facilitate an understanding of the key concepts that permeate the dissertation. Subsequently, the research question is defined, and the objectives are outlined, all of which we intend to achieve
through this research. The methodological approaches applied throughout this research are also detailed, as well as the proposed framing and structure of this dissertation, to serve as a roadmap for this research.

## 1.1 Research problem and motivation

The "Fundação para a Computação Científica Nacional" (FCCN), the Scientific Computing Unit of the "Fundação para a Ciência e a Tecnologia" (FCT), is a public institute under the responsibility of the Ministry of Science, Technology, and Higher Education, that serves the Portuguese academic and scientific community by providing technology for knowledge. FCCN's division, the Rede Ciência, Tecnologia e Sociedade (RCTS) Computer Emergency Response Team (CERT), specifically offers incident response and cybersecurity services to their constituencies—the research and education community. They ensure timely and effective handling of security incidents by providing necessary technology and support for the incident handling process.

Incident handling involves several high-level phases such as detection, triage, analysis, and response (Good Practice Guide for Incident Management, 2010, p. 34).

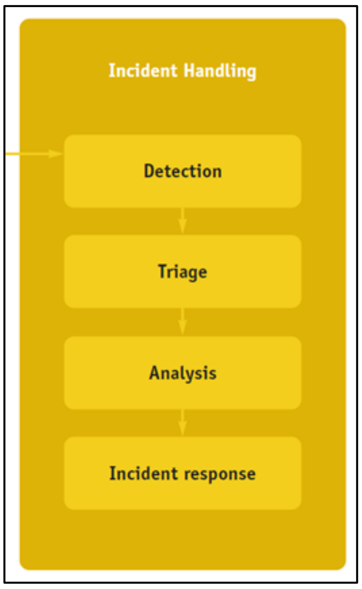


Figure 1.1 - Incident handling workflow[1]

Although incident handling workflows and their specific phases may have different definitions, most workflows include similar activities, such as receiving incident reports, registering reports, triage, and further processes (Good Practice Guide for Incident Management, 2010, p. 39).

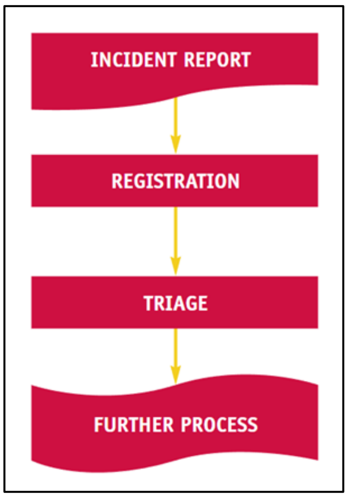


Figure 1.2 - Part of detailed Incident handling workflow – descriptive workflow[2]

[1] Good Practice Guide for Incident Management. (2010, December 20, p. 34). ENISA. https://www.enisa.europa.eu/publications/good-practice-guide-for-incident-management

[2] Good Practice Guide for Incident Management. (2010, December 20, p. 39). ENISA. https://www.enisa.europa.eu/publications/good-practice-guide-for-incident-management

An incident, defined here as "an event with a real adverse effect on the security of networks and information systems" (freely translated by the authors) (Decreto Lei no 46/2018, 13 de Agosto, 2018, article 3rd), undergoes several phases of scrutiny before reaching the triage phase, which consists of three sub-phases: verification, initial classification, and assignment (Good Practice Guide for Incident Management, 2010, pp. 45-46).

Incident triage, where Computer Security Incident Response Teams (CSIRTs) swiftly assess and prioritise incidents based on their criticality and potential impact, holds immense significance in cybersecurity. It serves as the initial filter in the incident handling process and influences the efficiency of subsequent incident response activities. Ensuring the prompt association of vital information, consistently across the various FCCN's proprietary resources and platforms (which we will refer to as data sources, from this point onwards) is crucial for an effective incident triage.

For RCTS CERT, information regarding constituents' attributes (e.g., Organisation name, IP addresses, etc.), used in the incident response process, exists across various data sources. Due to the disparity of these data sources and their different ownership, an organisation's attribute information is represented and stored differently. The workflow and data processing referred to above sets the problem identified for this dissertation, to which this research intends to deliver a tool to be made available as part of the procedures for the incident handlers on the operational first line (triage) of RCTS CERT, to cope with the potential inconsistency or inaccurate triage process.

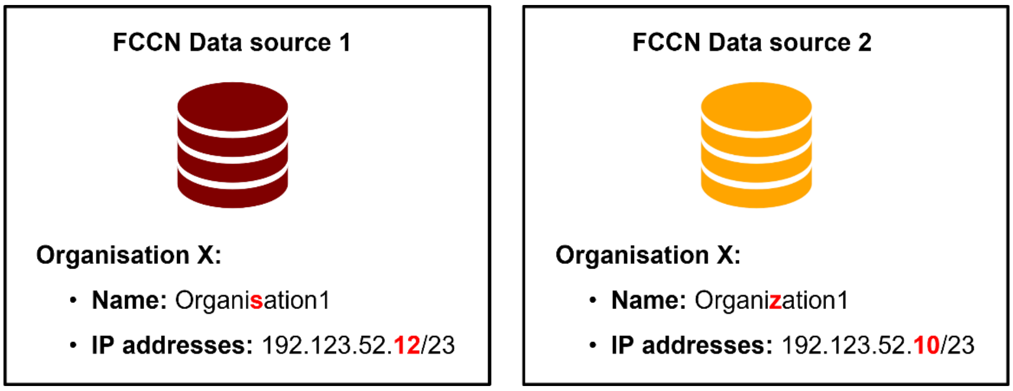


Figure 1.3 - Incongruences between values of the same organisation attributes stored in different data sources used by FCCN

The figure above illustrates the mismatches, discussed prior, of organisations values between data sources, which has the potential to cause a negative impact along the subsequent incident handling processes, due to the lack of semantic interoperability among these data sources, which leads to difficulties in finding and matching this crucial information necessary for effective incident triage.

## 1.2 Research question

This dissertation aims to answer the following research question:
“Is it possible to detect and highlight incongruences in equivalent attribute values stored across several heterogeneous data sources, for effective incident triage in the information incident response process?”

## 1.3 Research objectives

This dissertation sets forth the following objectives:

- Research the best practices and prevalent methods for data integration of heterogeneous data sources and data-level incongruence detection.
- Design and develop a web application capable of extracting, storing, and displaying data from various heterogeneous sources. In addition, scrutinise the data for data-level incongruences between the organisation's equivalent attributes, and highlight them.
- Implement and test the web application by employing its features to fulfil its intended purpose and requirements.
- Assess the efficiency of the web application by validating the execution of the requirements described in Chapter 3. This evaluation will provide insights into the web application’s effectiveness and areas for potential improvement, outlined in Chapter 5.

## 1.4 Research process

In this dissertation, a comprehensive research approach is adopted. It combines the use of a Design Science Research Process (DSRP) with a Systematic Literature Review (SLR), following the Preferred Reporting Items for Systematic Reviews and Meta-Analyses (PRISMA) methodology. This approach allows us to delve deep into the subject matter and formulate a solution based on the insights gathered.

Our research process begins with the DSRP, whose iterative nature allows for the continuous improvement and refinement of the proposed solution through repeated cycles to produce effective, applicable, and practical information technology artefacts to solve real-world problems (Peffers et al., 2020).

Throughout the DSRP, a Systematic Literature Review, following the PRISMA methodology, is conducted and presented in Chapter 2. The SLR is a rigorous method that involves a detailed analysis of existing scientific literature on a specific subject. It identifies relevant studies from a wide range of sources, providing a comprehensive overview of the subject (Kitchenham et al., 2009). The PRISMA methodology is designed to help systematic reviewers transparently report why and how the review was done, what the authors did, and what they found (Page et al., 2021).

By combining these methodologies, we intend to solidify a thorough understanding of the research problem. This understanding will form the backbone of this dissertation and guide the exploration and resolution of the problem at hand.

The DSRP entails the execution of six activities, as depicted in Figure 1.4, each of which is further discussed below.

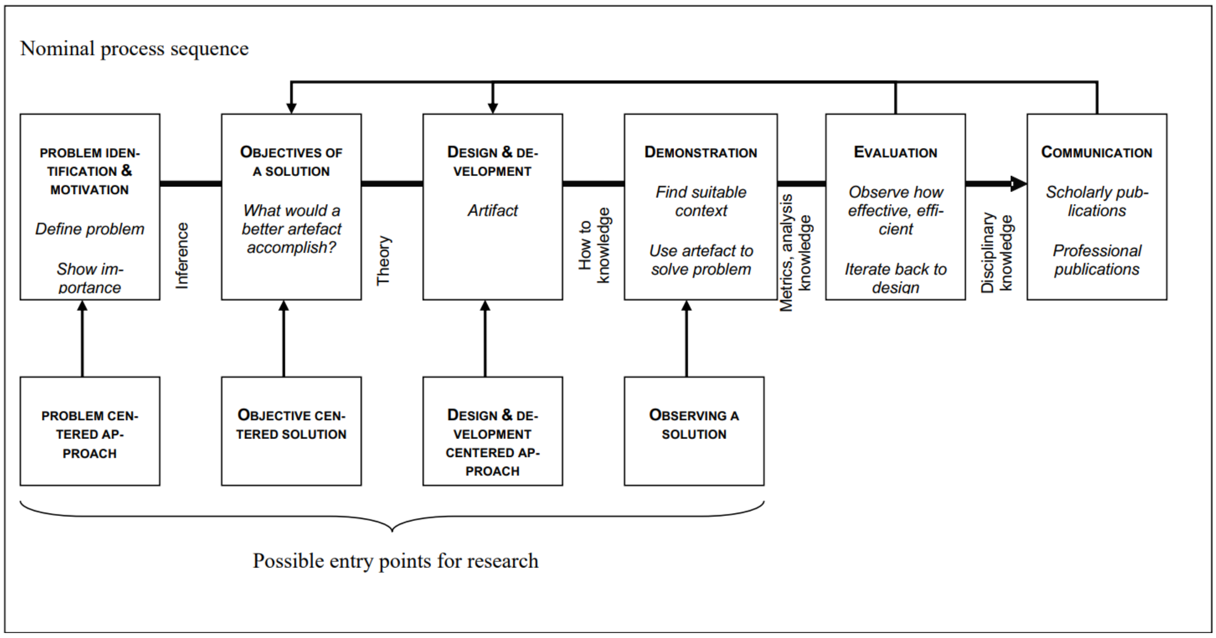

Nominal process sequence
PROBLEM IDENTIFICATION & MOTIVATION
Define problem
Show importance
Inference
OBJECTIVES OF A SOLUTION
What would a better artefact accomplish?
Theory
DESIGN & DEVELOPMENT
Artifact
How to knowledge
DEMONSTRATION
Find suitable context
Use artefact to solve problem
Metrics, analysis knowledge
EVALUATION
Observe how effective, efficient
Iterate back to design
Disciplinary knowledge
COMMUNICATION
Scholarly publications
Professional publications
PROBLEM CENTERED APPROACH
OBJECTIVE CENTERED SOLUTION
DESIGN & DEVELOPMENT CENTERED APPROACH
OBSERVING A SOLUTION
Possible entry points for research

Figure 1.4 - Design science research process (DSRP) model[3]

### 1.4.1 Problem identification and motivation

In the first activity of this process, we define the research problem and justify the value of the solution. We also break down the problem into smaller components, allowing us to better address the various complexities of the problem (Peffers et al., 2020).

[3] Peffers, K., Tuunanen, T., Gengler, C. E., Rossi, M., Hui, W., Virtanen, V., & Bragge, J. (2020, June 4, p. 11). Design Science Research Process: A Model for Producing and Presenting Information Systems Research. http://arxiv.org/abs/2006.02763

In cybersecurity, teams such as FCCN's RCTS CERT, handle incidents for their respective constituencies. This process encompasses a triage phase which requires swift assessment and prioritization of incidents based on their criticality and potential impact. This is crucial for an efficient incident response. However, for FCCN, the accuracy of this assessment is often hindered by the use of several heterogeneous data sources during this phase.

In this crucial process of cross-referencing key data, such as IP addresses and DNS domains between various data sources, the lack of semantic interoperability among them also causes a lack of precision in the association process, thereby impeding efficient incident response.

In this dissertation, we propose a solution that produces a web application that stores and standardises the data from various sources and detects disparate values. This approach ensures the accurate association of vital data across heterogeneous data sources. The value of this solution lies in its potential to enhance the efficiency of incident response, mainly in its triage phase, enabling prompt and accurate identification of the impacted constituents. Consequently, this research holds significant implications for improving incident response capabilities within the academic and scientific community served by FCCN.

### 1.4.2 Objectives of the proposed solution

In the second activity, we define the objective of the proposed solution, drawing support from the problem definition and context (Peffers et al., 2020).

The primary objective of this dissertation is to implement a web application capable of detecting discrepancies in corresponding data values across heterogeneous data sources. Thus, aiming to improve the success rate of the correct association of crucial information during incident triages or when needed after due correction of the discrepancies.

### 1.4.3 Design and development

The third activity relates to the artefact's creation and development, determining its desired functionality and architecture (Peffers et al., 2020).

To achieve the objectives, defined previously, set for the web application it should be capable of the following:

- Within a configurable period (pre-defined schedule), periodically trigger the extraction of data from heterogeneous data sources into a database. Standardise the data types and presentation of the data from all sources when storing the database.
- Display the values and run a comparison of the values, from the different sources it exists in, for the same type of information, and if found, highlight the discrepancies.

- Additionally, a search feature is necessary to speed up the search for a specific organisation.

We delve deeper into the technology stack used in the design and development of this tool in Chapter 3.

### 1.4.4 Demonstration

In the fourth activity, we demonstrate how the artefact can be used to solve instances of the research problem (Peffers et al., 2020).

Chapter 3 also presents a demonstration of this web application showcasing its application in a model scenario of the real operational environment. We illustrate how the tool detects and highlights discrepancies in information across data sources. This demonstration highlights the tool's effectiveness in improving successful information association during incident triages or when needed.

### 1.4.5 Evaluation

In the fifth activity, we evaluate the solution comparing the goals with the actual results obtained using the artefact (Peffers et al., 2020).

The effectiveness of the web application will be evaluated based on the completion of the goals defined in Chapter 2, and implementation of the design requirements, previously discussed and further developed in Chapter 3. This evaluation will validate that the tool fulfils its objectives and is effective in enhancing incident response capabilities.

### 1.4.6 Communication

The sixth activity comprises communicating the problem, the artefact, the novelty, the usefulness, and the effectiveness to audiences (Peffers et al., 2020).

In this dissertation, we communicate our findings and the value of our solution to both academic and professional audiences. We detail the problem of inconsistent data values across heterogeneous data sources in cybersecurity incident handling, describe our tool and its features, and discuss its benefits in improving incident response.

We also share the insights gained from demonstrating and evaluating the tool, thereby contributing to the broader discourse on enhancing cybersecurity measures within organisations like FCCN. This communication is encapsulated in the dissertation and will be further disseminated through presentations and publications.

## 1.5 Dissertation structure

This dissertation is organised into five comprehensive chapters, each serving a distinct purpose in the progression of this research. Thus, it creates an extensive and cohesive exploration of the research scope and ensures a logical flow of information.

Chapter 1 sets the stage for the research, providing an overview of the research topic context, identifying the problem, and outlining the motivation behind this research. It also presents the research question, objectives, and methodologies adopted, along with a brief overview of the structure of this dissertation.

Chapter 2 delves into the theoretical framework of the study, presenting an SLR and discussing the challenges that need the implementation of the solution proposed in this research. It analyses various existing relevant solutions and discusses how these are used in detecting data value conflicts across heterogeneous data sources.

Chapter 3 is dedicated to the design, development and demonstration of the proposed solution. This chapter details the architecture of the solution, including the necessary components required to meet the objectives of this research. It also presents the demonstration of our tool's functionalities in a model scenario of the real operational environment.

Chapter 4 focuses on validating the effectiveness of the developed solution. It presents test cases for each of the tool's requirements and the outcomes of the validation.

Finally, Chapter 5 concludes the dissertation by summarising the findings of this research, providing recommendations based on these results, and suggesting potential avenues for future work in this field.

# Chapter 2 – State of the art

This chapter introduces the existing knowledge in data integration, outlining the research field and setting the context for our research question. We begin by detailing the methodology employed for our SLR, ensuring transparency and replicability in our approach.

Subsequently, we establish a foundational understanding of crucial concepts surrounding data integration and inconsistencies before delving deeper into existing publications and related work, providing a comprehensive understanding of the state of the art in data integration. These subsequent sections explore topics such as models used for data integration, the tools employed for conflict identification and resolution, and the prevalent methods for data harmonisation.

Through this combined approach, we aim to assess how our research aligns with the scope of our investigation, draw informed conclusions about current practices, and identify potential areas for further development.

## 2.1 Systematic literature review

To gain a deeper understanding of the subject, we conducted a SLR, following the PRISMA methodology. This approach facilitated a systematic step-by-step exploration of the topic.

### 2.1.1 Outlining of the systematic literature review

We selected three online repositories Scopus, IEEE Xplore, and Web of Science, four our review. We chose them because of to their relevance to the information technology domain and the peer-reviewed content they offer.

Since each of these online repositories uses different search options, we adapted them for each repository, with the overall intent of searching for results by a term corresponding to the abstract, index term, and bibliographic citation data (such as document title, publication title, etc.). After conducting this initial search, in January 2024, we then filtered the results according to the following criteria:

- Language: English
- Publication type: Journal or Conference proceeding
- Publication year range: 2010-2024

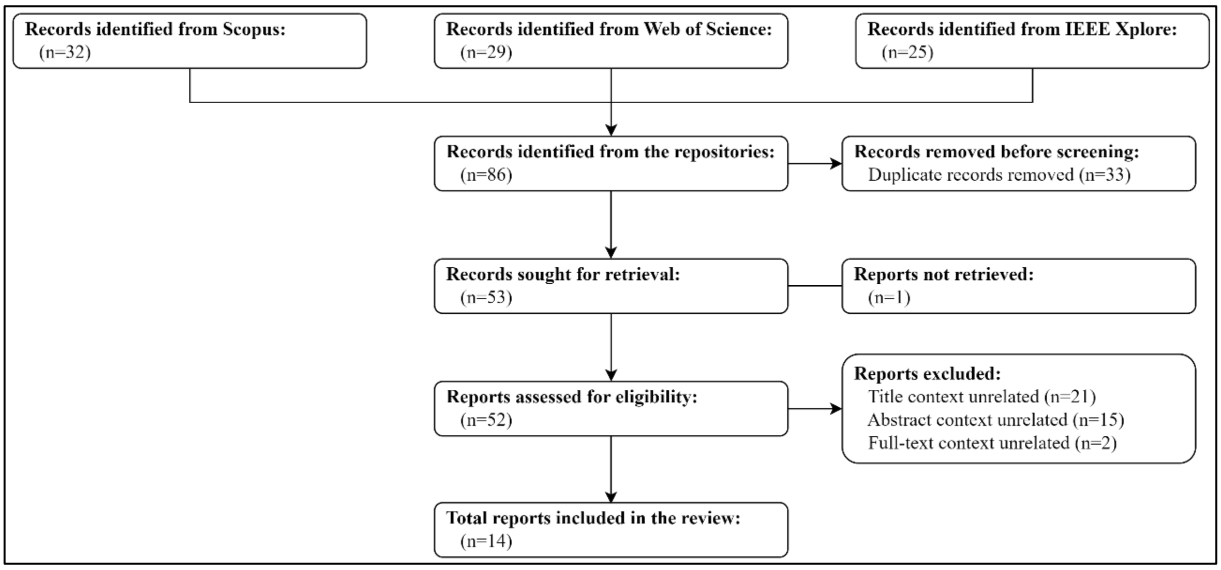
Table 2.1 presents the number of results returned from each repository, for each search term used.

Table 2.1 - Search terms used, and count of results returned per database

| Repository | Search term | Count |
|---|---|---|
| IEEE Xplore | ("All Metadata":data integration AND heterogeneous AND (inconsistenc* OR conflict*) AND (resolution OR rectif* OR reconcili*)) | 25 |
| Web of Science | TS=((data AND integration AND heterogeneous AND (inconsistenc* OR conflict*) AND (resolution OR rectif* OR reconcili*))) | 29 |
| Scopus | TITLE-ABS-KEY (data AND integration AND heterogeneous AND (inconsistenc* OR conflict*) AND (resolution OR rectif* OR reconcili*)) | 32 |

### 2.1.2 Conducting the systematic literature review

Figure 2.1 provides a detailed schematic representation of the review process, specifically illustrating the steps undertaken after concluding the search in the repositories, up to the final decision on which results to include in the review.



Records identified from Scopus: (n=32)
Records identified from Web of Science: (n=29)
Records identified from IEEE Xplore: (n=25)
Records identified from the repositories: (n=86)
Records removed before screening: Duplicate records removed (n=33)
Records sought for retrieval: (n=53)
Reports not retrieved: (n=1)
Reports assessed for eligibility: (n=52)
Reports excluded: Title context unrelated (n=21) Abstract context unrelated (n=15) Full-text context unrelated (n=2)
Total reports included in the review: (n=14)

Figure 2.1 - PRISMA 2020 flow diagram for the systematic review process

Our search across all repositories initially yielded a total of 86 records. After removing duplicates, we were left with 53 unique records. Out of these, 52 reports were accessible for free. We then screened these reports for context relevance based on their titles, which resulted in 31 potentially relevant reports. Further scrutiny of the abstracts of these 31 reports led us to a shortlist of 16 reports. After a thorough reading and analysis of these 16 reports, we found 2 to be insightful but not directly relevant to our research problem. Consequently, we excluded these reports, leaving us with 14 to include in our review. These are discriminated in Table 2.2.

Table 2.2 - Documents included in the review process

| **Author(s)** | **Title** | **Year** |
|---|---|---|
| Qutaany, A. Z. el, Bastawissy, A. H. el, & Hegazy, O. | A technique for mutual inconsistencies detection and resolution in virtual data integration environment | 2010 |
| Nachouki, G., & Quafafou, M. | MashUp web data sources and services based on semantic queries | 2011 |
| El-Demerdash, K. K., & Amer, F. | Data integration framework with an application for Ministry of Interior | 2012 |
| Boufares, F., & Salem, A. ben. | Heterogeneous data-integration and data quality: Overview of conflicts | 2012 |
| Dumitrescu, S.-R., Branescu-Raspop, I., & Popescu, D. | Format mediation for matching patient records in a three-layer information architecture | 2013 |
| Wei, D., & Xianxia, Z. | The Method to Resolve Schema Discrepancy of a Heterogeneous Database System | 2013 |
| Mirza, G. A. | Value name conflict while integrating data indatabase integration | 2014 |
| Saranya, K., Hema, M. S., & Chandramathi, S. | Data fusion in ontology based data integration | 2014 |
| Ghorbel, L., Zayani, C. A., & Amous, I. | A novel architecture for learner’s profiles interoperability | 2015 |
| Mirza, G. A. | Null Value Conflict: Formal Definition and Resolution | 2016 |
| Sonsilphong, S., Arch-int, N., Arch-int, S., & Pattarapongsin, C. | A semantic interoperability approach to health-care data: Resolving data-level conflicts | 2016 |
| Mirza, G. A., & Siddiqi, I. | Data level conflicts resolution for multi-sources heterogeneous databases | 2017 |
| Sharef, N. M., Shafazand, Y. M., Nazri, M. Z. A., & Husin, N. A. | Self-adaptive Based Model for Ambiguity Resolution of The Linked Data Query for Big Data Analytics | 2018 |
| Al-Baltah, I. A., Abd Ghani, A. A., Al-Gomaei, G. M., Abdulrazzak, F. H., & al Kharusi, A. A. | A scalable semantic data fusion framework for heterogeneous sensors data | 2020 |

## 2.2 Definitions

Data source: An entity or system that observes an object or phenomenon and provides information about it, in other words, collects and transmits data (e.g., websites, sensors, databases, or even human observers).

Ontology: Refers to the measure of reliability associated with the data information provided by a specific data source. It encompasses concepts related to the trustworthiness, accuracy, and consistency of the data. Similar to the weight of evidence, ontology helps assess the credibility of data from various sources.

Object: Represents an attribute or characteristic of an entity (e.g., the name of an organisation).

Data value: Pertains to the actual information associated with an object and includes specific measurements, descriptions, or characteristics (e.g., the organisation name "FCCN" is a data value). It is also known as "claim" in many articles, that is, the claim of an object given by a data source.

Data source schema: This defines the structure and properties of a data source, outlining how data within the source is organised and what attributes it possesses. However, it does not contain actual data.

## 2.3 Data integration

Data Integration is a critical process in the field of data management and analytics that involves the combination of data residing at homogeneous, autonomous, and heterogeneous sources into a unified global schema (Qutaany et al., 2010, p. 1), providing users with the ability to perceive the entire collection as a single source, query it transparently and receive a single and unambiguous answer (El-Demerdash et al., 2012, p. 1). The process can be broken down into two phases consisting of the integration of schema, and subsequently the integration of data, or data fusion. However, integrating different data sources into a single data analysis or integration operation can lead to problems when analysing a set of different data sources together, even if each source individually has no quality problems (Boufares et al., 2012).

A few studies have reported the use of a software system called DHResol to identify and resolve conflicts that arise during data integration (Mirza, G. A., 2014; Mirza et al., 2017), while others use a multi-layer architecture framework based on the principle that data access should be separated into multiple levels. An example of this framework is a three-layer architecture that isolates user applications from direct access to the database and consolidates data and corresponding rules in a single controlled place. This architecture is comprised of (Dumitrescu et al., 2013, p. 2):

- A User Interface Level, with the main function of translating tasks and results to something the user can understand. In this layer, the query language SPARQL is commonly used since it allows the users to write queries and extract information from non-uniform data, stored in various formats and sources.
- The Middleware Layer, where the mediation takes place, processes the data and creates a link between different data sources. In this layer, the mediator offers solid data access routines specific to each database incorporated, and an Ontology block is recommended for instances of complex processing (e.g. queries on different databases).
- A Database Level, where the data is stored and retrieved from a database or file system. The information is then passed back to the logic tier for processing and then eventually back to the user.

### 2.3.1 Integration of the schema

Schema integration, or schema merging, is the first step in a database integration system (Mirza et al., 2017, p. 1). This process involves schema matching and merging processes, which establish semantic mappings between participating data sources, to resolve heterogeneity at the schema level (El-Demerdash et al., 2012, p. 2).

This process is challenging due to differences in the structure and organisation of elements in data sources (Nachouki et al., 2011, p. 8), variations in query terms, and granularity conflicts (Sharef et al., 2018, p. 1).

#### 2.3.1.1 Schema-level inconsistencies

Schema-level inconsistencies, one of the three levels of inconsistencies in data integration, refer to differences in the used data models and the schemas within the same data model (Qutaany et al., 2010, p. 2), and pose a significant challenge in schema merging.

Mutual inconsistency is a problem in the schema merging process. Although the participating data sources are clean and consistent individually, inconsistencies arise when integrating them into one global access point (Qutaany et al., 2010).

Syntax inconsistency, a type of schema-level inconsistency, refers to different representation syntaxes among attributes with the same type (Boufares et al., 2012, p. 3), and is particularly prevalent when integrating different schemas from heterogeneous data sources (Mirza, G. A., 2014, p. 1).

Semantic heterogeneity, where two semantically different objects are referred to in the same way (El-Demerdash et al., 2012), and structural heterogeneity, referring to differences in the structure or format of schemas, are other forms of schema-level inconsistencies (Saranya et al., 2014).

#### 2.3.1.2 Representation-level inconsistencies

Representation-level inconsistencies (RLIs) are also one of three levels of inconsistencies in data integration (Qutaany et al., 2010, p. 2), and pose a significant concern. They occur when data is represented or encoded in different databases (Dumitrescu et al., 2013, p. 3).

They can appear as syntax inconsistencies, where different representation syntaxes exist among attributes with the same type in different data sources (Boufares et al., 2012, p. 3), and as data representational conflicts, such as having different lengths of attribute values (Mirza, G. A., 2016, p. 3).

Some techniques assume that RLIs have been well detected and resolved before handling mutual inconsistencies (Qutaany et al., 2010, pp. 6-7), an assumption that underscores the importance of addressing RLIs in the early stages of data integration and highlights the importance of considering the representation of the exchanged data as a dimension in data integration (Ghorbel et al., 2015, p. 2).

#### 2.3.1.3 Schema and representation-level inconsistencies resolution methods

The detection and resolution of schema and representation level inconsistencies is a prerequisite before addressing data level inconsistencies. The resolution of these inconsistencies often requires ontologies, to provide richer semantics and help overcome schematic and semantic conflicts (Saranya et al., 2014), used in different approaches: single ontology, multiple ontologies, and hybrid ontologies. In the single ontology approach, all data sources are related to one global ontology, while in the multiple ontology approach, each data source has its own ontology, and mapping between ontologies is necessary for integration (Nachouki et al., 2011). The use of an H-TMATCH algorithm also helps to identify corresponding concepts between two schemas (Nachouki et al., 2011, p. 8).

### 2.3.2 Integration of the data

The integration of the data phase involves addressing data level inconsistencies and merging data (Dumitrescu et al., 2013, p. 2), also called, data object fusion, which is performed in conjunction with semantically equivalent data objects coming from different sources.

#### 2.3.2.1 Data-level inconsistencies

One of the major challenges in data integration is the identification and resolution of data-level conflicts (Mirza, G. A., 2014) since not all data-level conflicts are resolved during the integration of the schema process. These refer to factual discrepancies among sources that describe the same data objects, and can be sorted into four types of conflicts (Sonsilphong et al., 2016, p. 5):

- Data type conflict: when the data values of the semantically equivalent data type properties are defined with different schema data types.
- Data format conflict: when the data values of the semantically equivalent data type properties are represented with different formats (or patterns).
- Data value conflict: when the data values of the semantically equivalent data type properties are synonymous or have different values but the same meaning.
- Data scaling conflict: when the data values of semantically equivalent data type properties have different scaling or units of measurement.

#### 2.3.2.2 Data-level inconsistencies resolution methods

Data conflict resolution is an important aspect of data integration and has received increasing attention in the field (Saranya et al., 2014). To handle these inconsistencies, techniques have been proposed that aim to handle mutual inconsistencies in the virtual data integration environment, assuming that schema-level inconsistencies and representation-level inconsistencies are well detected and resolved (Qutaany et al., 2010).

Z-Notation, the formal specification language, is often referenced for providing the power to formally define and represent the resolution for these conflicts (Mirza, G. A., 2014, p. 4). Z language functions make it possible to express the database schema as well as the scenarios where these conflicts arise.

Another popular approach was using mapping rules and heuristics to remove data conflicts and determine consistent representations (Saranya et al., 2014, p. 2), and using a Markov logic network to improve data quality by handling data conflicts in ontology-based data integration (Saranya et al., 2014, p. 6).

The “Cry with the wolves” method was also used to resolve conflicts by deciding which constraints to consider and which to ignore (Boufares et al., 2012, p. 6).

## 2.4 Conclusion

The provided research review paints a comprehensive picture of the state of the art in data integration, highlighting the advancements and challenges associated with data integration from diverse data sources. This complex process, which involves several steps and techniques to ensure a unified and meaningful view of information from multiple sources (Mirza et al., 2017, p. 1), almost always requires the use of semantic techniques, ontologies, specific architectures, and methods to address the challenges and conflicts that arise during the process.

Addressing schema and representation-level inconsistencies is also a crucial step in the data integration process. Resolving them is a prerequisite for correctly detecting and resolving various data-level inconsistencies. However, the literature often lacks variety in specific methods or procedures for handling these inconsistencies, indicating a need for further research in this area.

Future directions in this field include implementing conflict-avoidance methods and proactive measures, such as data schema alignment and quality improvement, to minimise conflicts during the integration process.

In conclusion, data integration remains an active research area with constant advancements. Addressing the identified challenges will facilitate seamless and efficient data integration, unlocking value within diverse data sources.

# Chapter 3 – Proposed solution design, development and demonstration

In this chapter, we present the design and development of the proposed solution—a web application called OSCAR (**O**rganisation **S**anitiser **C**orrectional **A**nd **R**ectifier) designed to detect and highlight conflicts in data values across heterogeneous data sources. We cover the solution's architecture, core features, and the technologies involved in its development. Additionally, we explain how information flows through the application's components, and demonstrate the web application functionalities.

To preserve privacy, the names of FCCN's platforms and data sources/repositories used to develop the web application were replaced by generic ones.

## 3.1 Requirements

To achieve the objectives set in this dissertation, the requirements for the web application were identified through frequent meetings with the application's end-user in FCCN, over a period of one year, which allowed for continuous refinement and adaptation to ensure the solution would meet the business needs. Throughout these meetings FCCN provided us resources for the solution's implementation and provided us feedback on the tool's design and development. The outcome was both the functional and non-functional requirements that should be fulfilled for the application to operate effectively.

### 3.1.1 Functional requirements

The functional requirements defined were the core features that the web application must provide to meet the end user's needs and align with this dissertation's objectives:

- User authentication: The application must require users to authenticate before accessing the data, ensuring security and control over sensitive information.
- Scheduled data extraction and storage: The application must automatically extract data from sources, standardise it, and store it in a database at configurable intervals, ensuring the use of the most up-to-date information.
- Incongruent information detection functionality: The application must include functionality which compares data from different sources and highlights any inconsistencies in equivalent attributes for further analysis.
- Search functionality: The application must include a search feature that enables users to quickly locate a specific organisation.

### 3.1.2 Non-functional requirements

The non-functional requirements defined address the quality attributes and performance characteristics that the application must include:

- Usability: The application should be user-friendly, with an intuitive interface that simplifies searching for and identifying data discrepancies.
- Availability: The application must always be available, ensuring that users can access it and use its features at any time.
- Performance: The application must process datasets efficiently and quickly return search results and discrepancies.
- Security: Strong security measures must be in place to protect data integrity and ensure secure data storage, including securing data in transit and at rest.
- Cross-browser compatibility: The application should have the same performance across multiple browsers.
- Technology stack: All application development and validation technologies must be free and open source.

## 3.2 Application architecture design

The application's architecture is built on top of Django, a full-stack web framework that handles both back-end logic and front-end rendering. The application uses MongoDB as the database to manage the heterogeneous data sources and their varied schemas. For efficient traffic management, Nginx serves as a reverse proxy, handling tasks such as security, load balancing, and content delivery. Gunicorn is used as the WSGI (Web Server Gateway Interface) HTTP (Hypertext Transfer Protocol) server, enabling Django to communicate with Nginx and manage multiple requests concurrently. This architecture ensures a scalable, flexible solution that handles complex data and traffic scenarios. A detailed diagram of this architecture, illustrating how data flows through each component, is provided in 3.1.

### 3.2.1 Application back-end

The back end of the application is primarily powered by Django and Nginx, with Gunicorn facilitating the communication between them. Django handles the application's core logic and routing requests, while Nginx acts as a reverse proxy server, optimizing web traffic and ensuring secure content delivery. Additionally, Systemd is employed for process management, ensuring that the application services start automatically and remain operational.

#### 3.2.1.1 Django

Django (the version used was 4.2.7) was chosen for this application due to its comprehensive documentation, built-in security features, and strong community support. Its ORM (Object-Relational Mapper) helps prevent SQL (Structured Query Language) injection attacks by using prepared statements to sanitize user input (Aborujilah et al.). Additionally, Django's template language automatically escapes potentially harmful characters, effectively protecting against XSS (Cross-Site Scripting) attacks (Aborujilah et al.). The framework also offers default CSRF (Cross-Site Request Forgery) protection by requiring a secret token for form submissions, ensuring that requests are valid (Aborujilah et al.). These built-in security mechanisms make Django an ideal choice to develop a secure application.

In Django, views handle HTTP requests and return HTTP responses. These views process the data from MongoDB and use the templates to render the final HTML (Hypertext Markup Language) sent to the user's browser. The views in this project handle tasks such as user authentication, logout, fetching data from MongoDB, and rendering the appropriate templates based on the requested URL (Uniform Resource Locator). Django's URL patterns define how the URLs are mapped, routing incoming requests to the appropriate view function.

This technology helps to fulfil the following requirements:

- User authentication (functional): Django's built-in authentication system handles secure user access to the application.
- Scheduled data extraction and storage (functional): Django handles data retrieval from MongoDB, preparing it for display.
- Search functionality (functional): Django handles search queries, processing user inputs and rendering the relevant results.
- Performance (non-functional): Django processes the datasets retrieved, ensuring their quick delivery to the user.

#### 3.2.1.2 Gunicorn

Gunicorn is a Python WSGI HTTP server that acts as a bridge between Nginx and Django, facilitating the seamless passage of HTTP requests between the two components. In this application, the WSGI configuration is managed in the "wsgi.py" file, which serves as the entry point for the Django application during deployment. Gunicorn is configured to use this WSGI file, allowing it to interface with the Django application. The configuration typically includes settings for the number of worker processes, binding addresses, and any necessary environment variables, ensuring optimal performance and resource management during runtime.

This technology helps to fulfil the following requirements:

- Performance (non-functional): Gunicorn ensures seamless communication between Nginx and Django, contributing to fast content delivery and response times.
- Availability (non-functional): By managing multiple worker processes, Gunicorn enhances the application's availability, allowing it to handle more simultaneous requests and maintain responsiveness during peak usage.

#### 3.2.1.3 Scheduler

The application employs a scheduler (using "django-apscheduler") that automatically triggers the data retrieval and processing tasks at predefined intervals to extract and integrate the data from multiple sources. Each data source, whether it's a daily updated file or an internal/external repository, is processed through these scheduled tasks and is responsible for extracting, cleaning, transforming, and inserting the most up-to-date information into MongoDB collections.

Even though MongoDB is schema-less, each function uses a fixed document structure to retrieve values from the data sources. This standardisation facilitates the further highlighting of any missing or inconsistent data.

This technology helps to fulfil the following requirements:

- Scheduled data extraction and storage (functional): The scheduler automates the data extraction and storage process, ensuring the most up-to-date information is used.

#### 3.2.1.4 Nginx

Nginx (the version used was 1.26) serves as a reverse proxy server in this application's architecture. We chose Nginx due to its superior performance in handling concurrent connections and its efficient resource management as a reverse proxy. According to Kithulwatta et al. (2023), Nginx outperforms Apache in various scenarios, particularly in high-traffic environments, making it ideal for performance optimising. Additionally, Nginx provides robust security features, such as request filtering and SSL (Secure Sockets Layer) and TLS (Transport Layer Security) termination. This helps protect backend servers from direct exposure to the internet (Kithulwatta et al., 2023).

In this application's architecture, Nginx is responsible for handling incoming HTTP requests on port 443 and forwarding them to Gunicorn, which then communicates with Django via WSGI. Additionally, it manages the delivery of static files, allowing the application to process dynamic content more efficiently. By offloading static file handling to Nginx, the application's overall performance is enhanced.

This technology helps to fulfil the following requirements:

- Performance (non-functional): By managing static file delivery and optimizing request routing, Nginx contributes to fast content delivery and response times.
- Security (non-functional): The SSL/TLS secure protocol, managed by Nginx, ensures secure data transfer between the client and server, safeguarding data in transit.
- Availability (non-functional): Nginx ensures the application remains accessible by efficiently routing web traffic and preventing bottlenecks.

#### 3.2.1.5 Systemd

Systemd is a Linux service manager responsible for starting and managing system services. In this architecture, Systemd is configured to automatically run Gunicorn as a service, ensuring that the application starts on boot and remains running continuously. If the Gunicorn process crashes or stops, Systemd ensures that it is restarted automatically, thereby maintaining the web applications' uptime and reliability.

This technology helps to fulfil the following requirements:

- Availability (non-functional): By automatically restarting Gunicorn in the event of a failure, Systemd enhances the application's availability, ensuring that users can access it at any time.
- Performance (non-functional): With Systemd managing Gunicorn, the application benefits from optimized resource management and quicker response times. Services can be started in parallel during boot, reducing downtime.

### 3.2.2 Application data layer

Each data source in the system has a unique structure and set of fields, often differing in format and naming conventions. The goal of the data layer is to standardise them into a unified structure, so that they can be effectively compared and discrepancies can be detected. To achieve this, the application data layer leverages MongoDB's document-oriented database, which provides the necessary flexibility to handle diverse data formats and structures efficiently.

#### 3.2.2.1 MongoDB

MongoDB is a database for storing and managing data extracted from heterogeneous sources. Although Django offers a powerful built-in solution for handling relational databases through its ORM system, MongoDB was chosen for this application because of its superior handling of unstructured and semi-structured data, and its ability to scale horizontally as new data sources are introduced.

In this project, each MongoDB collection represents a distinct data source, and each document within a collection corresponds to an organisation.

This technology helps to fulfil the following requirements:

- Scheduled data extraction and storage (functional): MongoDB stores the data from heterogeneous sources, allowing it to be retrieved and processed.

#### 3.2.2.2 SQLite3 DB

SQLite3 manages the application's internal data and Django-related data used in the authentication process, such as user credentials and session information. This ensures that only authenticated users can access sensitive data and functionalities within the application.

This technology helps to fulfil the following requirements:

- User authentication (functional): By securely storing user credentials and session data, SQLite3 ensures that the application requires users to authenticate before accessing sensitive information, thus maintaining security and control over the data.

### 3.2.3 Application front-end

The application's front end is built using Django's templating engine, which is tightly integrated with the back end to render dynamic content directly from the server. Instead of relying on a separate JavaScript framework, Django's full-stack nature handles both presentation and logic. JavaScript is used selectively for more complex interactions and Cascading Style Sheets (CSS) are used to style the application, for consistency across the interface.

#### 3.2.3.1 Django template

A Django template is a text file that defines the structure or layout of an HTML page, with placeholders for dynamic content. This allows for seamless back-end data integration into front-end presentation by rendering HTML with data passed from Django views.

Django templates use a combination of HTML and DTL (Django Template Language) to dynamically render content, for example, to iterate through a list of organisations, tags like *{% for org_id in org_list %}* and *{% for d in data_source_1_data %}* is used. Conditional logic, such as *{% if d.org_id == org_id %}*, ensures that all organisations exist in each collection before their data is rendered.

Django's templating system supports the modular design, which allows common elements (e.g., headers, footers, navigation bars) to be reused across multiple pages, by using the *{% extends 'example.html' %}* tag, which enables child templates to inherit attributes from a parent template. For example, in this application base.html defines the global structure of the website, and specific sections like tables with organisational information can be customised using *{% block results %}* and *{% endblock %}* in child templates. These examples are illustrated in Appendix A.

This technology helps to fulfil the following requirements:

- Usability (non-functional): Django templates provide a user-friendly and intuitive interface for users to interact with the application.
- Search functionality (functional): Templates render the search results dynamically and present them to users.
- Incongruent information detection functionality (functional): Templates present highlighted inconsistencies detected between data sources, making discrepancies easily identifiable.

#### 3.2.3.2 JavaScript

JavaScript is integrated into Django templates to handle tasks that require dynamic updates without reloading the entire page. While Django renders the initial HTML, JavaScript handles interactions, such as comparing values and highlighting incongruences, toggling dark/light mode, and passing variables from the back end to the client side.

Django allows passing back-end context variables directly into JavaScript. For example, in this application, the *org_list* variable is passed as safe JSON (JavaScript Object Notation) from a Django template to JavaScript using template tags like *{{ org_list|safe }}*.

JavaScript handles tasks such as comparing data from multiple sources or toggling dark/light mode are loaded using Django's *{% load static %}* template tag. This is showcased in Appendix A.

This technology helps to fulfil the following requirements:

- Usability (non-functional): JavaScript improves user experience by allowing for interactivity, such as toggling dark/light mode.
- Incongruent information detection functionality (functional): JavaScript performs the logic to compare data from multiple sources and highlight discrepancies.

#### 3.2.3.3 CSS

In the application CSS is used to style the front end of the application, to define a visually consistent and responsive design across all pages. It defines common styles such as fonts, colours, appearance of data displays, and other UI (User Interface) elements, to ensure a clear and user-friendly interface. CSS files are managed as static assets and are loaded onto every page using the *{% load static %}* tag in the HTML <*head*> section.

This technology helps to fulfil the following requirements:

- Cross-browser compatibility (non-functional): CSS ensures that the application works seamlessly across multiple browsers, providing consistent styling and layout.
- Usability (non-functional): CSS improves the overall user interface, ensuring that the application is visually clear, responsive, and easy to navigate.

## 3.3 Information flow

In this section, we detail the flow of information through the web application architecture, from data retrieval to its display in the user's browser. The following steps explain how data is extracted, processed, and served, and accompanying 3.1 illustrates the process.

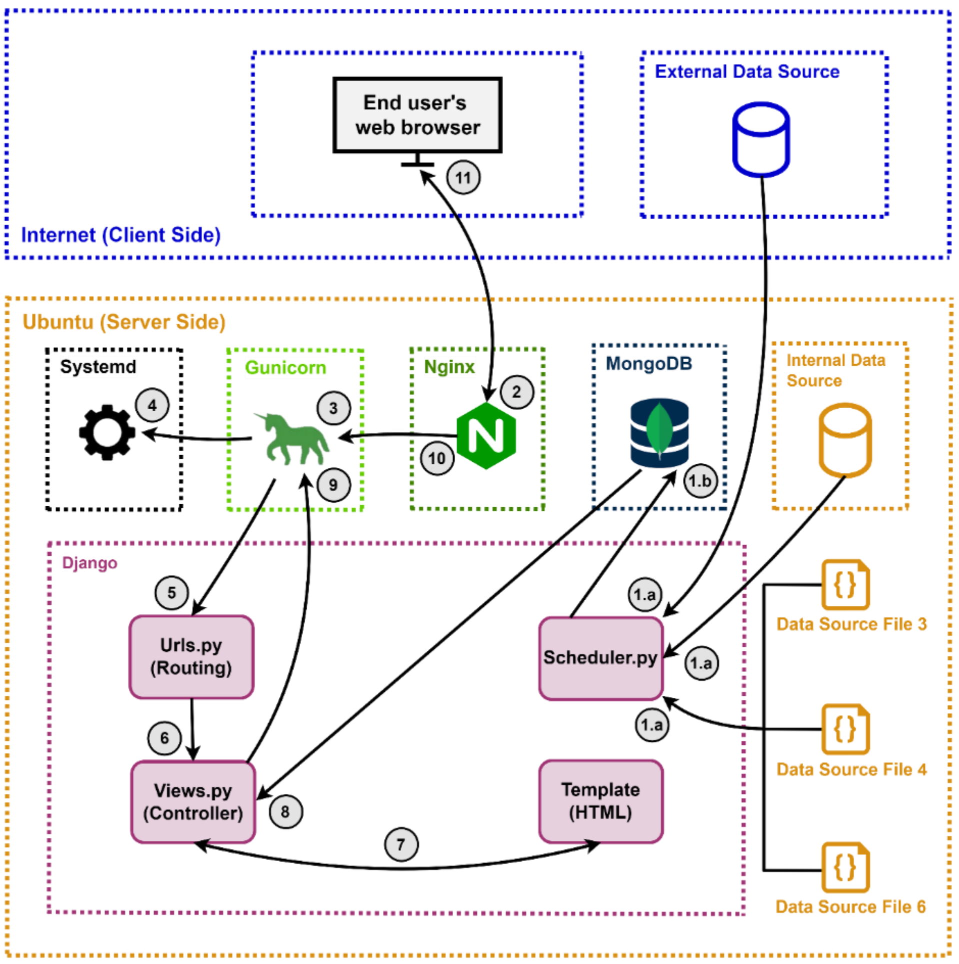


Figure 3.1 - Flowchart of the information’s flow through the web application architecture

1 - **Data retrieval and storage via scheduler:**

The scheduler is configured with several functions, each dedicated to retrieving data from specific data sources daily and storing it in MongoDB on a Virtual Machine (VM).

**a.** Each function in the scheduler is triggered sequentially. The functions retrieve data from the following types of sources.

**i.** A file updated daily by FCCN via a Jenkins job, containing data from the data source platform.

**ii.** Direct retrieval from a repository or platform, either located within the FCCN network or hosted externally on the Internet.

**b.** Each data source has its own dedicated MongoDB collection named after it. Data from each source is stored by overwriting the old data to ensure the most recent version is always displayed to the end user.

2 - **User request initiation:**

When a user opens the web application in a browser, an HTTP request is made to the web server Nginx through port 443, which receives the request and initiates the Request Phase.

3 - **Nginx forwards request to Gunicorn:**

After the request is received and passes through the predefined rules for security and routing in Nginx, it's forwarded via port 8000 to Gunicorn, the WSGI server that listens for incoming requests from Nginx.

4 - **Systemd manages Gunicorn service:**

Systemd is configured to automatically run Gunicorn as a service, ensuring that the application starts on boot and remains running.

5 - **Routing via Gunicorn:**

Gunicorn translates the incoming HTTP request into a Python object and passes it to the Django application, initiating the Process Request Phase within Django.

6 - **URL mapping and middleware processing:**

When an HTTP request is received, the Uniform Resource Locator (URL) is mapped to a specific Django View function based on the URL configuration in the urls.py file. The request is routed to the appropriate view, which contains the logic to handle the specific request.

7 - **Interaction with MongoDB:**

The Django View processes the retrieved data, applying any necessary logic, transformations, or data modelling as defined in the function. Once the data is processed, the view compiles the HTTP response by passing it to a Django Template for rendering.

8 - **Data processing and response preparation:**

The Django View then processes the data by conducting the remainder of the logic defined in the function, transforming and modelling the data for the HTTP response, and compiling the HTTP response created, back to the Django Template.

9 - **HTTP response preparation by Gunicorn:**

The completed HTTP response, which includes the rendered template and any linked static files (CSS, JavaScript), is then returned from Django to Gunicorn.

10 - **Gunicorn sends response back to Nginx:**

Gunicorn packages the response and passes the completed HTTP response back to Nginx via port 8000.

11 - **Final response delivery to user:**

Finally, Nginx securely sends the HTTP response, through port 443 back to the user's browser where the requested data is presented in the user interface. Nginx also takes care of serving the static files, which were pre-collected from Django's static folder and stored in the directory called *staticfiles*, that Nginx is configured to serve from.

## 3.4 Application functionalities

The following sections provide a technical insight into the application's functionalities and implementation.

### 3.4.1 Login page

The login page is the entry point for users, requiring authentication before accessing the application functionalities. This feature leverages Django's built-in authentication system to ensure that only authorised users can access the sensitive data. Form validation includes checks for username and password, error handling for incorrect logins, and redirects to the main interface upon successful authentication. Django's @login_required decorator is applied to all views to assure access to authenticated users only.

Additionally, password validation is enforced using a custom validator defined in the validator.py file to enhance security, adding an extra layer of protection to the authentication process. Users must comply with the password policy to create or change their passwords. The password requirements are the following:

- At least one uppercase letter
- At least one lowercase letter
- At least one digit
- At least one special character

A logout button is also provided in the main page, which redirects users back to the login page upon logging out.

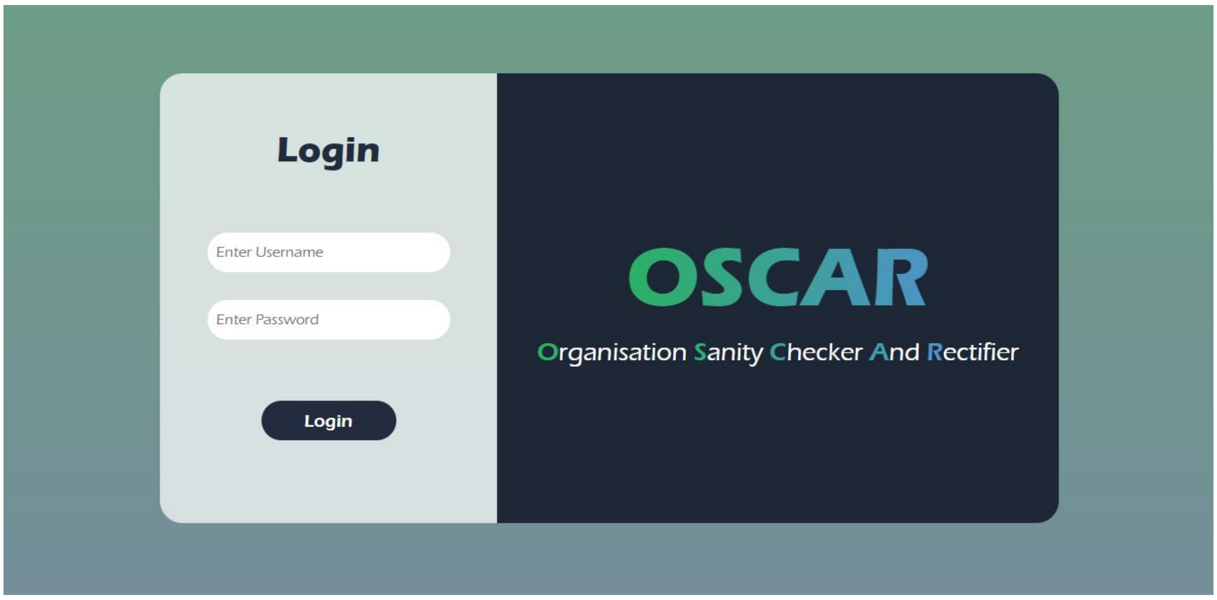


Figure 3.2 - Web application login page

### 3.4.2 Incongruent information detection functionality

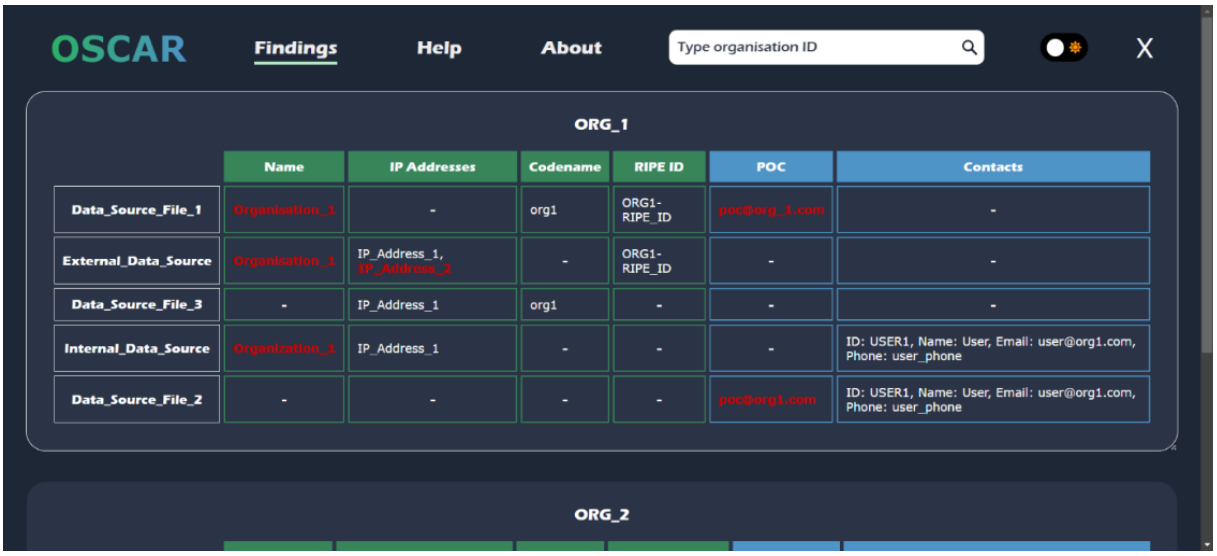
In the findings section, which serves as the default main page, each organisation's data is displayed in tables, with any inconsistencies in attributes (incongruences or missing values) highlighted across the data source rows. This layout allows users to visually assess where data may be incomplete or conflicting between sources. The JavaScript code is responsible for comparing and highlighting these discrepancies, while Django templates handle the dynamic rendering of data for each organisation.

The application employs specific logic to highlight discrepancies in key attributes such as names, IP addresses, descriptions, and others. A single dash ("-") indicates that an attribute does not exist in a particular source, while three red dashes ("---") signify that the attribute should exist but lacks a value in that source. For all attributes except IP addresses, the highlighting logic is the same, as shown in Appendix B. For example, if the name attribute is missing in a source where it should exist, and all other sources have the same name value for the organisation, the name fields are still highlighted to indicate inconsistency.

To ensure consistency in comparison, all name values are standardised through the *normaliseText()* function, also shown in Appendix B. This function removes diacritical marks and special characters from text, ensuring that minor formatting differences don't prevent a positive match between names such as "Organização" and "Organizacao".

For IP addresses, the system identifies addresses that are not universally present across all sources. Any blocks missing from a particular source are highlighted, giving a clear visual cue for missing or partially present IP ranges. The highlighting logic for IP addresses is also described in Appendix B.

Using standardised text comparison and a specific highlighting logic for each attribute ensures that users can easily interpret and assess data inconsistencies across different sources.



ORG_1

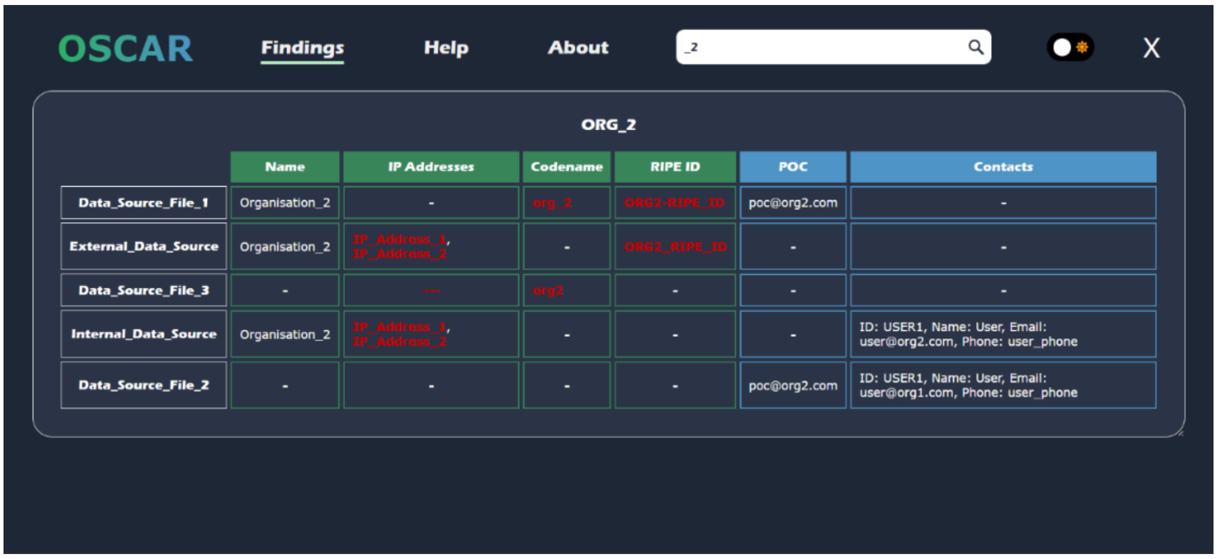
| | Name | IP Addresses | Codename | RIPE ID | POC | Contacts |
|---|---|---|---|---|---|---|
| Data_Source_File_1 | Organisation_1 | - | org1 | ORG1-RIPE_ID | poc@org_1.com | - |
| External_Data_Source | Organisation_1 | IP_Address_1, IP_Address_2 | - | ORG1-RIPE_ID | - | - |
| Data_Source_File_3 | - | IP_Address_1 | org1 | - | - | - |
| Internal_Data_Source | Organization_1 | IP_Address_1 | - | - | - | ID: USER1, Name: User, Email: user@org1.com, Phone: user_phone |
| Data_Source_File_2 | - | - | - | - | poc@org1.com | ID: USER1, Name: User, Email: user@org1.com, Phone: user_phone |

ORG_2

Figure 3.3 - Web application findings section

### 3.4.3 Search functionality

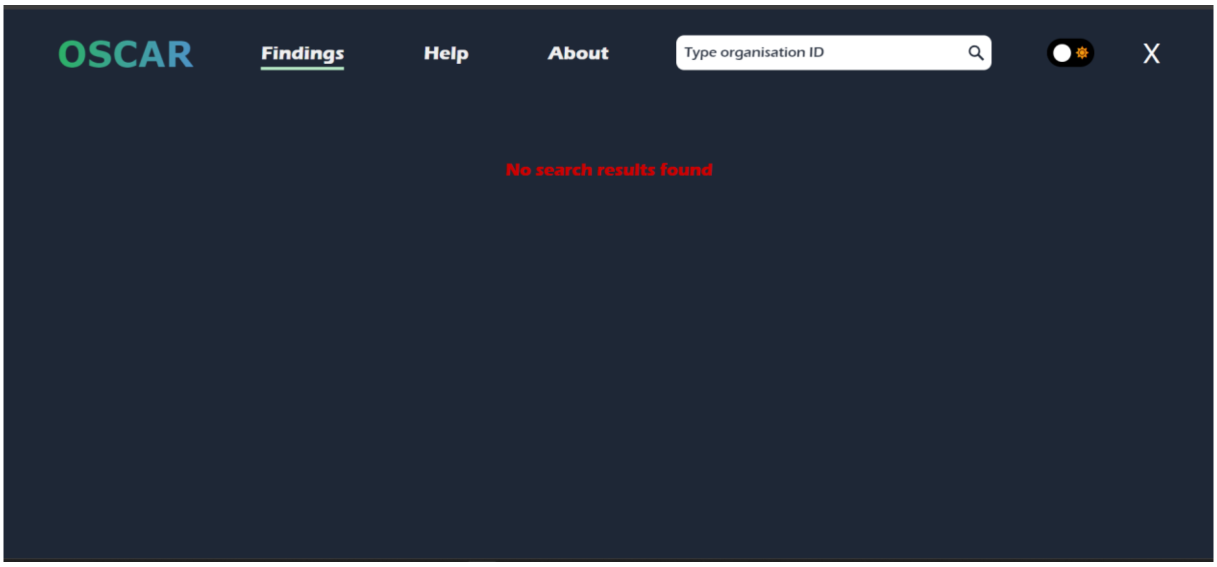
The search functionality allows users to quickly locate information about specific organisations by typing organisation IDs, partially or completely. The search input triggers a query across all the data stored in MongoDB, retrieving matching records. This feature is implemented using Django views to handle the back-end logic and render search results dynamically on the front end. In the end, the data comparison via JavaScript is triggered.



ORG_2

| | Name | IP Addresses | Codename | RIPE ID | POC | Contacts |
|---|---|---|---|---|---|---|
| Data_Source_File_1 | Organisation_2 | - | org_2 | ORG2-RIPE_ID | poc@org2.com | - |
| External_Data_Source | Organisation_2 | IP_Address_1, IP_Address_2 | - | ORG2_RIPE_ID | - | - |
| Data_Source_File_3 | - | --- | org2 | - | - | - |
| Internal_Data_Source | Organisation_2 | IP_Address_1, IP_Address_2 | - | - | - | ID: USER1, Name: User, Email: user@org2.com, Phone: user_phone |
| Data_Source_File_2 | - | - | - | - | poc@org2.com | ID: USER1, Name: User, Email: user@org1.com, Phone: user_phone |

Figure 3.4 - Web application search functionality results outcome

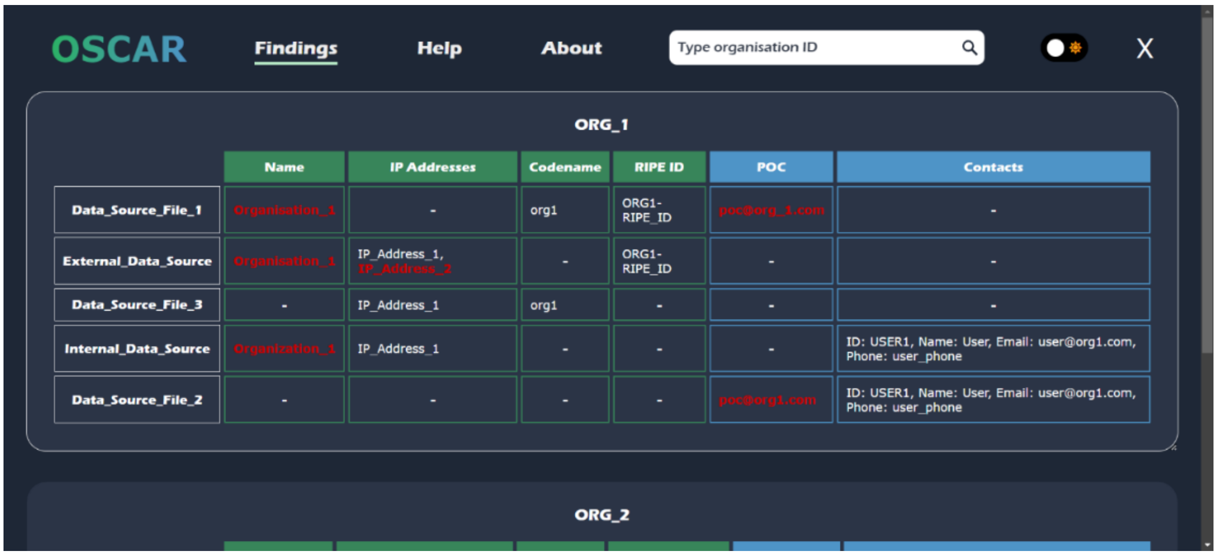


No search results found

Figure 3.5 - Web application search functionality no results outcome

### 3.4.4 Light and dark mode

The light and dark mode feature enhances the user experience by allowing users to toggle between light and dark themes and is implemented using JavaScript and CSS. JavaScript handles the mode switch and stores the user's preference in the browser's local storage, while CSS dynamically adjusts the theme.

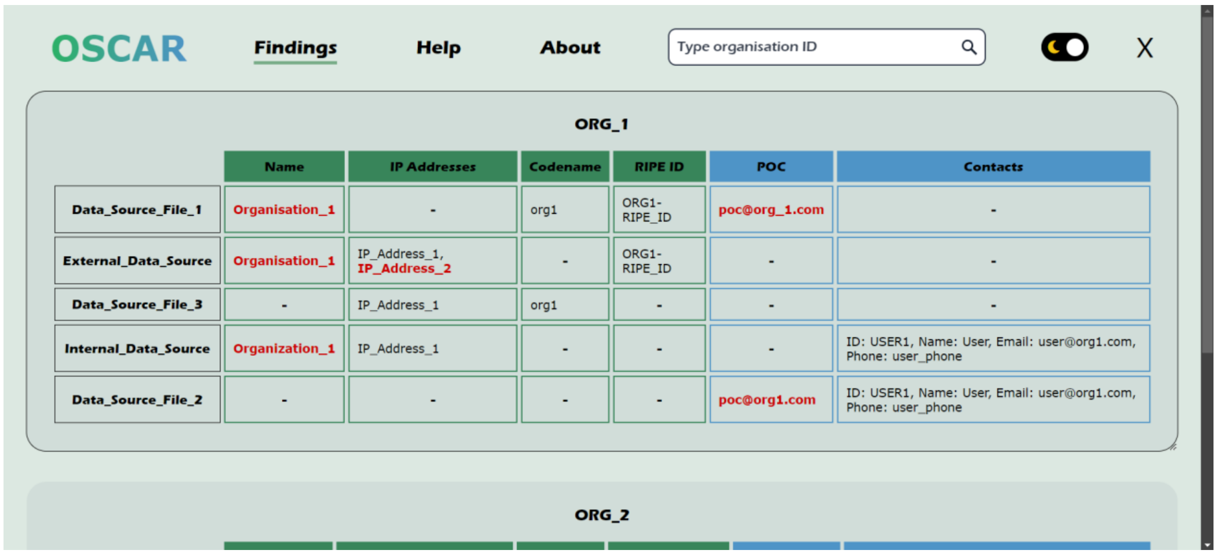


ORG_1

| | Name | IP Addresses | Codename | RIPE ID | POC | Contacts |
|---|---|---|---|---|---|---|
| Data_Source_File_1 | Organisation_1 | - | org1 | ORG1-RIPE_ID | poc@org_1.com | - |
| External_Data_Source | Organisation_1 | IP_Address_1, IP_Address_2 | - | ORG1-RIPE_ID | - | - |
| Data_Source_File_3 | - | IP_Address_1 | org1 | - | - | - |
| Internal_Data_Source | Organization_1 | IP_Address_1 | - | - | - | ID: USER1, Name: User, Email: user@org1.com, Phone: user_phone |
| Data_Source_File_2 | - | - | - | - | poc@org1.com | ID: USER1, Name: User, Email: user@org1.com, Phone: user_phone |

ORG_2

Figure 3.6 - Web application dark theme (default theme)



ORG_1

| | Name | IP Addresses | Codename | RIPE ID | POC | Contacts |
|---|---|---|---|---|---|---|
| Data_Source_File_1 | Organisation_1 | - | org1 | ORG1-RIPE_ID | poc@org_1.com | - |
| External_Data_Source | Organisation_1 | IP_Address_1, IP_Address_2 | - | ORG1-RIPE_ID | - | - |
| Data_Source_File_3 | - | IP_Address_1 | org1 | - | - | - |
| Internal_Data_Source | Organization_1 | IP_Address_1 | - | - | - | ID: USER1, Name: User, Email: user@org1.com, Phone: user_phone |
| Data_Source_File_2 | - | - | - | - | poc@org1.com | ID: USER1, Name: User, Email: user@org1.com, Phone: user_phone |

ORG_2

Figure 3.7 - Web application light theme

# Chapter 4 – Testing and validation

This chapter provides an overview of the testing and validation processes to ensure that the application meets its functional and non-functional requirements. We developed test cases to assess the requirements established in Chapter 3, including expected outcomes that demonstrate the application's compliance with its specifications. Ultimately, FCCN validated the expected outcomes by confirming that the application functioned as intended.

## 4.1 Functional requirements test cases

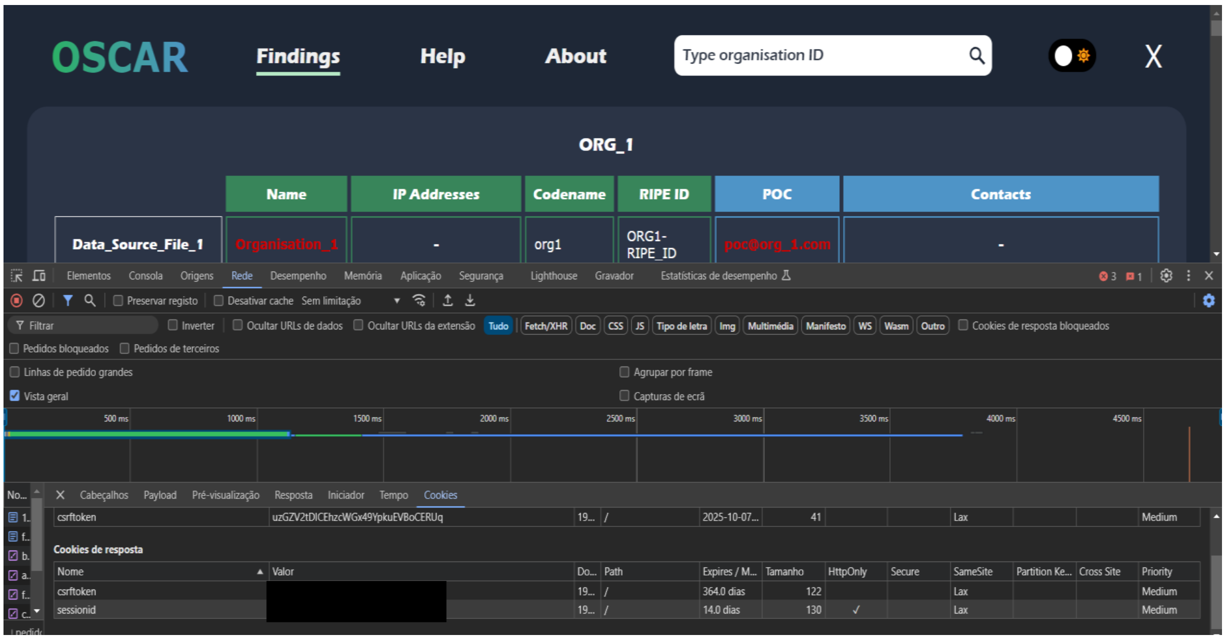
This section outlines the test cases developed in collaboration with FCCN to validate the application's functional requirements.

Table 4.1 - Functional requirements, test cases and expected outcomes

| Requirement | Test Case | Expected Outcome |
|---|---|---|
| User authentication | Verify the functionality of the authentication system | Valid users should log in successfully, while invalid credentials should be rejected with clear error messages |
| Scheduled data extraction and storage | Validate the automaticity of data extraction and storage | The application should automatically extract and store data at configurable intervals |
| Incongruent information detection functionality | Validate the detection of discrepancies between equivalent attributes | The system should accurately detect and highlight inconsistencies across the sources |
| Search functionality | Test search accuracy and response time. | The search results should be accurate and displayed in under 5 seconds |

### 4.1.1 User authentication test case outcome

To validate the user authentication test case, we conducted login attempts using both valid and invalid credentials. We observed the application's responses, specifically checking for appropriate error messages in cases of invalid credentials. Additionally, we utilized the developer console to verify session cookies or tokens that indicate a successful login.



ORG_1

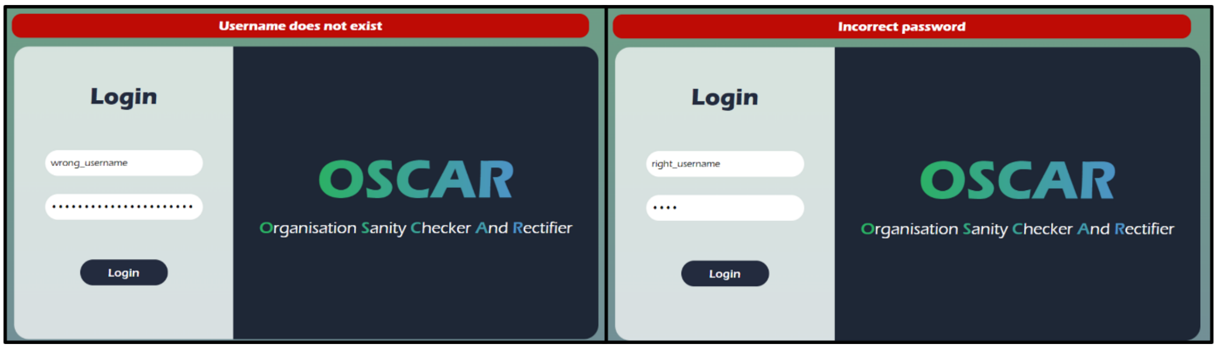
| | Name | IP Addresses | Codename | RIPE ID | POC | Contacts |
|---|---|---|---|---|---|---|
| Data_Source_File_1 | Organisation_1 | - | org1 | ORG1-RIPE_ID | poc@org_1.com | - |

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| csrftoken | uzGZV2tDICEhzcWGx49YpkuEVBoCERUq | 19... | / | 2025-10-07... | 41 | | | Lax | | Medium |

Cookies de resposta

| Nome | Valor | Do... | Path | Expires / M... | Tamanho | HttpOnly | Secure | SameSite | Partition Ke... | Cross Site | Priority |
|---|---|---|---|---|---|---|---|---|---|---|---|
| csrftoken | [redacted] | 19... | / | 364.0 dias | 122 | | | Lax | | | Medium |
| sessionid | | 19... | / | 14.0 dias | 130 | ✓ | | Lax | | | Medium |

Figure 4.1 - Confirmation of user authentication using the session cookie *sessionid*


Username does not exist

Login

wrong_username

••••••••••••••••••••••••

Login
OSCAR
Organisation Sanity Checker And Rectifier

Incorrect password

Login

right_username

••••

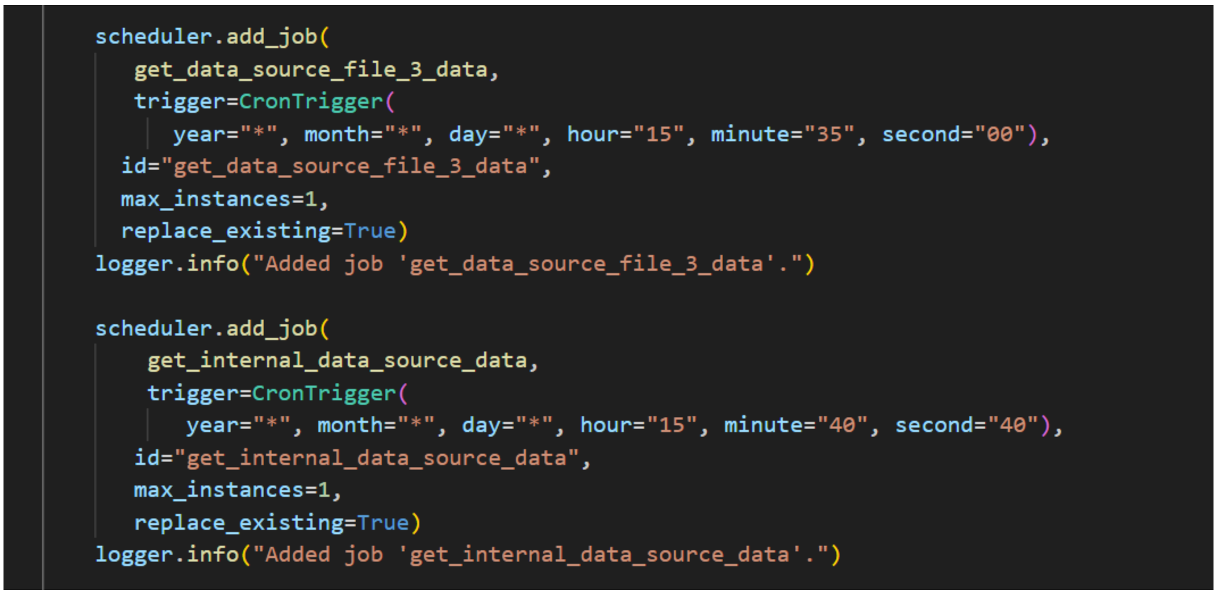
Login
OSCAR
Organisation Sanity Checker And Rectifier

Figure 4.2 - Left side: error message after attempting to log in with an invalid username, right side: error message after attempting to log in with an invalid password

We then concluded that the requirement was fulfilled, as the test results aligned with the expected outcome. Figure 4.1 confirmed successful user authentication, while the Figure 4.2 demonstrated that users with invalid credentials received appropriate error messages, effectively denying them access.

### 4.1.2 Scheduled data extraction and storage test case outcome

To validate the scheduled data extraction test case, we scheduled tasks for data extraction from each source and verified that the system automatically retrieved and updated the data at the specified intervals.



```
scheduler.add_job(
    get_data_source_file_3_data,
    trigger=CronTrigger(
        year="*", month="*", day="*", hour="15", minute="35", second="00"),
  id="get_data_source_file_3_data",
  max_instances=1,
  replace_existing=True)
logger.info("Added job 'get_data_source_file_3_data'.")

scheduler.add_job(
    get_internal_data_source_data,
    trigger=CronTrigger(
        year="*", month="*", day="*", hour="15", minute="40", second="40"),
  id="get_internal_data_source_data",
  max_instances=1,
  replace_existing=True)
logger.info("Added job 'get_internal_data_source_data'.")
```

Figure 4.3 - Scheduled data extraction tasks in “scheduler.py” with specified intervals for each source

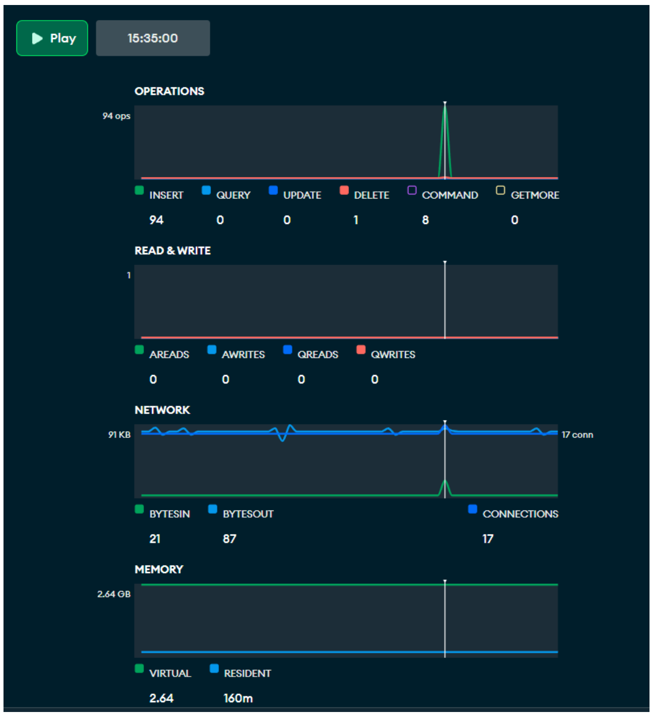
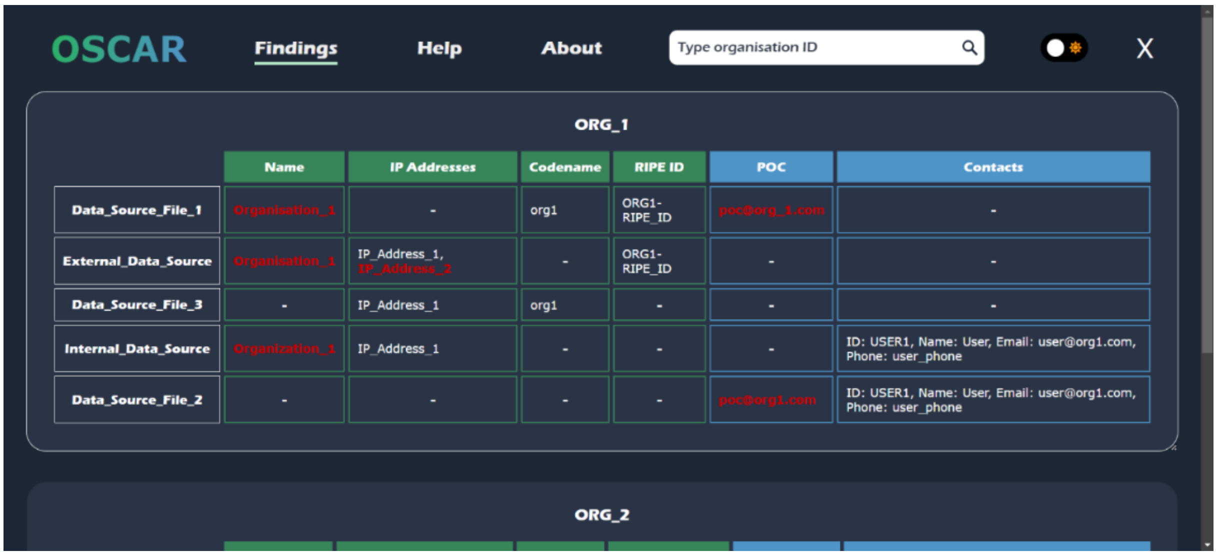


Figure 4.4 - MongoDB Compass showing insert and delete operations in the collections after data extraction

We concluded that the requirement was met, as the test outcomes corresponded with the expected results. The application demonstrated its ability to automatically synchronize data between the original data sources and the MongoDB database, effectively extracting and storing data at the configured intervals.

### 4.1.3 Incongruent information detection functionality test case outcome

We manually introduced discrepancies into equivalent organisational attributes across various test data sources to conduct the incongruent information detection functionality test case. We then ran the system to compare and identify these inconsistencies.



ORG_1

| | Name | IP Addresses | Codename | RIPE ID | POC | Contacts |
|---|---|---|---|---|---|---|
| Data_Source_File_1 | Organisation_1 | - | org1 | ORG1-RIPE_ID | poc@org_1.com | - |
| External_Data_Source | Organisation_1 | IP_Address_1, IP_Address_2 | - | ORG1-RIPE_ID | - | - |
| Data_Source_File_3 | - | IP_Address_1 | org1 | - | - | - |
| Internal_Data_Source | Organization_1 | IP_Address_1 | - | - | - | ID: USER1, Name: User, Email: user@org1.com, Phone: user_phone |
| Data_Source_File_2 | - | - | - | - | poc@org1.com | ID: USER1, Name: User, Email: user@org1.com, Phone: user_phone |

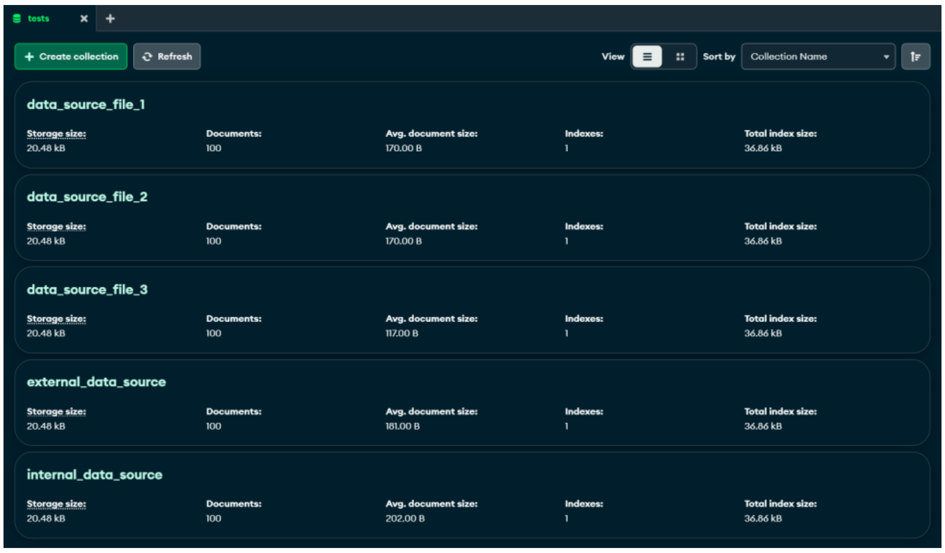
ORG_2

Figure 4.5 - Web interface displaying the organisation's information and highlighting the inconsistencies in the attributes from different data sources

We successfully achieved the expected outcome, as the system effectively detected and highlighted the inconsistencies between the attributes from the different data sources.

### 4.1.4 Search functionality test case outcome

To conduct the search functionality test case, we populated MongoDB with a dataset of 100 organisations and utilized the search bar in the web interface to locate a specific organisation. We also employed the browser's developer console (F12) to measure the time taken to return the search results.



| data_source_file_1 | | | | |
|---|---|---|---|---|
| Storage size: | Documents: | Avg. document size: | Indexes: | Total index size: |
| 20.48 kB | 100 | 170.00 B | 1 | 36.86 kB |

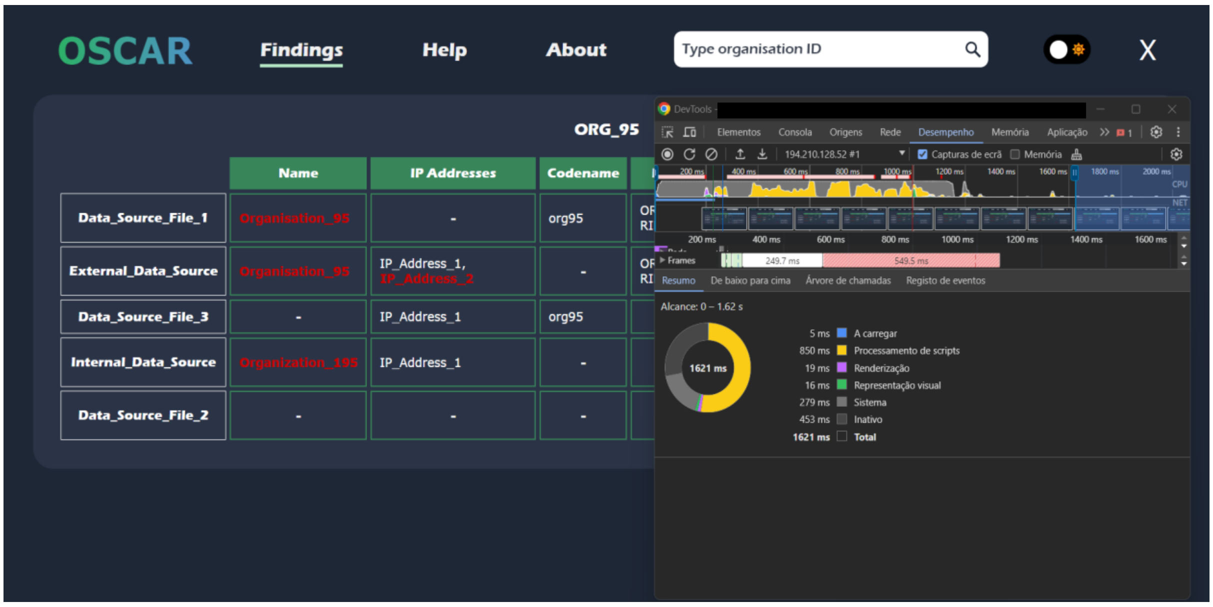
| data_source_file_2 | | | | |
|---|---|---|---|---|
| Storage size: | Documents: | Avg. document size: | Indexes: | Total index size: |
| 20.48 kB | 100 | 170.00 B | 1 | 36.86 kB |

| data_source_file_3 | | | | |
|---|---|---|---|---|
| Storage size: | Documents: | Avg. document size: | Indexes: | Total index size: |
| 20.48 kB | 100 | 117.00 B | 1 | 36.86 kB |

| external_data_source | | | | |
|---|---|---|---|---|
| Storage size: | Documents: | Avg. document size: | Indexes: | Total index size: |
| 20.48 kB | 100 | 181.00 B | 1 | 36.86 kB |

| internal_data_source | | | | |
|---|---|---|---|---|
| Storage size: | Documents: | Avg. document size: | Indexes: | Total index size: |
| 20.48 kB | 100 | 202.00 B | 1 | 36.86 kB |

Figure 4.6 - Test database displaying 100 documents (organisations) across each data source



ORG_95

| | Name | IP Addresses | Codename |
|---|---|---|---|
| Data_Source_File_1 | Organisation_95 | - | org95 |
| External_Data_Source | Organisation_95 | IP_Address_1, IP_Address_2 | - |
| Data_Source_File_3 | - | IP_Address_1 | org95 |
| Internal_Data_Source | Organization_195 | IP_Address_1 | - |
| Data_Source_File_2 | - | - | - |

Alcance: 0 – 1.62 s

| | |
|---|---|
| 5 ms | A carregar |
| 850 ms | Processamento de scripts |
| 19 ms | Renderização |
| 16 ms | Representação visual |
| 279 ms | Sistema |
| 453 ms | Inativo |
| **1621 ms** | **Total** |

Figure 4.7 - Total time taken by the search functionality to render organisation number 95, including a breakdown of the time spent on each step of the process

We successfully achieved the expected outcome, as the search returned the correct result in just 1.62 seconds, meeting the requirement of being under 5 seconds, as shown in Figure 4.8.

## 4.2 Non-functional tests

This section outlines the test cases developed in collaboration with FCCN to validate the non-functional requirements of the application.

Table 4.2 - Non-functional requirements, test cases and expected outcomes

| Requirement | Test Case | Expected Outcome |
|---|---|---|
| Usability | Verify the usability of the interface | Users should be able to navigate the application with ease, with minimal errors or confusion, and report that the interface is intuitive and user-friendly |
| Availability | Test the application's availability | The application should be always available to use and responsive throughout regular operations, without noticeable downtime |
| Performance | Measure the performance of the response times of the application | The application should efficiently process the data and return results within a reasonable time |
| Security | Verify data integrity and secure data storage (in transit and at rest) | The system should block unauthorized access attempts and secure data from common security threats, ensuring data integrity in transit and at rest |
| Cross-browser compatibility | Test the application on multiple browsers for compatibility | The application should work correctly and look the same across different browsers |
| Technology stack | Verify that the technology stack consists of free and open-source technologies | All components of the technology stack, including any third-party libraries or dependencies, should be licensed under free and open-source licenses |

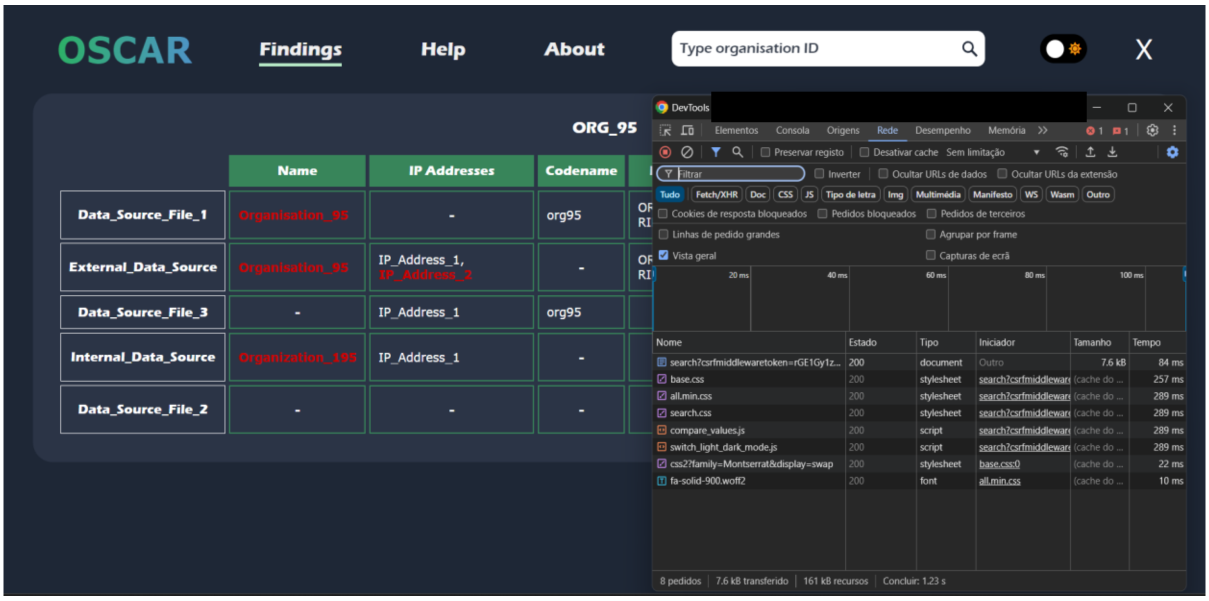
### 4.2.1 Usability test case outcome

We conducted the usability test case by asking the end-users to perform tasks such as logging in, searching for organisations, and viewing discrepancies. Afterwards, we collected their feedback on the ease of use, interface clarity, and any challenges encountered.

The feedback confirmed that the interface is intuitive and easy to use, meeting the usability requirements. Therefore, we concluded that the expected outcome was achieved, given that the positive response confirms the application's design effectiveness and user satisfaction.

### 4.2.2 Availability test case outcome

To perform the availability test case, we conducted typical user operations such as searching for organisation data and viewing discrepancies. We monitored HTTP status codes and network requests in real-time using the F12 Developer Tools Network tab, checking for any failed requests (e.g., 500 Internal Server Error, 404 Not Found) that could indicate availability issues.



ORG_95

| | Name | IP Addresses | Codename |
|---|---|---|---|
| Data_Source_File_1 | Organisation_95 | - | org95 |
| External_Data_Source | Organisation_95 | IP_Address_1, IP_Address_2 | - |
| Data_Source_File_3 | - | IP_Address_1 | org95 |
| Internal_Data_Source | Organization_195 | IP_Address_1 | - |
| Data_Source_File_2 | - | - | - |

| Nome | Estado | Tipo | Iniciador | Tamanho | Tempo |
|---|---|---|---|---|---|
| search?csrfmiddlewaretoken=rGE1Gy1z... | 200 | document | Outro | 7.6 kB | 84 ms |
| base.css | 200 | stylesheet | search?csrfmiddleware | (cache do ... | 257 ms |
| all.min.css | 200 | stylesheet | search?csrfmiddleware | (cache do ... | 289 ms |
| search.css | 200 | stylesheet | search?csrfmiddleware | (cache do ... | 289 ms |
| compare_values.js | 200 | script | search?csrfmiddleware | (cache do ... | 289 ms |
| switch_light_dark_mode.js | 200 | script | search?csrfmiddleware | (cache do ... | 289 ms |
| css2?family=Montserrat&display=swap | 200 | stylesheet | base.css:0 | (cache do ... | 22 ms |
| fa-solid-900.woff2 | 200 | font | all.min.css | (cache do ... | 10 ms |

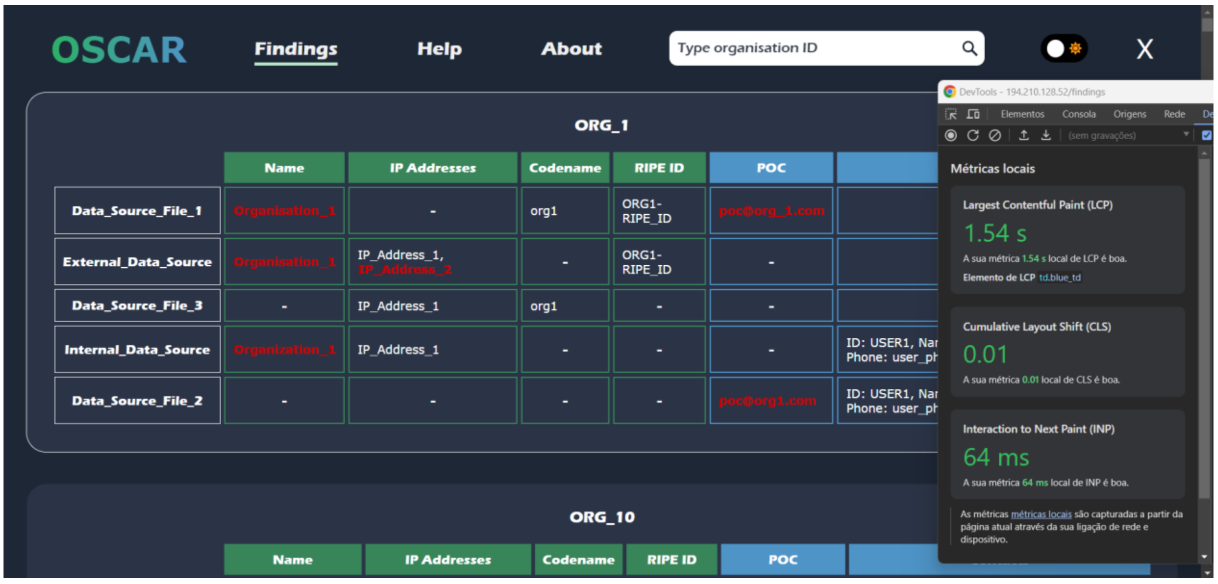
8 pedidos | 7.6 kB transferido | 161 kB recursos | Concluir: 1.23 s

Figure 4.8 - HTTP requests, status codes, and response times when searching for an organisation

We successfully achieved the expected outcome, as the application remained fully available during both the search and discrepancy viewing actions, consistently returning 200 OK and 304 Found status codes.

### 4.2.3 Performance test case outcome

To perform the performance test case, we evaluated the time required for the system to detect discrepancies and return search results. We measured key performance metrics, including Largest Contentful Paint (LCP), Cumulative Layout Shift (CLS), and Interaction to Next Paint (INP), to assess the application's efficiency.



DevTools - 194.210.128.52/findings

ORG_1

| | Name | IP Addresses | Codename | RIPE ID | POC | |
|---|---|---|---|---|---|---|
| Data_Source_File_1 | Organisation_1 | - | org1 | ORG1-RIPE_ID | poc@org_1.com | |
| External_Data_Source | Organisation_1 | IP_Address_1, IP_Address_2 | - | ORG1-RIPE_ID | - | |
| Data_Source_File_3 | - | IP_Address_1 | org1 | - | - | |
| Internal_Data_Source | Organization_1 | IP_Address_1 | - | - | - | ID: USER1, Nar Phone: user_ph |
| Data_Source_File_2 | - | - | - | - | poc@org1.com | ID: USER1, Nar Phone: user_ph |

ORG_10

| | Name | IP Addresses | Codename | RIPE ID | POC | |
|---|---|---|---|---|---|---|

Métricas locais

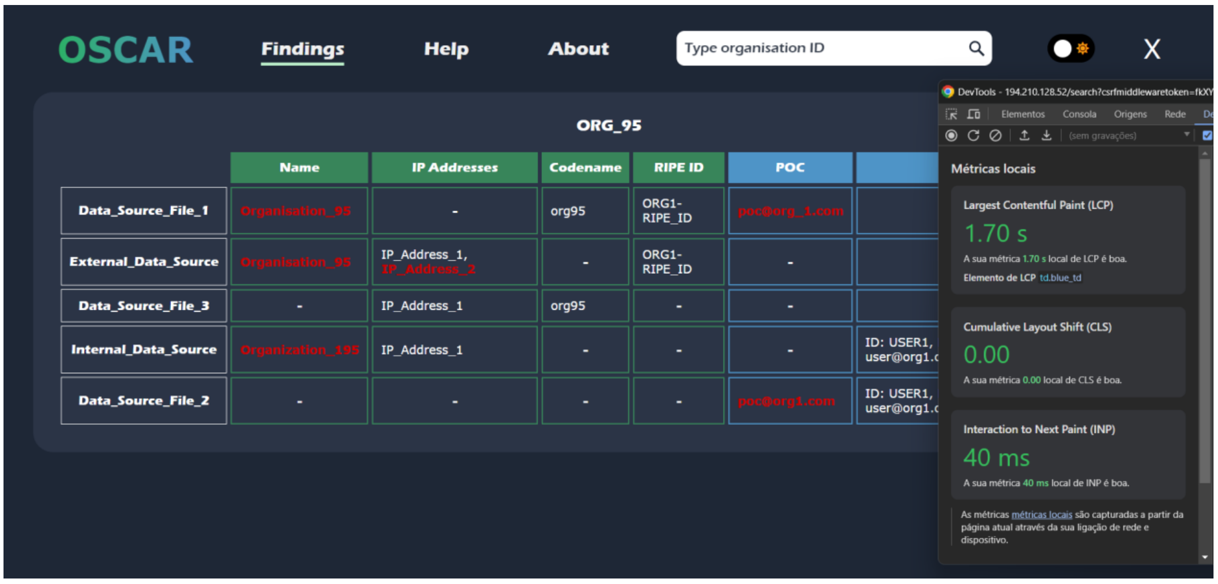
Largest Contentful Paint (LCP)
1.54 s
A sua métrica 1.54 s local de LCP é boa.
Elemento de LCP td.blue_td

Cumulative Layout Shift (CLS)
0.01
A sua métrica 0.01 local de CLS é boa.

Interaction to Next Paint (INP)
64 ms
A sua métrica 64 ms local de INP é boa.

As métricas métricas locais são capturadas a partir da página atual através da sua ligação de rede e dispositivo.

Figure 4.9 - Incongruent information detection functionality metrics results



DevTools - 194.210.128.52/search?csrfmiddlewaretoken=fkXY

ORG_95

| | Name | IP Addresses | Codename | RIPE ID | POC | |
|---|---|---|---|---|---|---|
| Data_Source_File_1 | Organisation_95 | - | org95 | ORG1-RIPE_ID | poc@org_1.com | |
| External_Data_Source | Organisation_95 | IP_Address_1, IP_Address_2 | - | ORG1-RIPE_ID | - | |
| Data_Source_File_3 | - | IP_Address_1 | org95 | - | - | |
| Internal_Data_Source | Organization_195 | IP_Address_1 | - | - | - | ID: USER1, user@org1.c |
| Data_Source_File_2 | - | - | - | - | poc@org1.com | ID: USER1, user@org1.c |

Métricas locais

Largest Contentful Paint (LCP)
1.70 s
A sua métrica 1.70 s local de LCP é boa.
Elemento de LCP td.blue_td

Cumulative Layout Shift (CLS)
0.00
A sua métrica 0.00 local de CLS é boa.

Interaction to Next Paint (INP)
40 ms
A sua métrica 40 ms local de INP é boa.

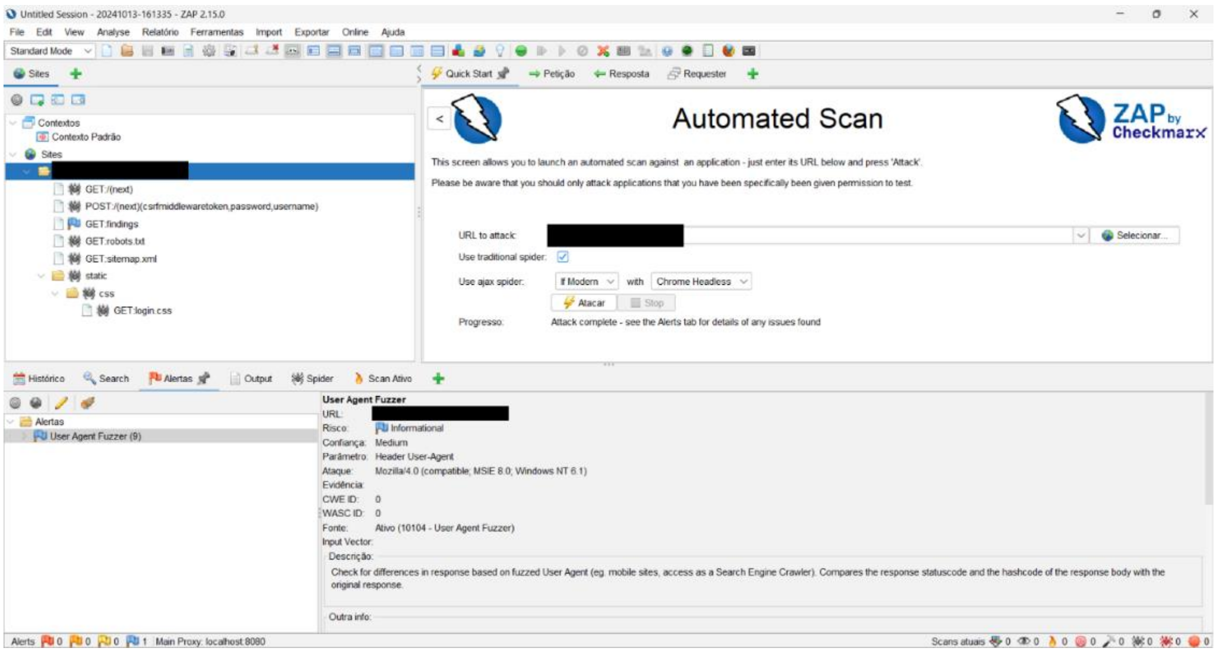
As métricas métricas locais são capturadas a partir da página atual através da sua ligação de rede e dispositivo.

Figure 4.10 - Search functionality metrics results

We successfully achieved the expected outcome, as the application efficiently processed the data and returned results within a reasonable time, as confirmed by the Developer Tools in Figures 4.10 and 4.11.

### 4.2.4 Security test case outcome

We performed the security test case to evaluate the system's ability to secure data in transit and at rest and detect vulnerabilities. Using OWASP ZAP, we conducted penetration tests targeting common vulnerabilities such as SQL injection and XSS attacks.



Contextos
Contexto Padrão
Sites
[redacted]
GET:/(next)
POST:/(next)(csrfmiddlewaretoken,password,username)
GET:findings
GET:robots.txt
GET:sitemap.xml
static
css
GET:login.css

This screen allows you to launch an automated scan against an application - just enter its URL below and press 'Attack'.

Please be aware that you should only attack applications that you have been specifically been given permission to test.

URL to attack: [redacted] Selecionar...
Use traditional spider: ✓
Use ajax spider: If Modern with Chrome Headless
Atacar Stop
Progresso: Attack complete - see the Alerts tab for details of any issues found

Alertas
User Agent Fuzzer (9)

**User Agent Fuzzer**
URL: [redacted]
Risco: Informational
Confiança: Medium
Parâmetro: Header User-Agent
Ataque: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1)
Evidência:
CWE ID: 0
WASC ID: 0
Fonte: Ativo (10104 - User Agent Fuzzer)
Input Vector:
Descrição:
Check for differences in response based on fuzzed User Agent (eg. mobile sites, access as a Search Engine Crawler). Compares the response statuscode and the hashcode of the response body with the original response.
Outra info:

Alerts 0 0 0 1 Main Proxy: localhost:8080
Scans atuais 0 0 0 0 0 0 0 0

Figure 4.11 - OWASP ZAP alerts section

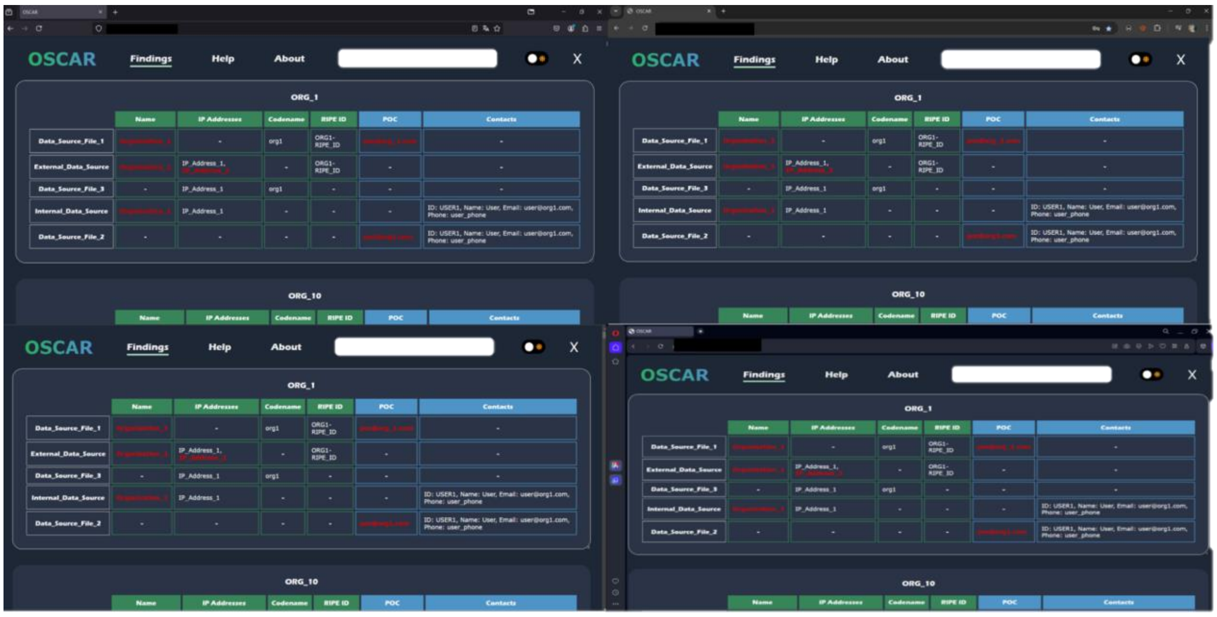
As shown in Figure 4.12, the alerts section revealed only informational alerts, which are not critical vulnerabilities but offer suggestions to enhance the application's security posture of the application. The use of HTTPS (Hypertext Transfer Protocol Secure) ensures that data in transit is secure, protecting it from interception during transmission.

Additionally, passwords are stored in the SQLite3 database using a hashing algorithm like PBKDF2, ensuring that sensitive information remains secure at rest. Both the organisational data stored in the MongoDB database, and the data in the SQLite3 database, are stored within FCCN's virtual machine. This is only accessible through a VPN, adding a layer of security for sensitive data at rest.

We reached the expected outcome, confirming that the application effectively secures data from common security threats, ensuring data integrity in transit and at rest.

### 4.2.5 Cross-browser compatibility test case outcome

We tested the application on popular browsers, including Chrome, Firefox, Opera and Safari to ensure consistent functionality and appearance.


[redacted]

ORG_1

| | Name | IP Addresses | Codename | RIPE ID | POC | Contacts |
|---|---|---|---|---|---|---|
| Data_Source_File_1 | [illegible] | - | org1 | ORG1-RIPE_ID | [illegible] | - |
| External_Data_Source | [illegible] | IP_Address_1, [illegible] | - | ORG1-RIPE_ID | - | - |
| Data_Source_File_3 | - | IP_Address_1 | org1 | - | - | - |
| Internal_Data_Source | [illegible] | IP_Address_1 | - | - | - | ID: USER1, Name: User, Email: user@org1.com, Phone: user_phone |
| Data_Source_File_2 | - | - | - | - | [illegible] | ID: USER1, Name: User, Email: user@org1.com, Phone: user_phone |

ORG_10

| Name | IP Addresses | Codename | RIPE ID | POC | Contacts |
|---|---|---|---|---|---|

Figure 4.12 - Incongruent information detection is displayed across different browsers (top left: Firefox, top right: Chrome, bottom left: Safari, bottom right: Opera)

The expected outcome was reached, as the application functions consistently and renders correctly the same content across the different browsers.

### 4.2.6 Technology stack test case outcome

We conducted the technology stack test case by collecting the names of all technologies used in the application and verifying their licensing information to ensure they are free and open-source. The verification process involved checking each technology's official websites and repositories to confirm their compliance with open-source requirements.

Table 4.3 - Technologies used in the development and testing of the web application, their availability as free and open-source software along with their respective licenses

| Technology | Free | Open Source | License |
|---|---|---|---|
| Python | Yes | Yes | Python Software Foundation |
| Django | Yes | Yes | BSD |
| Nginx | Yes | Yes | BSD |
| MongoDB | Yes | Yes | Server-Side Public License - SSPL |
| SQLite3 | Yes | Yes | Public Domain |
| OWASP ZAP | Yes | Yes | Apache License 2.0 |

We successfully achieved the expected outcome as all technologies used in the application were confirmed to be free and open-source, meeting the requirement of using a fully open-source technology stack.

# Chapter 5 – Conclusion

In this chapter, we summarise the key findings of our research and potential areas for improvement and enhancement discovered during the application's development and testing.

## 5.1 Conclusions

This dissertation focused on implementing a software-based system designed to address the research question: "Is it possible to detect and highlight incongruences in equivalent attribute values stored across several heterogeneous data sources, for effective incident triage in the information incident response process?"

The ability to identify discrepancies among equivalent data attributes is crucial in enhancing the accuracy and reliability of data used in incident response processes. By employing a system that identifies these discrepancies, we can facilitate timely and effective incident triage. While the literature explored provided insights into existing methodologies, particularly in how to correct discrepancies, it did not offer a comprehensive, ready-to-use solution tailored for detecting incongruences. The application developed in this dissertation focuses on detecting discrepancies, rather than correcting them, as we were able to develop the detection functionality, while the correction of these discrepancies remains a possible area for future improvement.

Throughout the research, several key requirements were established (Chapter 3, Section 3.2) that guided the application's development. These included the capability for scheduled data extraction, the efficient detection of incongruences, and an intuitive user interface to facilitate ease of use. The successful implementation of these features ensures that the system not only performs its intended functions but does so in a manner that is accessible and efficient for users.

The system has been tested in real operational settings by end users, which confirmed the outcomes of the analysis in the preceding chapter. The feedback received confirmed that the system successfully identifies and highlights incongruences between equivalent attribute values across various data sources, and fulfils the other established requirements, thus addressing the research question.

In summary, we conclude that the developed application has the potential to enhance the quality of the data used in RCTS CERT's incident response, by extracting data from heterogeneous sources and detecting incongruences. It also allows users to analyse the data directly through the interface.

## 5.2 Future work

Several promising avenues for future work could be explored to enhance the application such as leveraging machine learning techniques to suggest which data sources are most likely to contain correct values based on previous corrections. By storing historical correction data, the system could learn patterns and make informed predictions about the accuracy of incoming data.

Another potential improvement is implementing a functionality that enables users or the system to identify the correct values among all data sources and push corrections for identified discrepancies back to the originating sources via their respective APIs (Application Programming Interface), if they exist.

## References

A. Aborujilah, J. Adamu, S. M. Shariff and Z. Awang Long (2022). Descriptive Analysis of Built-in Security Features in Web Development Frameworks. 16th International Conference on Ubiquitous Information Management and Communication (IMCOM), 1-8. https://doi.org/10.1109/IMCOM53663.2022.9721750

Al-Baltah, I. A., Abd Ghani, A. A., Al-Gomaei, G. M., Abdulrazzak, F. H., & al Kharusi, A. A. (2020). A scalable semantic data fusion framework for heterogeneous sensor data. JOURNAL OF AMBIENT INTELLIGENCE AND HUMANIZED COMPUTING. https://doi.org/10.1007/s12652-020-02527-5

Boufares, F., & Salem, A. ben. (2012). Heterogeneous data integration and data quality: Overview of conflicts. 2012 6th International Conference on Sciences of Electronics, Technologies of Information and Telecommunications (SETIT), 867–874. https://doi.org/10.1109/SETIT.2012.6482029

Decreto Lei no 46/2018, de 13 de Agosto. Diário da República: I série, No 155 (2018). Accessed: Nov. 18, 2023. [Online]. Available: https://files.diariodarepublica.pt/1s/2018/08/15500/0403104037.pdf

Dumitrescu, S.-R., Branescu-Raspop, I., & Popescu, D. (2013). Format mediation for matching patient records in a three-layer information architecture. 2013 4th International Symposium on Electrical and Electronics Engineering (ISEEE), 1–5. https://doi.org/10.1109/ISEEE.2013.6674369

El-Demerdash, K. K., & Amer, F. (2012). Data integration framework with an application for Ministry of Interior. 2012 8th International Conference on Informatics and Systems (INFOS), DE-18-DE-28.

From Good Practice Guide for Incident Management. (2010, December 20). ENISA. https://www.enisa.europa.eu/publications/good-practice-guide-for-incident-management

Ghorbel, L., Zayani, C. A., & Amous, I. (2015). A novel architecture for learner's profiles interoperability. 2015 IEEE/ACIS 14th International Conference on Computer and Information Science (ICIS), 405–410. https://doi.org/10.1109/ICIS.2015.7166628

Kitchenham, B., Pearl Brereton, O., Budgen, D., Turner, M., Bailey, J., & Linkman, S. (2009). Systematic literature reviews in software engineering – A systematic literature review. Information and Software Technology, 51(1), 7–15. https://doi.org/10.1016/j.infsof.2008.09.009

Mirza, A., & Siddiqi, I. (2017). Data level conflicts resolution for multi-sources heterogeneous databases. 2016 6th International Conference on Innovative Computing Technology, INTECH 2016, 36–40. https://doi.org/10.1109/INTECH.2016.7845088

Mirza, G. A. (2014). Value name conflict while integrating data indatabase integration. In S. X. Yang, J. P. Li, I. Bloshanskii, & I. Ahmad (Eds.), 2014 11th International Computer Conference on Wavelet Active Media Technology and Information Processing, ICCWAMTIP 2014 (pp. 316–320). Institute of Electrical and Electronics Engineers Inc. https://doi.org/10.1109/ICCWAMTIP.2014.7073417

Mirza, G. A. (2016). Null Value Conflict: Formal Definition and Resolution. Proceedings - 2015 13th International Conference on Frontiers of Information Technology, FIT 2015, 132–137. https://doi.org/10.1109/FIT.2015.32

Nachouki, G., & Quafafou, M. (2011). MashUp web data sources and services based on semantic queries. INFORMATION SYSTEMS, 36(2), 151–173. https://doi.org/10.1016/j.is.2010.08.001

Page, M. J., McKenzie, J. E., Bossuyt, P. M., Boutron, I., Hoffmann, T., Mulrow, C. D., Shamseer, L., Tetzlaff, J., Akl, E. A., Brennan, S., Chou, R., Glanville, J., Grimshaw, J., Hróbjartsson, A., Lalu, M. M., Li, T., Loder, E., Mayo-Wilson, E., McDonald, S., . . . Moher, D. (2021). The PRISMA 2020 statement: an updated guideline for reporting systematic reviews. The BMJ, n71. https://doi.org/10.1136/bmj.n71

Peffers, K., Tuunanen, T., Gengler, C. E., Rossi, M., Hui, W., Virtanen, V., & Bragge, J. (2020, June 4). Design Science Research Process: A Model for Producing and Presenting Information Systems Research. http://arxiv.org/abs/2006.02763

Qutaany, A. Z. el, Bastawissy, A. H. el, & Hegazy, O. (2010). A technique for mutual inconsistencies detection and resolution in virtual data integration environment. 2010 The 7th International Conference on Informatics and Systems (INFOS), 1–8.

Saranya, K., Hema, M. S., & Chandramathi, S. (2014). Data fusion in ontology based data integration. International Conference on Information Communication and Embedded Systems (ICICES2014), 1–6. https://doi.org/10.1109/ICICES.2014.7033792

Sharef, N. M., Shafazand, Y. M., Nazri, M. Z. A., & Husin, N. A. (2018). Self-adaptive Based Model for Ambiguity Resolution of The Linked Data Query for Big Data Analytics. INTERNATIONAL JOURNAL OF INTEGRATED ENGINEERING, 10(6), 176–182. https://doi.org/10.30880/ijie.2018.10.06.025

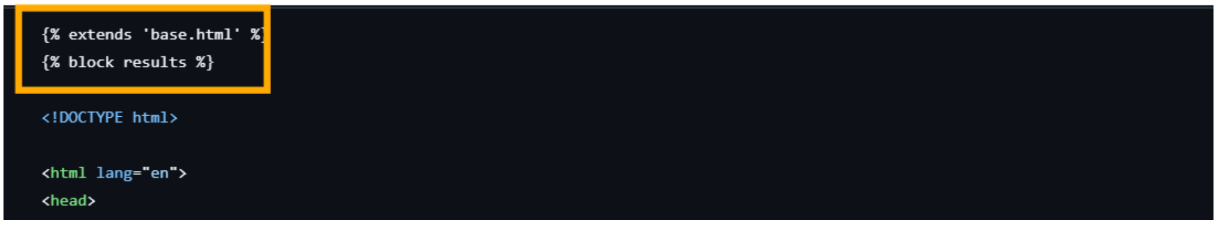
Sonsilphong, S., Arch-int, N., Arch-int, S., & Pattarapongsin, C. (2016). A semantic interoperability approach to health-care data: Resolving data-level conflicts. EXPERT SYSTEMS, 33(6), 531–547. https://doi.org/10.1111/exsy.12167

Wei, D., & Xianxia, Z. (2013). The Method to Resolve Schema Discrepancy of a Heterogeneous Database System. 2013 IEEE 10th International Conference on High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing, 2342–2346. https://doi.org/10.1109/HPCC.and.EUC.2013.337

W. M. C. J. T. Kithulwatta, K. P. N. Jayasena, B. T. G. S. Kumara and R. M. K. T. Rathnayaka, (2022). Performance Evaluation of Docker-based Apache and Nginx Web Server. 3rd International Conference for Emerging Technology (INCET), 1-6. https://doi.org/10.1109/INCET54531.2022.9824303

## Appendix A: Django template tags

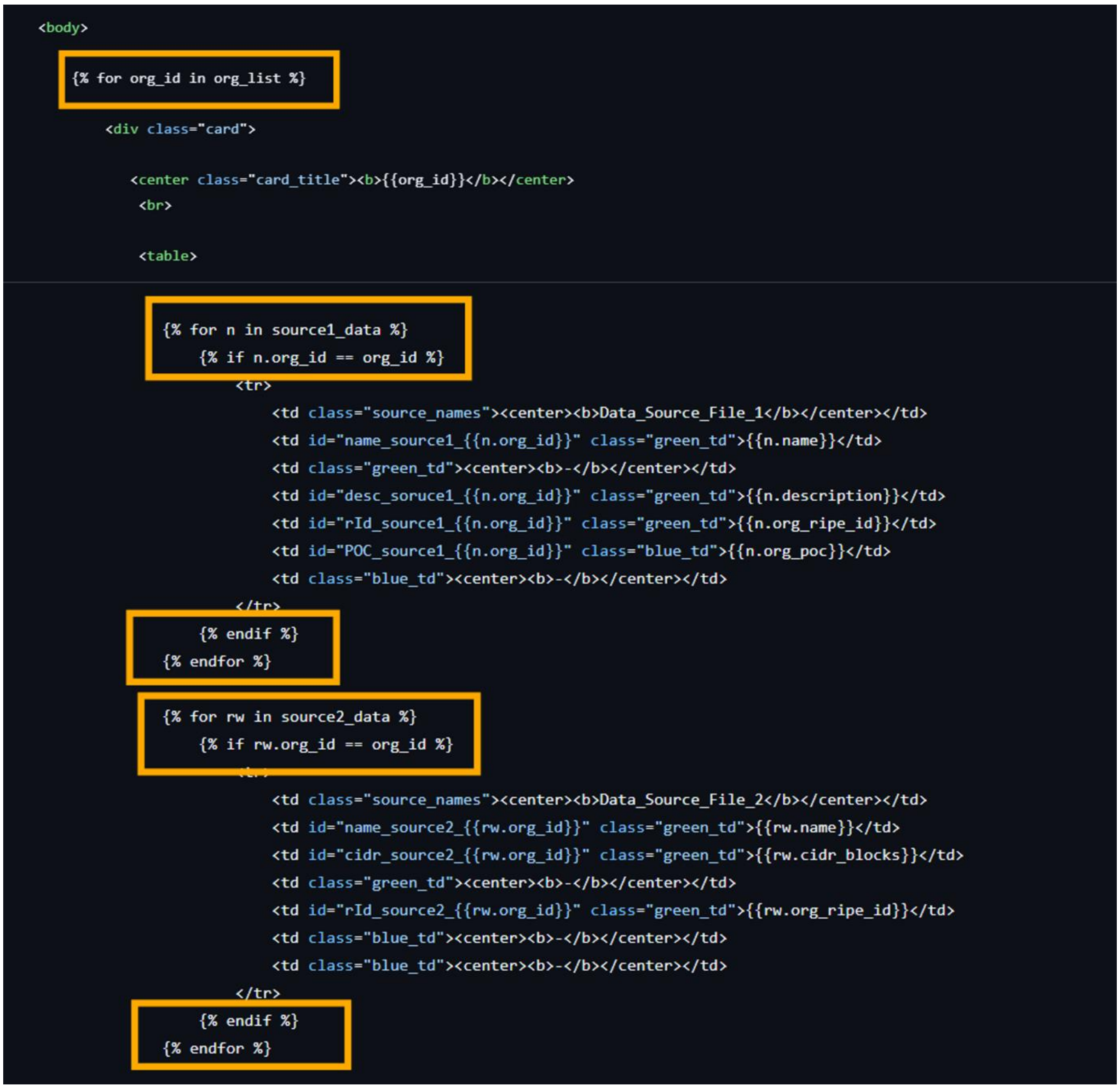
The tag *{% extends 'base.html' %}* indicates that the current template will inherit from base.html. By using this inheritance each specific template can provide its unique content without rewriting the entire HTML structure. The tag *{% block results %}* defines a block named results, where we insert content specific to the current template. In this case the content within this block will be dynamically generated based on the data provided to the template.



```
{% extends 'base.html' %}
{% block results %}

<!DOCTYPE html>

<html lang="en">
<head>
```

Figure A.1 - Template inheritance and block definition tags

The *{% for org_id in org_list %}* loop iterates through a list called *org_list*, which is passed to the template from a Django view. For each *org_id*, the inner loops then check against *source1_data* and *source2_data* to find and display relevant information. The *{% if n.org_id == org_id %}* and *{% if rw.org_id == org_id %}* statements serve to filter and display only the data relevant to the current organisation ID being processed, linking *source1_data* and *source2_data* to their respective organisations. Each organisation data entry is rendered using placeholders such as *{{n.name}}* and *{{rw.name}}*, which access attributes of the objects in data lists *source1_data* and *source2_data*.

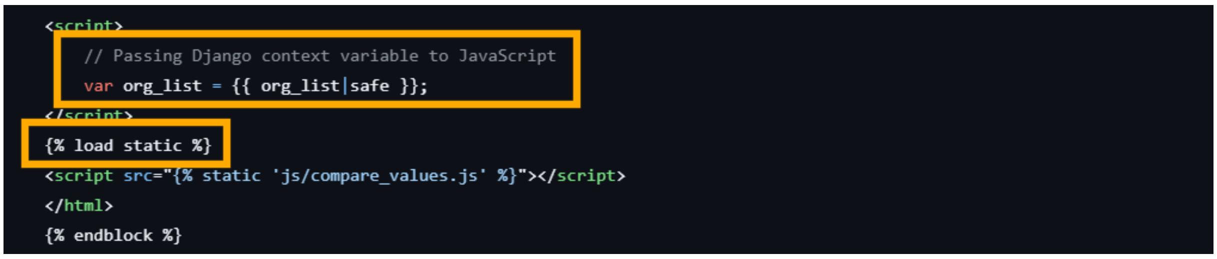


```
<body>

  {% for org_id in org_list %}

        <div class="card">

          <center class="card_title"><b>{{org_id}}</b></center>
           <br>

           <table>

            {% for n in source1_data %}
                {% if n.org_id == org_id %}
                    <tr>
                        <td class="source_names"><center><b>Data_Source_File_1</b></center></td>
                        <td id="name_source1_{{n.org_id}}" class="green_td">{{n.name}}</td>
                        <td class="green_td"><center><b>-</b></center></td>
                        <td id="desc_soruce1_{{n.org_id}}" class="green_td">{{n.description}}</td>
                        <td id="rId_source1_{{n.org_id}}" class="green_td">{{n.org_ripe_id}}</td>
                        <td id="POC_source1_{{n.org_id}}" class="blue_td">{{n.org_poc}}</td>
                        <td class="blue_td"><center><b>-</b></center></td>
                    </tr>
                {% endif %}
            {% endfor %}

            {% for rw in source2_data %}
                {% if rw.org_id == org_id %}
                    <tr>
                        <td class="source_names"><center><b>Data_Source_File_2</b></center></td>
                        <td id="name_source2_{{rw.org_id}}" class="green_td">{{rw.name}}</td>
                        <td id="cidr_source2_{{rw.org_id}}" class="green_td">{{rw.cidr_blocks}}</td>
                        <td class="green_td"><center><b>-</b></center></td>
                        <td id="rId_source2_{{rw.org_id}}" class="green_td">{{rw.org_ripe_id}}</td>
                        <td class="blue_td"><center><b>-</b></center></td>
                        <td class="blue_td"><center><b>-</b></center></td>
                    </tr>
                {% endif %}
            {% endfor %}
```

Figure A.2 - Iteration, conditional and dynamic content rendering tags

By using *{{ org_list|safe }}*, we instruct Django to insert the value of *org_list* directly into the JavaScript code without escaping it. The *|safe* filter tells Django that the content is safe for rendering as-is, allowing for proper JavaScript syntax (such as arrays or objects) without escaping quotes or special characters. This is essential for making server-side data available to client-side scripts, enabling dynamic functionality based on the contents of the *org_list.*

The *{% load static %}* allows us to access static files, such as CSS, JavaScript, and images that are not generated dynamically. Using this tag ensures that our static files are served correctly regardless of the deployment environment, by abstracting away the URL structure. This allows for easier maintenance and updates to the static assets.

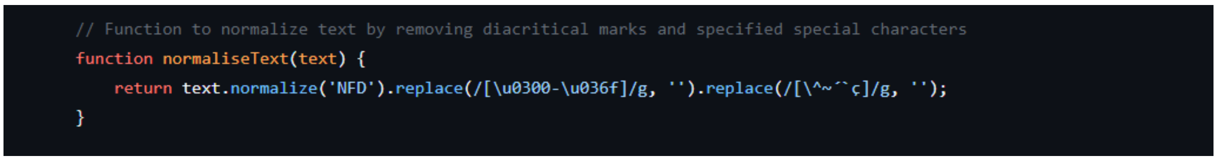


```
<script>
    // Passing Django context variable to JavaScript
    var org_list = {{ org_list|safe }};
</script>
{% load static %}
<script src="{% static 'js/compare_values.js' %}"></script>
</html>
{% endblock %}
```

Figure A.3 - Context variables and loading static files tags

## Appendix B: Highlighting logic for the organisation's attributes

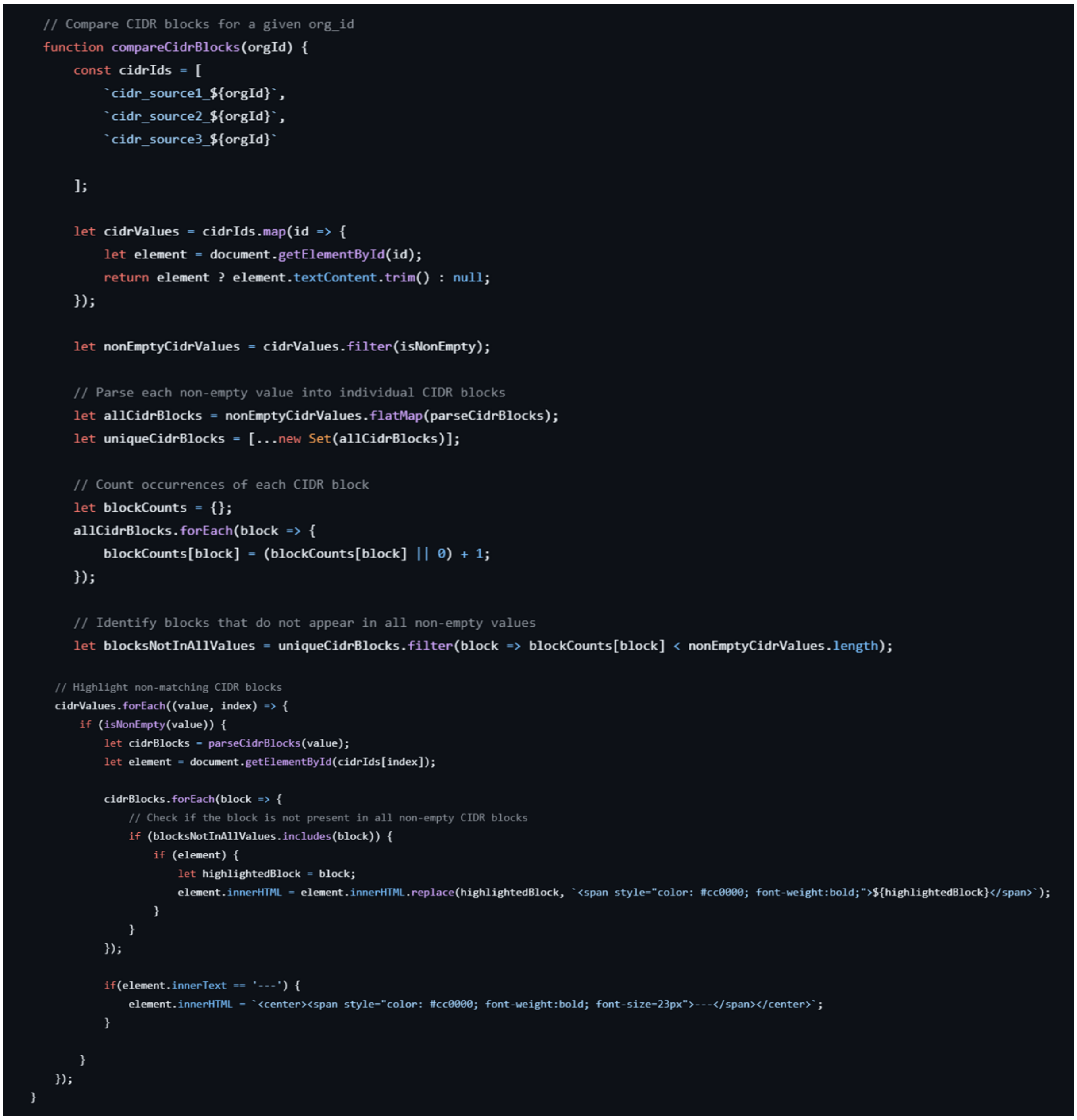
The function normaliseText() standardises text by removing diacritical marks (accents) and specific special characters to ensure consistent comparisons across sources. Using unicode normalisation NFD (Normalisation Form Decomposition), it decomposes characters with diacritical marks into base characters and removes accents using a regex expression. Additional characters like ^, ~, ´, `, and ç are also removed to prevent mismatches due to minor variations in text formatting. This is particularly useful for ensuring accurate comparisons of names and other non-technical text fields.



```
// Function to normalize text by removing diacritical marks and specified special characters
function normaliseText(text) {
    return text.normalize('NFD').replace(/[\u0300-\u036f]/g, '').replace(/[\^~´`ç]/g, '');
}
```

Figure B.1 - normaliseText() function in JavaScript

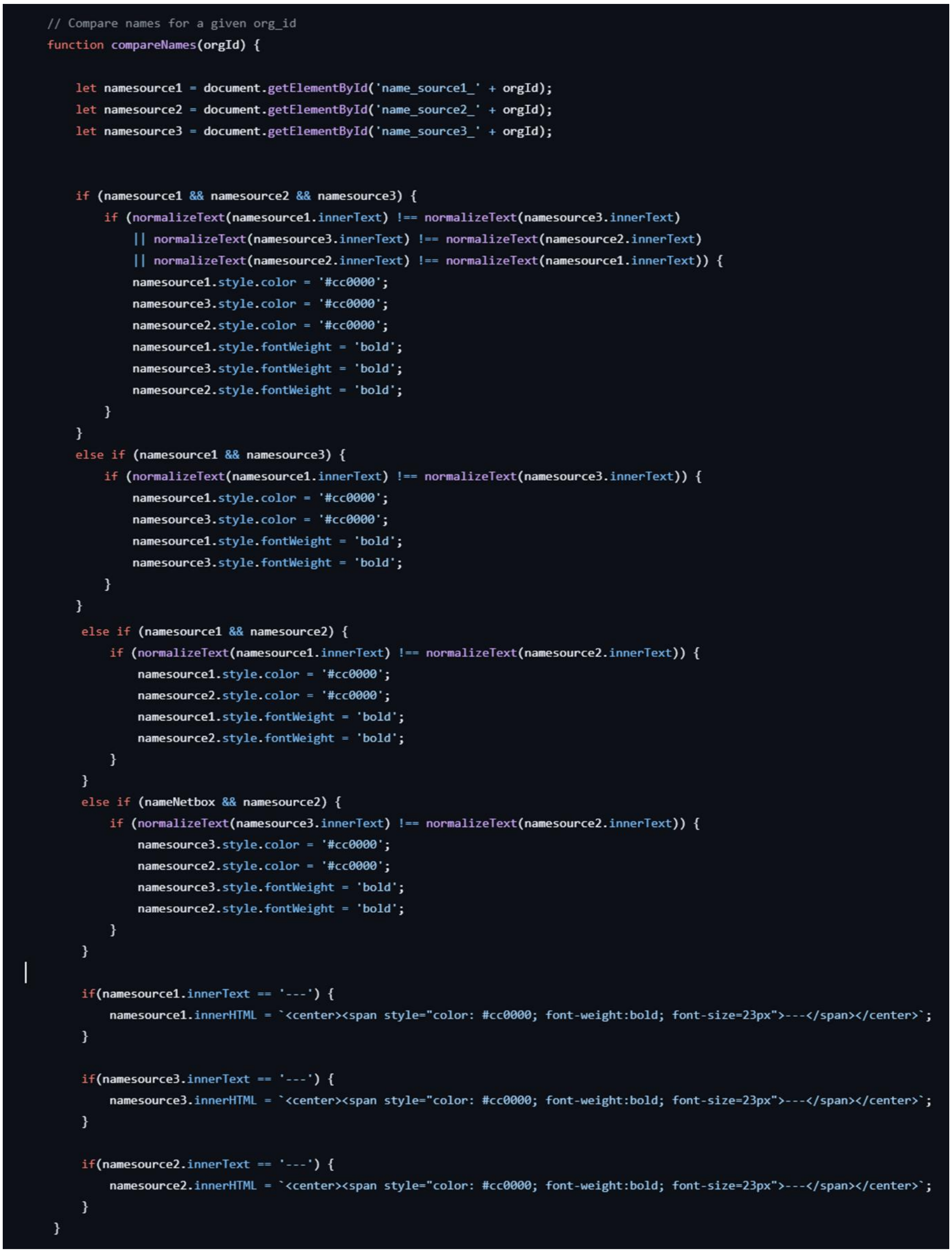
This function compares IP addresses across multiple data sources for a given organisation, specified by an organisation ID. It retrieves IP address values from each data source, splitting them into individual IP addresses. By counting occurrences of each address, the function identifies the ones that are not universally present across all sources. If an IP address is found in some sources but not others, it is flagged as inconsistent, and the function highlights these discrepancies directly in the HTML elements associated with each data source for the organisation.



```javascript
  // Compare CIDR blocks for a given org_id
  function compareCidrBlocks(orgId) {
      const cidrIds = [
          `cidr_source1_${orgId}`,
          `cidr_source2_${orgId}`,
          `cidr_source3_${orgId}`

      ];

      let cidrValues = cidrIds.map(id => {
          let element = document.getElementById(id);
          return element ? element.textContent.trim() : null;
      });

      let nonEmptyCidrValues = cidrValues.filter(isNonEmpty);

      // Parse each non-empty value into individual CIDR blocks
      let allCidrBlocks = nonEmptyCidrValues.flatMap(parseCidrBlocks);
      let uniqueCidrBlocks = [...new Set(allCidrBlocks)];

      // Count occurrences of each CIDR block
      let blockCounts = {};
      allCidrBlocks.forEach(block => {
          blockCounts[block] = (blockCounts[block] || 0) + 1;
      });

      // Identify blocks that do not appear in all non-empty values
      let blocksNotInAllValues = uniqueCidrBlocks.filter(block => blockCounts[block] < nonEmptyCidrValues.length);

    // Highlight non-matching CIDR blocks
    cidrValues.forEach((value, index) => {
        if (isNonEmpty(value)) {
            let cidrBlocks = parseCidrBlocks(value);
            let element = document.getElementById(cidrIds[index]);

            cidrBlocks.forEach(block => {
                // Check if the block is not present in all non-empty CIDR blocks
                if (blocksNotInAllValues.includes(block)) {
                    if (element) {
                        let highlightedBlock = block;
                        element.innerHTML = element.innerHTML.replace(highlightedBlock, `<span style="color: #cc0000; font-weight:bold;">${highlightedBlock}</span>`);
                    }
                }
            });

            if(element.innerText == '---') {
                element.innerHTML = `<center><span style="color: #cc0000; font-weight:bold; font-size=23px">---</span></center>`;
            }

        }
    });
}
```

Figure B.2 - compareCidrBlocks() function in JavaScript

The function compareNames() compares the name attribute of an organisation across data sources by retrieving each name value based on an organisation ID. The same logic used in this function is applied to the other functions that compare other attributes. Using the normaliseText() function, it standardises each name value to avoid mismatches caused by formatting differences. If there is any variation in the names across sources, it applies a highlight to each name field for that organisation, signalling inconsistency. Additionally, it handles cases where a name field is marked as missing (---), centring the text and ensuring a consistent visual representation for absent values.



```javascript
// Compare names for a given org_id
function compareNames(orgId) {

    let namesource1 = document.getElementById('name_source1_' + orgId);
    let namesource2 = document.getElementById('name_source2_' + orgId);
    let namesource3 = document.getElementById('name_source3_' + orgId);


    if (namesource1 && namesource2 && namesource3) {
        if (normalizeText(namesource1.innerText) !== normalizeText(namesource3.innerText)
            || normalizeText(namesource3.innerText) !== normalizeText(namesource2.innerText)
            || normalizeText(namesource2.innerText) !== normalizeText(namesource1.innerText)) {
            namesource1.style.color = '#cc0000';
            namesource3.style.color = '#cc0000';
            namesource2.style.color = '#cc0000';
            namesource1.style.fontWeight = 'bold';
            namesource3.style.fontWeight = 'bold';
            namesource2.style.fontWeight = 'bold';
        }
    }
    else if (namesource1 && namesource3) {
        if (normalizeText(namesource1.innerText) !== normalizeText(namesource3.innerText)) {
            namesource1.style.color = '#cc0000';
            namesource3.style.color = '#cc0000';
            namesource1.style.fontWeight = 'bold';
            namesource3.style.fontWeight = 'bold';
        }
    }
     else if (namesource1 && namesource2) {
         if (normalizeText(namesource1.innerText) !== normalizeText(namesource2.innerText)) {
             namesource1.style.color = '#cc0000';
             namesource2.style.color = '#cc0000';
             namesource1.style.fontWeight = 'bold';
             namesource2.style.fontWeight = 'bold';
         }
     }
     else if (nameNetbox && namesource2) {
         if (normalizeText(namesource3.innerText) !== normalizeText(namesource2.innerText)) {
             namesource3.style.color = '#cc0000';
             namesource2.style.color = '#cc0000';
             namesource3.style.fontWeight = 'bold';
             namesource2.style.fontWeight = 'bold';
         }
     }

     if(namesource1.innerText == '---') {
         namesource1.innerHTML = `<center><span style="color: #cc0000; font-weight:bold; font-size=23px">---</span></center>`;
     }

     if(namesource3.innerText == '---') {
         namesource3.innerHTML = `<center><span style="color: #cc0000; font-weight:bold; font-size=23px">---</span></center>`;
     }

     if(namesource2.innerText == '---') {
         namesource2.innerHTML = `<center><span style="color: #cc0000; font-weight:bold; font-size=23px">---</span></center>`;
     }
 }
```

Figure B.3 - compareNames() function in JavaScript